NASA/CP—2000-209888

Eighth Goddard Conference on Mass Storage Systems and Technologies

in cooperation with the

Seventeenth IEEE Symposium on Mass Storage Systems

Edited by
Benjamin Kobler, Goddard Space Flight Center, Greenbelt, Maryland
P C Hariharan, Systems Engineering and Security, Inc., Greenbelt, Maryland

Proceedings of a conference held at
The Inn and Conference Center
University of Maryland, University College
College Park, Maryland, USA
March 27–30, 2000

National Aeronautics and
Space Administration

Goddard Space Flight Center
Greenbelt, Maryland 20771

March 2000

NASA/CP—2000–209888

Eighth Goddard Conference on Mass Storage Systems and Technologies

in cooperation with the

Seventeenth IEEE Symposium on Mass Storage Systems

Edited by
Benjamin Kobler, Goddard Space Flight Center, Greenbelt, Maryland
P C Hariharan, Systems Engineering and Security, Inc., Greenbelt, Maryland

Proceedings of a conference held at
The Inn and Conference Center
University of Maryland, University College
College Park, Maryland, USA
March 27–30, 2000

National Aeronautics and
Space Administration

Goddard Space Flight Center
Greenbelt, Maryland 20771

March 2000

Available from:

NASA Center for AeroSpace Information
7121 Standard Drive
Hanover, MD 21076-1320
Price Code: A17

National Technical Information Service
5285 Port Royal Road
Springfield, VA 22161
Price Code: A10

Preface

This volume collects together 37 papers from the Eighth Goddard Conference on Mass Storage Systems and Technologies; for the third succeeding year, it is being held in cooperation with the IEEE Symposium on Mass Storage Systems and Technologies and is therefore also the Seventeenth IEEE Symposium on Mass Storage Systems.

The tutorials on the first day cover storage tuning, file systems, storage area networks (SAN), network video storage servers and the stability of optical disk media. Over the following three days of the Conference, there are twelve sessions on various themes: Performance, Data Management, File Systems, Emerging Technologies, Site Reports, and Standards.

An invited panel on the third day will consider the future of current mass storage technologies. There is intense development work and competition in the disk and tape fields, both magnetic and optical. Magnetic hard disk continues to double its areal density about every year, and is well ahead of tape in this respect. Tapes on the other hand, are still the media of choice for archives, principally because of their volumetric efficiency. Optical disk media maintain the edge in track density. For distribution, CD and DVD have the advantage because they are replicated using an injection-molding process. Reliable copies can be made with less labor than for any other technology in less than 4 sec per copy. Multi-layer recording is already part of the DVD specifications, and there are other technologies over the horizon, (*e.g.,* MFD - multi-layer fluorescent disc) which promise up to 16 layers on a disc. The newer optical discs are already in the same league as high-end tapes in volumetric density, and may well overtake tapes in this respect. In the tape world, there is more activity in longitudinal recording, as exemplified by the *Linear Tape Open* consortium. Experts in the fields of magnetic tape, magnetic disk and optical media present a concise overview of these aspects of the competing technologies with the moderator and the audience joining in with their questions, views, comments and observations.

Denis Gabor, who was awarded the Nobel Prize in Physics in 1971, invented holography (from the Greek *holos* whole and Latin and Greek *–graphos* written, writing, drawing) in 1949. In holography, one stores both the amplitude and the phase of the wavefront emanating from an object. Efforts to develop practical storage devices using holographic techniques have been under way since the 70's, and there were papers describing attempts to design holographic storage devices in the IEEE Symposia in the late 80's. Market and economic factors, as well as developments in materials science, have reached a stage where it now appears that holography may be another option for affordable data storage. The Program Committee decided to invite a group of research teams to report on their work towards achieving the goal of three-dimensional storage during a special invited session on holography held on the last day of the Conference.

In view of the limited time available in the general sessions, a poster session has been slotted to make available to the audience many of the papers that did not fit into the themes of the plenary sessions. These poster papers have also been included in this publication.

Vendor exhibits will continue through the three days of the general sessions. On the final day, some vendors will talk about their products.

The Program Committee has worked diligently with the authors of the papers to assist the editors in the production of this volume. Rodney Van Meter deserves particular thanks for the Perl script, which made for painless pagination of Postscript files.

Ben Kobler
P C Hariharan

Table of Contents

Tutorials

Performance

Data Management

File Systems 1

Posters

Emerging Technologies

Invited Panel: The Future of Current Mass Storage Technologies

Site Reports

File Systems 2

Standards

Nascent Promises: Holographic Storage

Disk Subsystem Performance Evaluation: From Disk Drives to Storage Area Networks

Thomas M. Ruwart
University of Minnesota
Laboratory for Computational Science and Engineering
356 Physics
116 Church Street SE
Minneapolis, Minnesota 55455
tmr@tc.umn.edu
tel +1 612 626-8091
fax +1 612 626-0030

Abstract

Disk subsystems span the range of configuration complexity from single disk drives to large installations of disk arrays. They can be directly attached to individual computer systems or configured as larger, shared access Storage Area Networks (SANs). It is a significant task to evaluate the performance of these subsystems especially when considering the range of performance requirements of any particular installation and application. Storage subsystems can be designed to meet different performance criteria such as bandwidth, transactions per second, latency, capacity, connectivity, ...etc. but the question of *how* the subsystem will perform depends on the software and hardware *layering* and the number of layers an I/O request must traverse in order to perform the actual operation. As an I/O request traverses more and more software and hardware layers, alignment and request size fragmentation can result in performance anomalies that can degrade the overall bandwidth and transaction rates. Layer traversal can have a significant negative impact on the observed performance of even the fastest hardware components. This paper walks through the Storage Subsystem Hierarchy, defining these layers, presents a method for testing in single and multiple computer environments, and demonstrates the significance of careful, in-depth evaluation of Storage Subsystem Performance.

1 Introduction

Disk subsystem manufacturers make many claims about the performance of their products. However, these performance claims cannot be taken out of context of the final implementation. Rather, it is necessary to evaluate the performance of disk subsystems within a configuration that is as close as possible to the actual configuration in which the subsystem will ultimately be employed. Such an evaluation requires a benchmark program that can closely mimic the access patterns of the intended applications and provide results that are *meaningful* and *reproducible.*

This paper presents examples of disk I/O performance anomalies and describes the cause of these problems as well as strategies to minimize their effects. The paper begins by describing the hardware and software components that an I/O request must traverse in order to move data between the computer system memory and the storage media. The Testing Philosophy and Methodology is then presented that describes how and why the individual components are evaluated as well as basic assumptions and tradeoffs that must be made in order to provide meaningful and reproducible results. The performance of the entire Storage Subsystem Hierarchy is then be evaluated under ideal conditions. This sets an upper bound for performance of the system as a whole. Knowing this upper performance limit, it is possible to address the *Impedance Matching Problem* which examines various performance anomalies and their causes. Example test results are given throughout the paper to illustrate relevant concepts.

2 The Storage Subsystem Hierarchy

The Storage Subsystem Hierarchy describes the levels of hardware and software that an I/O request must traverse in order to initiate and manage the movement of data between the application memory space and a storage device. The I/O request is initiated by the application when data movement is required either explicitly in the case of file operations or implicitly in the case of memory-mapped files for example. The request is initially processed by several layers of system software such as the file system manager, logical device drivers, and the low-level hardware device drivers. During this processing the application I/O request may be split into several inter-related "physical" I/O requests that are subsequently sent out to the appropriate storage devices to satisfy these requests. These physical I/O requests must pass through the Physical Connection Layer that makes the physical connection between the Host Bus Adapter on the computer system and the storage device. After arriving at the storage device, the I/O requests may be further processed and split into several more I/O requests to the actual storage "units" such as disk drives. Each Storage Unit processes its request and data is eventually transferred between the storage unit and the application memory space. The following sections present a more detailed description of each level in the hierarchy with respect to its function and performance implications.

2.1 Computer System

The Computer System is a critical piece of the Storage Subsystem Hierarchy in that it encapsulates all the software components and the necessary interface hardware to communicate with the Physical Connection layer (i.e. the Host Bus Adapter). The components within the Computer System include the processors, memory, and internal busses that connect the memory to the processors and to the Host Bus Adapters. The performance characteristics of each of these major components (i.e. the clock-speed, number of processors, processor architecture, memory bus bandwidth, ...etc.) plays a significant role in the overall performance of the Storage Subsystem as will be demonstrated in a later section. However, given the fastest hardware available, the Storage Subsystem will only perform as well as the underlying software, starting with the Application Program

2.2 Application Program

The term "Application Program" as it is used here is any program running on the Host Computer System that requires data movement between the memory in the host computer and a Storage Unit. Application programs can be either typical User programs or can be parts of the Operating System on the host computer such as the paging subsystem. In any case, these programs have the ability to make I/O requests to any of the lower-level layers in the hierarchy if the Operating System provides an appropriate programming interface to do so. For example, the benchmark program used to gather statistical data for this study can access a storage unit through the file system manager, the logical volume device driver(s), or through the disk device drivers. In general, Applications that can manage and access the lower levels of the hierarchy achieve better performance than Applications that must traverse through the higher level layers such as the File System Manager.

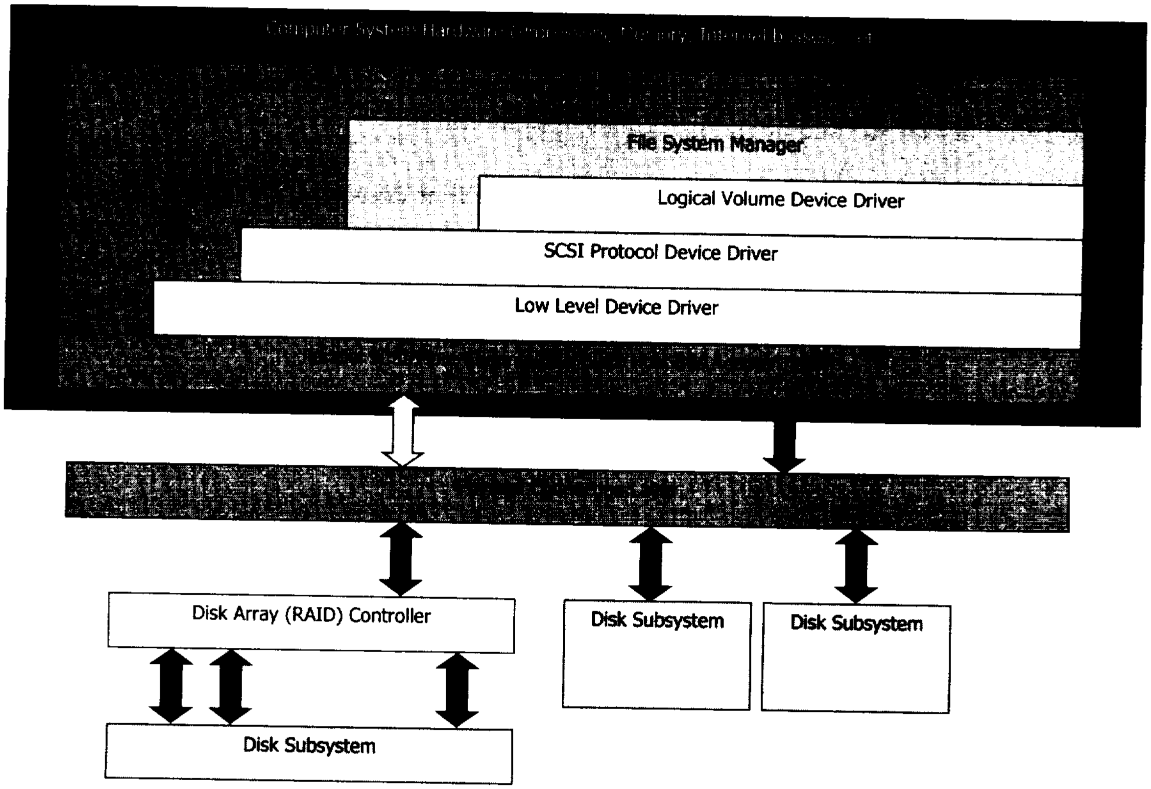

Computer System Hardware (Processors, Memory, Internal busses, etc)
File System Manager
Logical Volume Device Driver
SCSI Protocol Device Driver
Low Level Device Driver
Disk Array (RAID) Controller
Disk Subsystem
Disk Subsystem
Disk Subsystem

Figure 1. The Storage Subsystem Hierarchy

2.3 File System Manager

The File System Manager is mentioned here for completeness but it is not used in any of the testing performed for this paper with one exception. The File System Manager provides a level of abstraction for the Application Program in order to simplify the process of accessing data for the application programmer. Because of the amount of "indirect" I/O processing that can accompany a single Application I/O request (such as space allocation, inode lookups, ...etc.), I/O performance testing "through" the File System Manager can become enormously complex and produce misleading results. Therefore, it is beyond the scope of this paper to include any testing through the File System Manager or to report the performance idiosyncrasies of the File System Manager itself. The I/O benchmark program used in this study always bypasses the File System Manager for data movement operations. The one exception to this occurs in the testing that was performed for multi-host access to a set of shared disks in a Windows NT environment. For these tests, a Shared File System was necessary in order to gain "shared" access to the disks from all the hosts involved in the tests. Unfortunately, this functionality is not yet easily available in the Device Drivers available under NT.

2.4 Logical Volume Device Drivers

The Logical Volume Device Drivers provide a mechanism to easily group storage devices into a single "logical" device in order to increase storage capacity, performance, and/or to simplify the manageability of large numbers of storage devices. The Logical Device Driver presents a single device *object* to the Application. The Logical Device Driver is then responsible for taking a single I/O request from the Application (or the File System Manager) and mapping this request onto the lower level storage devices, which may be either actual storage devices or other logical volumes.

There are many ways to configure a logical volume that consists of multiple underlying storage devices. One common configuration is to stripe *across* (also known as striping *wide*) all the storage devices in an effort to increase available bandwidth or throughput (operations per second). In a wide-striped logical volume, data is laid out on the disk in "stripe units". A stripe unit is the amount of sequential data that is transferred to/from a single storage device within the logical volume before moving to the next storage device in the volume. The stripe unit can be any number of bytes from a single 512-byte sector to several megabytes but is generally a constant within a logical volume (Figure 2).

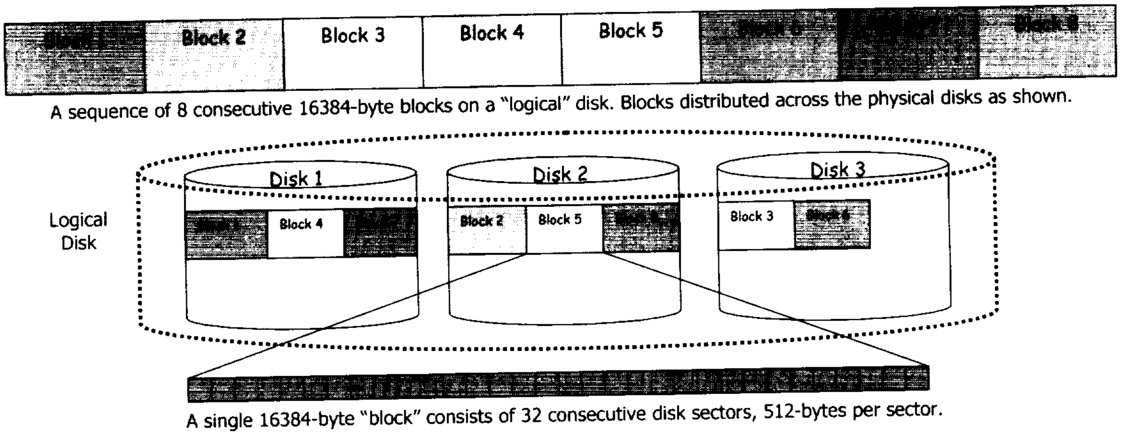

Block 2
Block 3
Block 4
Block 5
A sequence of 8 consecutive 16384-byte blocks on a "logical" disk. Blocks distributed across the physical disks as shown.
Logical Disk
Disk 1
Disk 2
Disk 3
Block 4
Block 2
Block 5
Block 3
A single 16384-byte "block" consists of 32 consecutive disk sectors, 512-bytes per sector.

Figure 2. The layout of a Logical Volume.

2.5 I/O Protocol Device Driver

The I/O Protocol Device Driver is responsible for translating the I/O request from the upper level device drivers into a form that fits the I/O protocol used to communicate the request to the underlying storage devices. In general, an internal I/O request consisting of a command (read or write), a data buffer address, and a data transfer length is converted into a SCSI command and will convey this information to the host bus adapter via the low-level device driver and the disk devices.

2.6 Low-Level Device Driver

This device driver takes the high-level information (i.e. the SCSI command) from the I/O Protocol Device Driver and interfaces directly with the host-bus adapter that will perform the actual data transfer between the storage device and application memory. For example, given a PCI-to-Fibre Channel Host Bus Adapter, this device driver will set up the host bus adapter with the address of the SCSI command buffer, the application data buffer, and the target device and then tell the host bus adapter to begin the operation. The host bus adapter will transfer the SCSI command buffer to the intended target device. The target device at some later point will request a data transfer operation that will be managed in part by the host bus adapter. At the end of the entire operation, an interrupt is generated to notify the Low-Level Device Driver of the completion status the I/O operation. Under normal circumstances, the Low-Level Device Driver then propagates the completion status to the upper-level drivers, eventually reaching the User Application Program.

2.7 Physical Connection Layer

This layer defines the hardware that physically attaches the host-bus adapter to the storage device. These connections can be as simple as a single 3-foot cable or as elaborate as a multi-stage communication fabric spanning many miles. In general, there are two types of physical connections: Parallel-busses or Channels and Serial Interfaces. Parallel busses include SCSI and IDE, Channels include IBM 370-type Block-Multiplexer Channels. Bus-type interfaces are most commonly used inside personal computers and other systems for system disks and other peripheral devices that do not require a great deal of performance (i.e. greater than 100 MB/sec). SCSI busses are also used for larger storage configurations that extend outside the physical boundaries of a computer cabinet. However, due to the nature of the Parallel SCSI bus architecture, the length of SCSI busses is severely restricted when compared to that of Serial Interfaces.

The most common Serial Interfaces include Fibre Channel, USB, and FireWire, to name just a few. For disk storage, Fibre Channel is currently the most prominent Serial Interface. Serial Interfaces have distinct advantages over the more traditional Parallel Bus architectures in the number of different connection *topologies* that are possible. These topologies include Point-to-Point, Loop, and Switched Network (Figure 3). Furthermore, the distance limitations of Serial Interfaces tend to be significantly longer than Parallel Busses making it easier to implement in physically large or extended configurations.

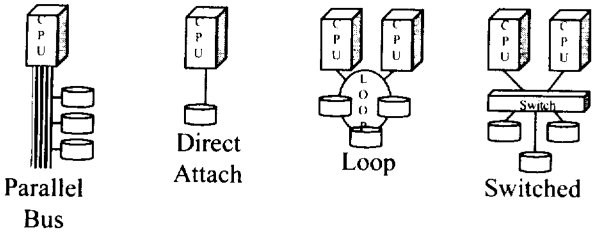


Figure 3. Storage Area Network Topologies.

Point-to-Point connections dedicate a single host connection to a single storage device. This is not the most efficient use of a host connection but does guarantee access to the device via that connection.

The Loop topology, also known as an Arbitrated Loop in Fibre Channel terms, behaves more like a traditional Parallel Bus. However, a Fibre Channel Arbitrated Loop, for example, can more easily accommodate multiple host computers as well as a larger number of storage devices. There are practical limitations on the number of devices and the overall length of a Loop but these can be overcome using a Switched Network topology.

The Switched Network topology allows for any number of options in physically connecting the storage devices to the host computers. It is the most flexible in terms of allowing for multiple access paths to a single storage device, multiple host shared access, fault tolerance, and performance. However, this flexibility also means increased complexity in managing all the nodes connected to the SAN, whether they are host computers or disk devices. These multi-host, multi-device configurations are commonly referred to as Storage Area Networks or SANs.

2.8 Storage Device and Storage Units

The last two layers in the hierarchy are the Storage Devices and Storage Units. The distinction is that a Storage Device is made up of one or more Storage Units but can "appear" to be a single device. The example is that of a Disk Array which is a Storage *Device* that contains several individual Storage *Units* (disk drives) but can appear to the system as a single, very large, disk drive. In the case of a disk array, the I/O request is received from the host bus adapter and is divided up into one or more I/O requests to the underlying disk drives. Storage *Units* are individually addressable storage devices that cannot be further subdivided into smaller physical units. The principal example of this is a Disk Drive.

3 Performance Implications and the *Impedance Matching Problem*

Each layer of software and/or hardware between the Application and the Storage Device adds *overhead* and other anomalies that can result in highly irregular performance as viewed by the Application. Overhead is essentially the amount of time it takes for the I/O request to traverse the specific layer. The source of overhead in each layer is specific to a layer and is not necessarily constant within a layer.

An example of this is the overhead induced by the Physical Connection layer. A physical connection consisting of a short cable introduces virtually no overhead since the propagation time of a signal at the speed of light over a 3-foot distance is not significant. On the other hand, propagation of a similar signal traversing a 20-mile storage area network through multiple switching units will introduce noticeable overhead [2].

An interesting artifact resulting from the *interaction* of the components in the Storage Subsystem Hierarchy is analogous to the Impedance Matching problem in electrical signal on wires. The term "Impedance Matching" is used as an analogy to what happens when there is a mismatch of operational characteristics between two interacting objects. In an electrical circuit, an impedance mismatch has an effect on the "performance" of the circuit in terms of its gain or amplitude at particular frequencies. In the Storage Subsystem Hierarchy, an "impedance mismatch" has more to do with things like I/O request size and alignment mismatches that have an effect on the performance (bandwidth or transaction rate) of the storage subsystem as viewed by the application. The *effects* of these mismatches can be viewed from several different perspectives including the Application perspective, the Disk Device perspective, and the System perspective. The effects of this phenomenon are presented in the sections that follow.

However, it is first necessary to describe exactly how these effects are identified and analyzed.

4 Testing Philosophy and Methods

When approaching the problem of evaluating a storage subsystem it is important to know and understand the operational boundaries of the applications using the storage subsystem. Performance tests are often run on equipment and results are generated or provided that have no real connection to the actual "application" that will be using the storage subsystem. The evaluation of a storage subsystem is intended to answer the basic question of how well applications perform on a storage subsystem in a given configuration.

There are many approaches to answering this question. One way is to run the application on a specific configuration of the equipment under evaluation. The configuration can be "tuned" until the "performance" is optimized for a specific application. However, this is not always easy to do nor is it an accurate method of testing performance if the behavior of the application is not well understood under all circumstances. Furthermore, the results obtained by testing a single application or a small set of applications may not extend beyond those applications to other applications or even to the same application as it (the application) scales in size, complexity, …etc.

The evaluation method advocated by this paper is based on the idea of testing the individual components of a Storage Subsystem followed by testing various "configurations" of these components. It is essential to first understand the performance characteristics of the individual hardware and software components of the entire storage subsystem before the combined performance of the overall subsystem can be accurately assessed. Successive layers/components of the Storage Subsystem Hierarchy are then added to the evaluation testing and the effects of each addition are recorded.

Each successive layer of the Storage Subsystem Hierarchy adds functionality and/or performance to the application. However, a side effect of each successive layer is to add overhead to each I/O request as well as a significant amount of complexity to the evaluation process. The increase in complexity results from the fact that each successive layer adds new independent variables to the performance tuning equation. As a result, this complexity grows exponentially with each successive layer. Understanding the effects on the performance of each of these variables as well as how the variables interact is the goal of the evaluation process. With this information, it becomes easier to identify the cause of performance problems and to compensate by adjusting these and other related variables.

For example, the evaluation process would start by characterizing the performance of a single disk drive. Multiple disk drives can then be added to the same I/O Channel in order to test the performance limits of the host adapter. Several host adapters can then be added, each with a sufficient compliment of disk devices so as to saturate the system bus that connects the host adapters to the memory subsystem of the computer or to saturate the memory bus itself. In either case, the performance of the computer system internal data bus architecture is characterized. The disk drives can also be integrated with a disk array (RAID) controller and the performance of the RAID controller can be characterized as well.

5 The I/O Spectrum and Performance Metrics

There are several metrics used to gauge the performance of a disk subsystem. The *I/O Spectrum* is a concept that divides the performance metrics into two basic types. At one end of the spectrum is *bandwidth* and the other end is *transactions per second* or *IOPs* (I/O Operations per second). Bandwidth is simply the maximum number of bytes transferred per second. This is generally characterized by relatively few, very large data transfer requests per second. IOPs is a measure of how many relatively small data transfers can be processed by the disk subsystem per second. In general, as the size of the requested data increases, the performance moves from *IOPs* to *Bandwidth* along the I/O Spectrum (Figure 4.)

Transactions per second
Bandwidth

Figure 4. The I/O Spectrum.

Related to the IOPs metric are two other metrics worth mentioning: *Response Time* and *Jitter*. Response Time is simply the time it takes to get a piece of data once the request has been issued. It is important to note that Response Time is not simply 1/(IOPs). For example, if a Storage Device has an IOP rating of 2,000 I/O operations per second, this means that the storage controller can accept 2,000 I/O requests every second and that it can simultaneously deliver 2,000 responses per second. However, once an I/O request is received by the Storage Device the associated response may be the next response out, or it may be the 100th response out, or it may be the 6,000th response out. The associated Response Times for each of these possibilities is 1/2000th of a second, 100/2000→1/20th of a second, or 6000/2000→3 seconds.

Jitter is closely related to the Response Time metric. It is a measure of how much the Response Time *changes* over time. For example, given 1000 I/O requests that each have a required response time of 1/30th of a second, jitter measures how many of the 1000 requests failed to meet the response time criteria. Jitter is important in real-time applications that require Response Times to be *consistent*. Such an application is streaming video where the individual video frames must appear before or at the correct time, every 1/30th of a second for example, or the frame is dropped from being displayed.

6 The I/O Benchmark Program

As previously mentioned there are many aspects of performance that are of interest and there are many ways to gather performance data display the information in an easy to understand format. Simply stating that a disk drive can deliver 24MB/sec or 1500 transactions per second does not convey nearly enough information. Rather, the performance of a disk drive as a function of some other variable such as request size or media position is more informative. Furthermore, being able to capture and display this information in a time-correlated manner is useful in understanding the interaction of multiple components within a Storage Subssytem. This is especially important in a shared-access environment where a single computer system does not have the ability to control access to a storage subsystem.

The I/O benchmark program used to gather this information must have several qualities:

- Highly configurable
- Generate "reproducible" results
- Generate "reproducible" usage scenarios
- Very fine degree of tunability

There are many I/O benchmarking programs readily available such as BONNIE, IOZONE, DiskPerf, IOMeter, ...etc. These programs all address different aspects of I/O performance and were not sufficiently focused on the fine details of I/O behavior to be used for this study. Therefore, the benchmark program used to generate the results in this paper has been specifically developed over the past several years at the University of Minnesota and contains features necessary to satisfy the criteria mentioned above. This program is called *xdd* and is available from the web site listed in the title of this paper. Xdd is used to measure many of the disk device performance characteristics as well as helping to identify many of the performance anomalies that appear in more complex configurations.

7 Testing Framework

Testing in a multiple-host environment required the creation of a framework to coordinate testing on multiple systems concurrently[4]. The two basic functions of this framework are:

- Accounting for the existence of multiple clocks
- Coordinating the initiation of tests to run concurrently on multiple hosts

Xdd makes use of precise time stamps to quantify and report storage performance characteristics. Each host accessing the shared storage has its own internal sense of time, and without a common reference clock it is impossible to interpret the relationship between tests run on separate hosts. Thus a consistent time base is needed in order to correlate test results generated by separate systems.

7.1 The Reference Clock

Each of the systems used for testing has a clock register that updates at a high frequency, allowing for very precise measurement of elapsed time. The resolution of this clock varies for different systems (ranging from 2 to 80 nanoseconds per tick or so), so clock values are converted to a common time unit (picoseconds) for the purpose of synchronization.

A very simple clock model to establish a common time base. It is assumed that all clocks run at the same, constant rate. Therefore it can be assumed that conversion from a given machine's "local time" to the common "global time" involves only the addition of a constant to the local clock's value. With this simplified model it is only necessary to determine the value of the constant difference between pairs of clocks.

One machine is designated to keep the global sense of time. That machine provides a service with which others communicate to determine the offsets of their own clock from the global time. Each client initiates a request to the server to get the current global time. The difference between the time value returned and the client's local time is recorded as the basis for the offset. This offset is further adjusted to compensate for the propagation delay required to carry the time request and its response over the communication medium. This propagation delay bounds the error in the difference between the estimated and the actual offset between the two clocks. This request/response protocol is repeated a number of times, and use the offset associated the minimum propagation delay as the final offset value.

7.2 Coordination of Concurrent Tests

With a common time base defined, it is possible to coordinate the initiation of tests on different host systems. Xdd is able to determine the time offset for the machine under test. The program is provided an indication of a (global) time at which all tests are to begin. This global time is converted to a localized start time using the offset value. Xdd then polls the high-resolution clock repeatedly until the start time has arrived. At that point test execution begins. Test results generated by individual hosts are buffered during test execution, and saved to disk for later analysis.

8 Disk Device Basics

In order to understand some of these performance anomalies, a short course in disk devices is necessary. It is assumed that the reader has a basic understanding of how Disk Drives are put together in the sense that they contain platters, heads, cylinders, sectors, and lots of 1's and 0's. However, it is worth describing some of the more subtle design concepts of a disk drive that have an impact on the performance. These concepts include Zoned Bit Recording, Caching, Rotational Latency, Seek Time, the On-board Disk Processor Overhead, Command Queues, and Disk Arrays.

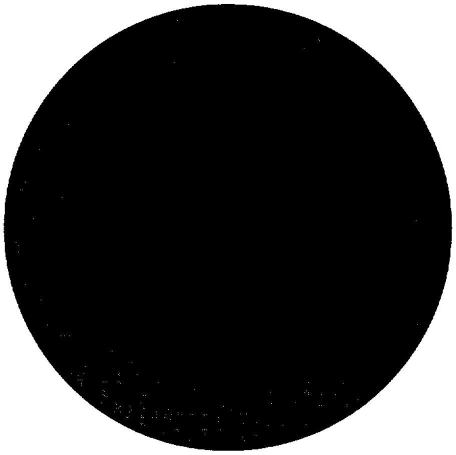
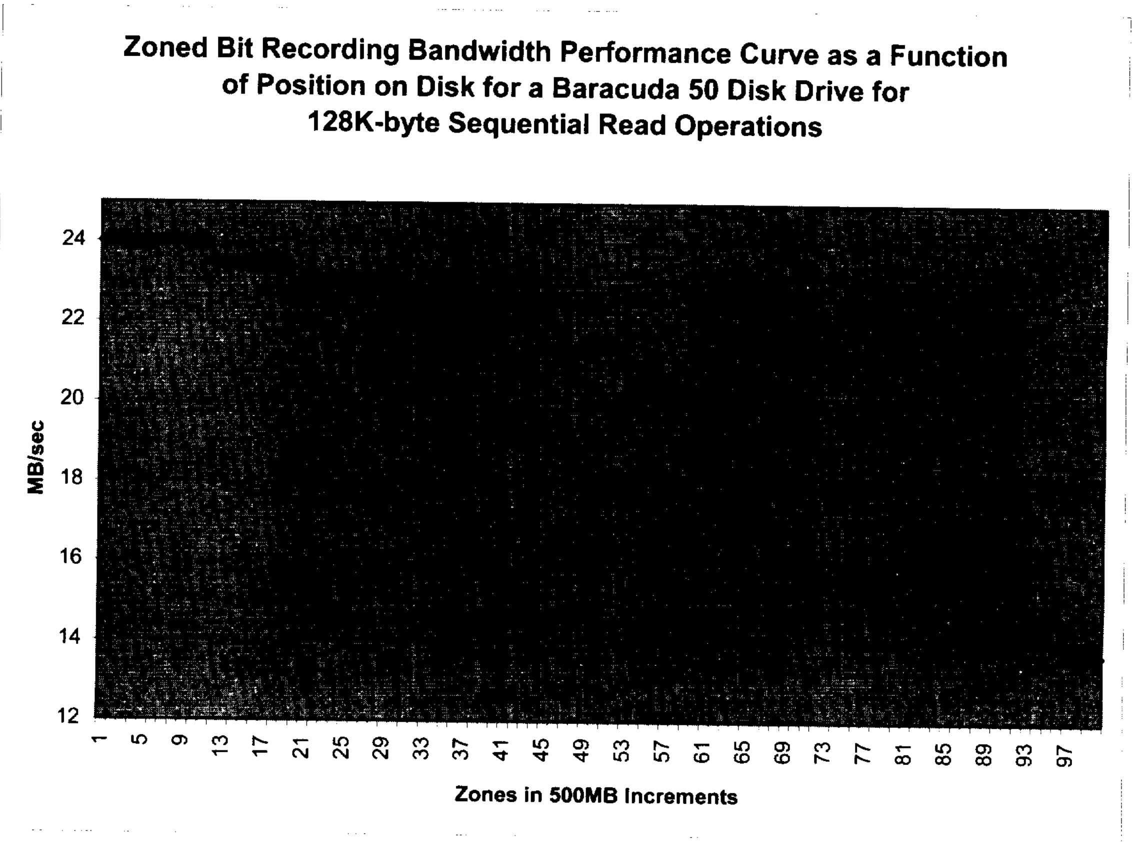


Figure 5. Zoned Bit Recording. Note how the outer band has more sectors than the inner bands.

8.1 Zoned Bit Recording

The data transfer is the rate at which actual *user data* can be read from or written to the media. This transfer rate can vary in such a way that depends on the *physical location* on the disk media where the transfer is to take place. This is due to a recording technique called *Zoned Bit Recording (ZBR)* whereby more data is written on the outer tracks of a disk platter than on the inner tracks. This allows for more efficient utilization of the recording area and hence greater overall capacity. ZBR is used on most current generation disk drives. Given that a disk drive spins at a constant rate, 7200 RPM for example, the outer zones that contain more data will transfer data at a higher data rate than the inner zones that contain less data.

This is clearly demonstrated in Graph1 where the Effective Data Transfer Rate is plotted against the physical position on the platter. Each increment along the X-axis is a 500MB segment of the disk. As data is read from segments successively further into the disk, the data rate at which the data is transferred decreases. However, the decreases are not gradual but are distinct "steps" along the performance curve. Each of the horizontal plateaus is a physical zone on the disk. This graph shows that there are 14 distinct zones on this disk which matches the manufacturer's specification. It is interesting to note that the width of outer zones is larger than the width of the inner zones.

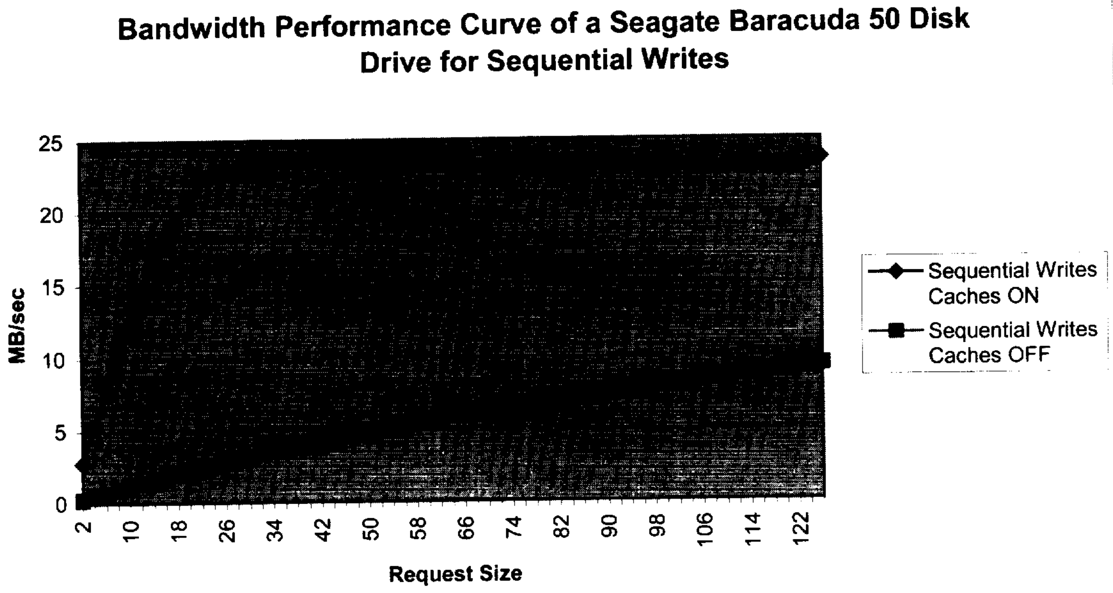


Graph 1.

8.2 Caching

When data is read off the disk media it is stored temporarily in a buffer *cache* before it is sent to the host bus adapter (controller). The data transfer rate from the cache onto the bus is normally done at the speed of the bus that is usually much faster than the transfer rate off the media. The cache can also be used during write operations to accept incoming data and "complete" the write operation before the data is actually written on the media. This can speed up the process of writing small amounts of data to a disk device by a factor of 10-100 since the requesting application does not need to incur the additional overhead of the seek operation and rotational delay associated with writing the data to the disk media.

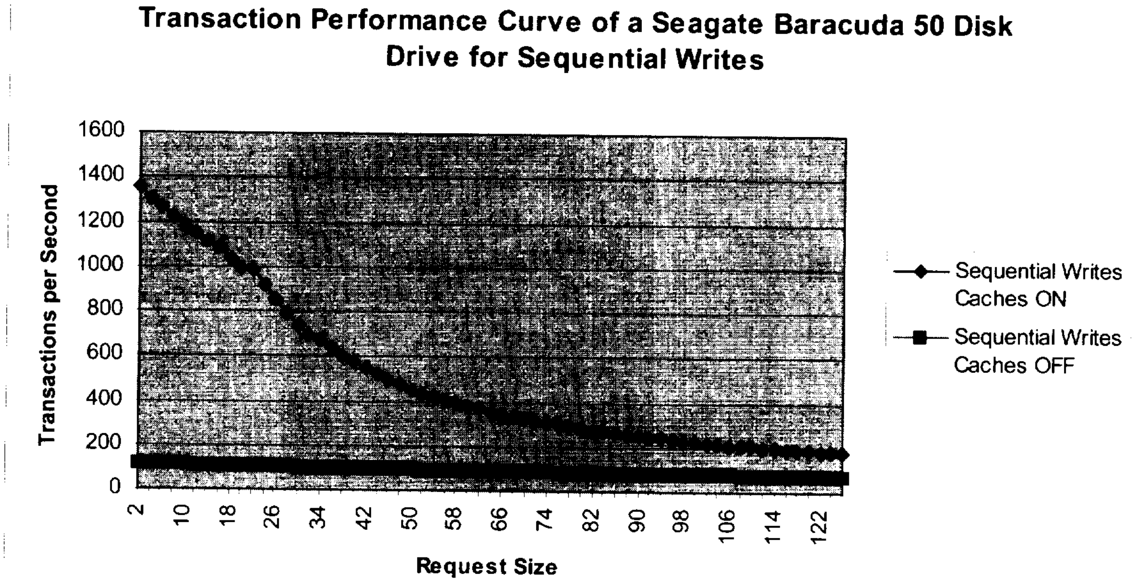
The size of the buffer cache also has an impact on the transfer rate performance of a disk drive. Consider a disk with no buffer cache. The data would then proceed directly from the bus to the media and would be limited to the data transfer rate to the media. If the buffer cache size was increased to one megabyte for example, then data transfers could proceed between the cache and the bus at bus speeds while simultaneously transferring data between the media and the cache at media speeds. Buffer caches can be very helpful when streaming data sequentially off the disk media. After data from a single read request is transferred into the buffer cache, the disk can perform a *read-ahead* operation and continue to transfer subsequent data into the buffer cache in anticipation of the next read request. Without a buffer cache and the read-ahead operation, the subsequent request would arrive and the disk would have to wait for an entire rotation of the disk before the data transfer could begin again.



Graph 2.

Graphs 2 and 3 demonstrate the effectiveness of a Write Cache for purely sequential write operations. The graph plots Bandwidth and Transaction performance as a function of request size. It is clear that the write cache significantly improves the performance of the disk for any size write operation. For small operations, in the 1024-byte per transaction range, the transaction rate is approximately 14 times higher when using the cache for write operations than having the cache disabled.

Graphs 4 and 5 further demonstrate the effects of caches on purely random transactions: reads and writes. These graphs show that random read operations do not benefit from the cache and closely track the performance of non-cached random write operations. However, the cache is still effective in improving the performance of small random write operations up to about 64Kbytes where the performance curve tracks the non-cached performance of both reads and writes.



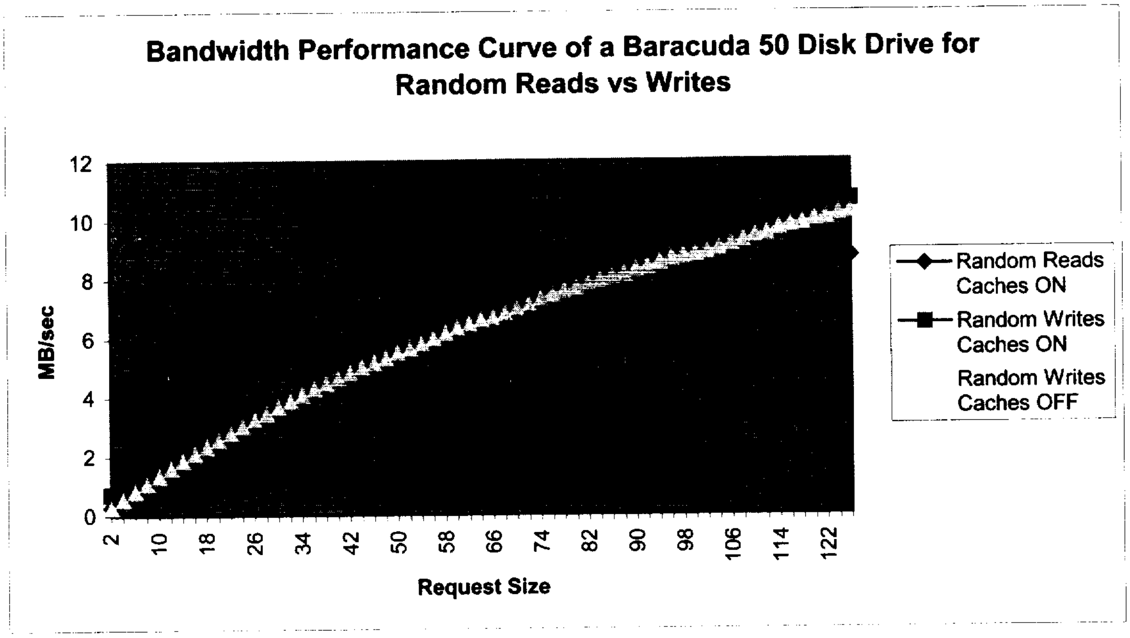
Graph 3.

8.3 Rotational Latency and Seek Time

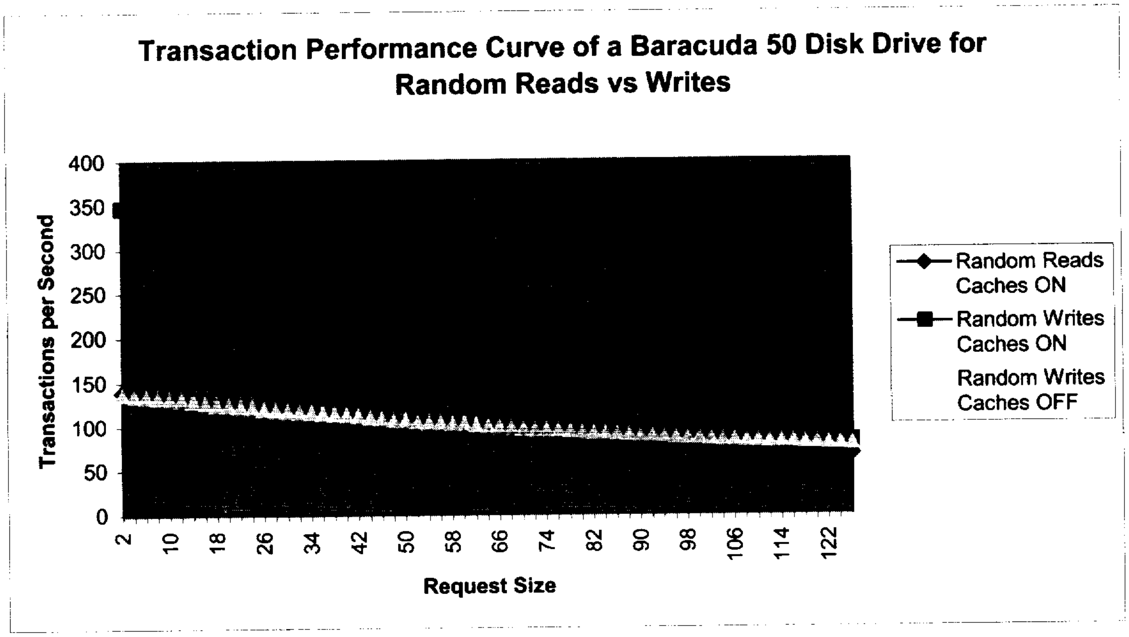
The Rotational Latency is the time that it takes for the disk platter to rotate such that the requested sector is directly under the read/write head. The rotational latency is simply 1 divided by the rotational speed of the disk. Rotational rates for the most common disk drives are 5400, 7200, and 10,000 revolutions per minute. This translates to rotation times of 11.1ms, 8.3ms, and 6ms respectively.

The seek time is the time it takes to position the read/write head over the correct cylinder on the platter. This time can vary by a factor of 10-20 from a single track-to-track seek to a full drive seek (from cylinder 0 to the last cylinder on the disk). Typically seek times range from slightly less than 1 millisecond to about 20 milliseconds for a full seek. Seek operations for write operations take longer than those for read operations because write operations need to seek to the required cylinder and be in perfect alignment before starting the write operation. Read operations however, can start reading before the head completely settles.

Graphs 4 and 5 show the effects of Rotational Latency and Seek Time on read and write performance when the I/O operations are randomly distributed over the disk. Graph 4 plots Bandwidth as a function of Request Size and also shows the effectiveness of the Write Cache on Random Write operations. Graph 5 plots IOPs as a function of request size for the same access pattern. It is clear that for this particular model of disk drive, the write cache does not have any impact on performance for request sizes beyond 64Kbytes.



Graph 4.



Graph 5.

8.4 On-Board Disk Processor Overhead

The on-board disk processor overhead is the amount of time it takes the disk drive to set up a data transfer not including the seek time and data transfer time. This becomes critical for small data transfers. As the data transfer size becomes smaller, the ratio of the actual time to transfer the data to the time to set up the transfer command gets smaller. On disk drives this is only a problem on for request sizes less than 8192 bytes. On disk arrays however, the processor overhead is significant for data transfers as high as 512Kbytes.

8.5 Command Queues

Most all SCSI disk devices have on-board Command Queues that allow the disk device to queue I/O requests locally to reduce the dead-time between requests. The disk device controllers may also have the option to re-order requests in the queue. An example of this is *seek re-ordering*. As requests come into the disk device, the controller may choose to execute those requests that have data physically located near on another and postpone the execution of a request that requires a longer seek operation. This has two side effects. First, the number of transactions per second is maximized by this strategy. Secondly the order of the requests coming *in* is not necessarily the order in which the requests come *out* of the disk device (i.e. it is not a FIFO). Thus, the *response time* of any particular request is not guaranteed. It is possible, however, to disable command queues and/or alter the caching and seek algorithms on many disk devices in order to attain the desired behavior but it is important to note that use of the command queues can result in these performance anomalies.

8.6 Disk Arrays

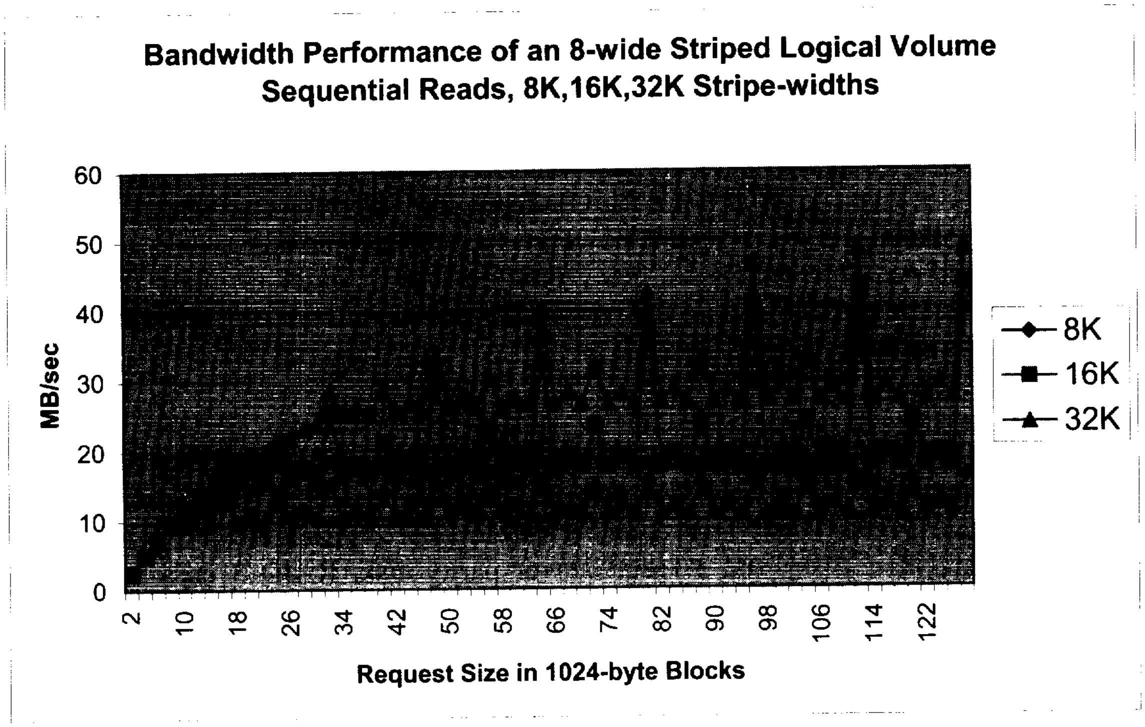
Disk arrays also know as a Redundant Array of Independent Disks (RAIDs) consist of a Disk Array Controller and several disk drives. There are several RAID levels of which two are of interest here: RAID level 3 and RAID level 5. In each of these RAID levels there are several data disks and a redundant or parity disk. RAID 3 uses a dedicated parity disk whereas RAID 5 distributes the parity data among all the disks in the array.

Another distinguishing factor between RAID 3 and 5 is that for each request that comes into a RAID 3, every disk in the array must accessed for each of these requests. This simplifies the internal architecture the RAID 3 and allows for maximum bandwidth. In a RAID 5 disk array however, the disk drives can be accessed individually which maximizes the IOPs performance but significantly complicates the internal architecture and configuration options.

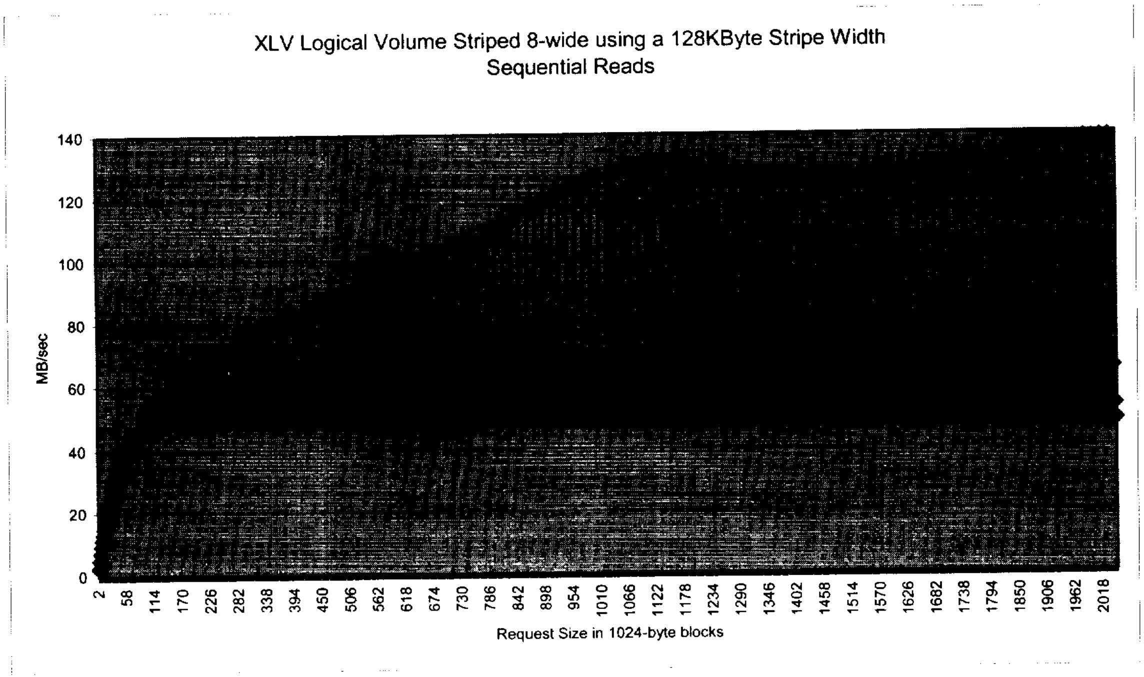
Other important factors in the bandwidth performance of a disk *array* are the internal striping factor and the mode in which it is running and. The internal striping factor is the number of bytes accessed on an individual disk within the array before proceeding to the next disk in the stripe group. Typically, on RAID 3 disk arrays this is 1 byte and is generally not configurable. On RAID 5 disk arrays the striping factor can range from 512 bytes to 64 Kbytes or more. Small striping factors in RAID 5 disk arrays lead to good Transaction performance but relatively poor Bandwidth performance. Conversely, large striping factors in RAID 5 disk arrays lead to poor Transaction performance but good Bandwidth performance.

8.7 Logical Volumes

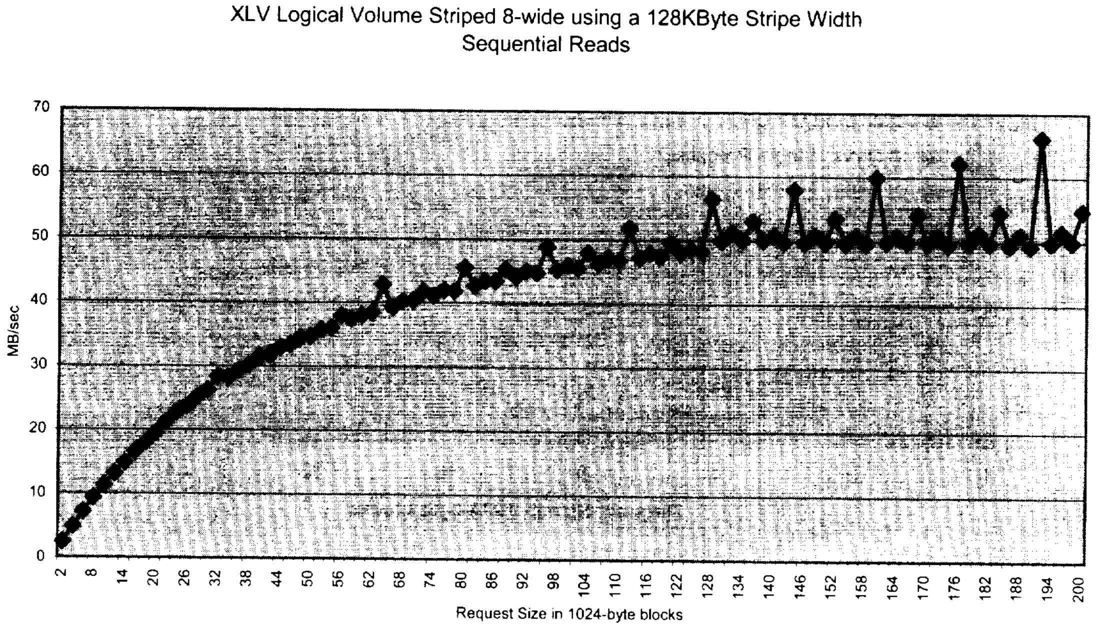
Even though Logical Volumes allow for scalable performance, there are performance anomalies that occur within a Logical Volume that are not entirely obvious. These anomalies manifest themselves as dramatic shifts in performance that are triggered simply by a change in the amount of requested data or from the alignment of the data on the logical volume. The following graphs, 6-10, show a variety of these performance anomalies that are a direct example of the Impedance Mismatch problem. Each of these volumes was created using the Silicon Graphics XLV Logical Volume Manager and measurements were taken from an SGI ONYX2 computer system with eight processors and a Dual Channel Prisa PCI64 Fibre Channel Host Bus Adapter.



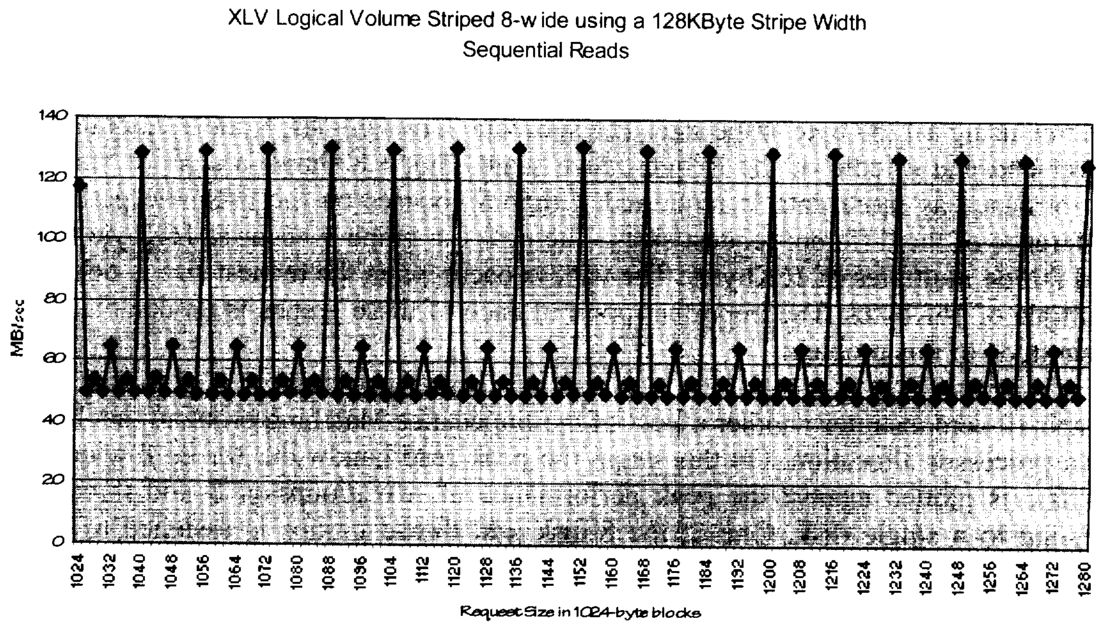
Graph 6.



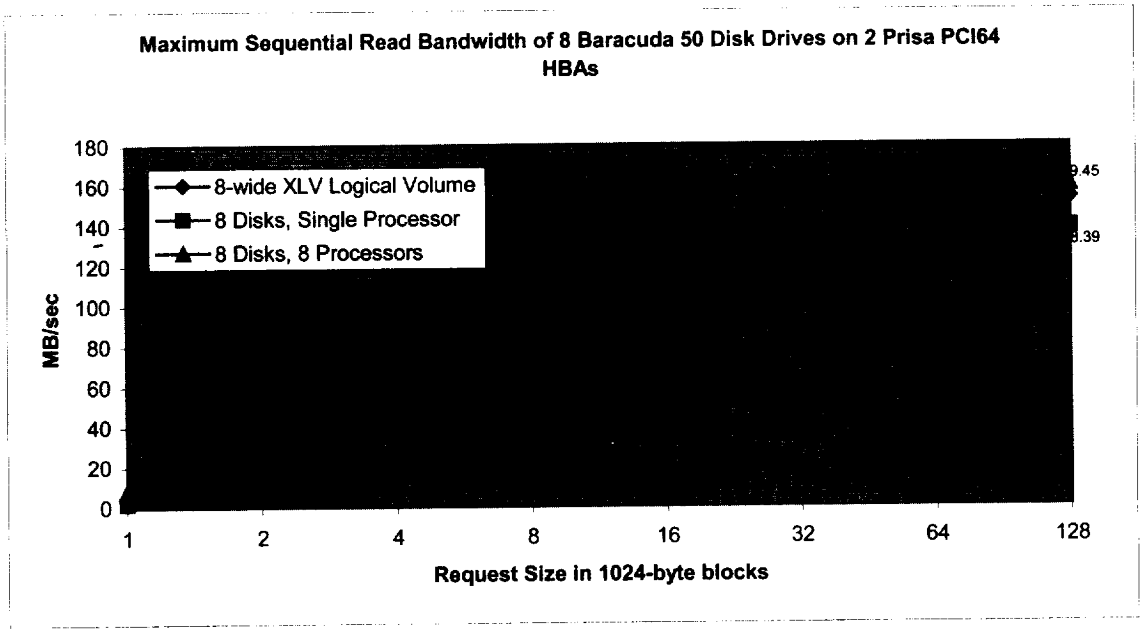
Graph 7.



Graph 8.



Graph 9.



Graph 10.

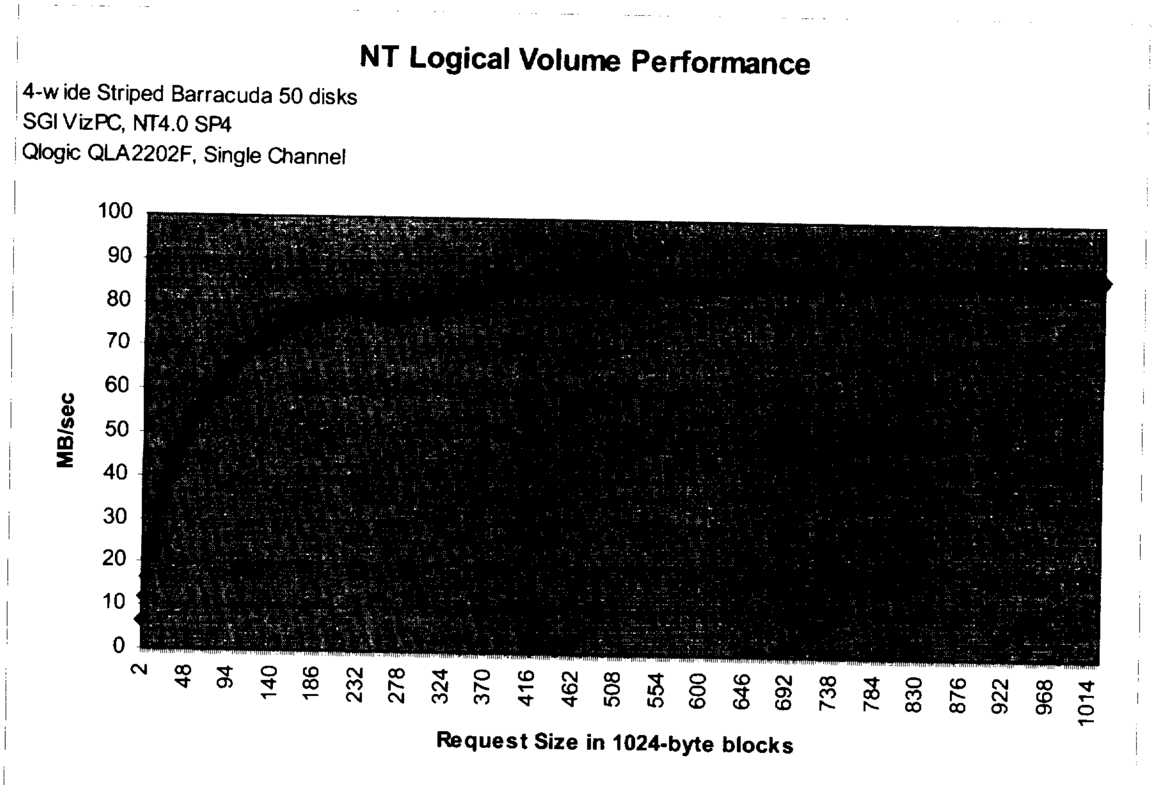
Graph 6 shows the performance of three 8-wide logical volumes with difference striping factors. This graph shows that the overall sequential read bandwidth increases as the stripe unit size increases but so does the *variability* in the bandwidth. For example, the Logical Volume using a 16Kbyte stripe unit can have its performance vary from 18MB/sec up to 44MB/sec simply by choosing a different request size (the number of bytes requested by the application on each read operation).

Graph 7 is a more dramatic view of the same phenomenon but this time on a logical volume using a 128Kbyte stripe size. The subsequent two graphs, 8 and 9, zoom in on the lower end and middle of the Request Size scale. Graph 8 shows a smooth ramp-up in performance as more data is requested. Graph 9 focuses on the dramatic performance difference of the different request sizes. The peaks in graph 9 occur at 16Kbyte intervals and fall on multiples of 16Kbytes. The valleys occur when the request size is not an even multiple of 16Kbytes. The important point of each of these graphs is do demonstrate the magnitude of this problem.

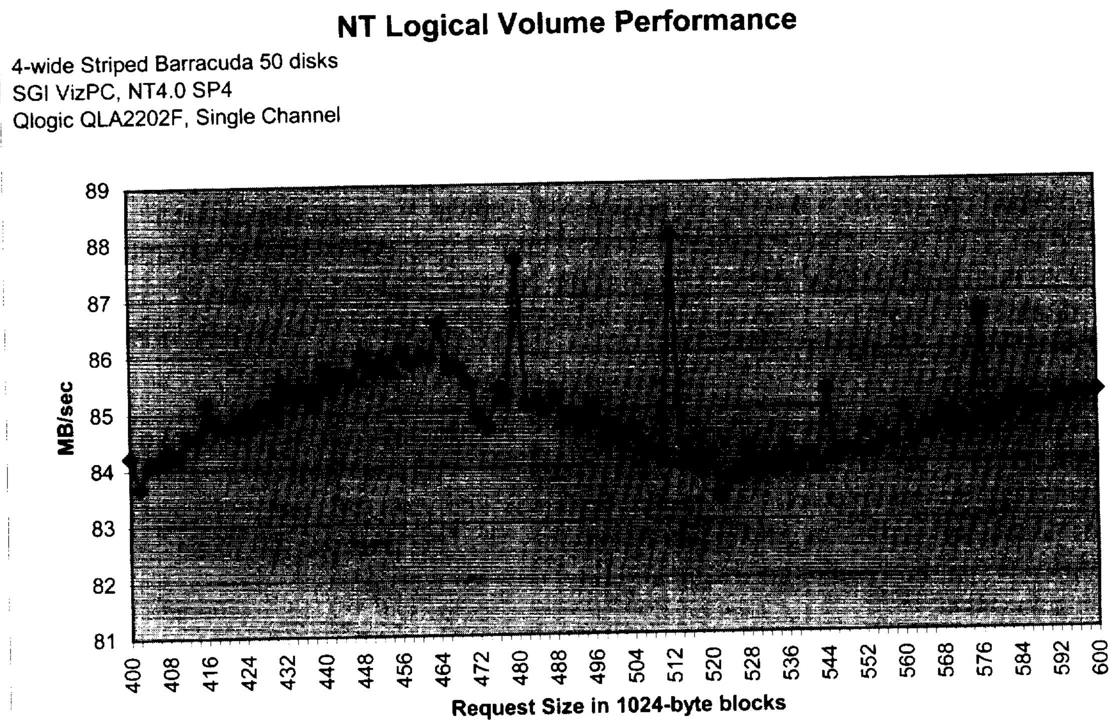
Graph 10 demonstrates another aspect of the Impedance Matching problem that has to do with processor allocation. On this graph the peak read bandwidth for an 8-wide logical volume is plotted against the peak performance of two groups of 8 xdd threads each running to a single disk. One of the 8-disk xdd thread groups is assigned to a single processor in the SGI ONYX2. The other thread group is distributed across all 8 processors in the ONYX2, one thread to each processor. It is clear that the distributed case performs significantly better than the logical volume and the single-processor case. The reason for this has to do with the fact that at lower request sizes more requests are processed per second. It turns out that a single processor gets overwhelmed with processing requests with between 6-8 of these particular disks each running at full speed. When the request processing is distributed across multiple processors, a higher overall performance rate is observed. Also, since the single-processor case closely follows the 8-wide XLV case, it can be concluded that the performance limitations of the XLV logical volume is due to a problem with having all the XLV request processing funneling through a single processor.

Not all Logical Volume software is created equal though. Graphs 11 and 12 show the performance curves for a 4-wide Windows NT Logical Volume striped set of disks. The performance of this logical volume does exhibit some performance variation but not nearly as dramatic as the variations seen in the XLV logical volume. Graph 12 focuses on a small part of Graph 11. This shows the variation to be about 4MB/sec as opposed to the 80MB/sec seen in Graph 9.

The conclusion here is that the Logical Volume performance variations shown in the past several graphs is a function of the Logical Volume software and associated implementation parameters. A more detailed analysis of these Logical Volume performance anomalies is presented detail in [2]. The purpose of these graphs is to demonstrate that things can go wrong and how they go wrong.



Graph 11.

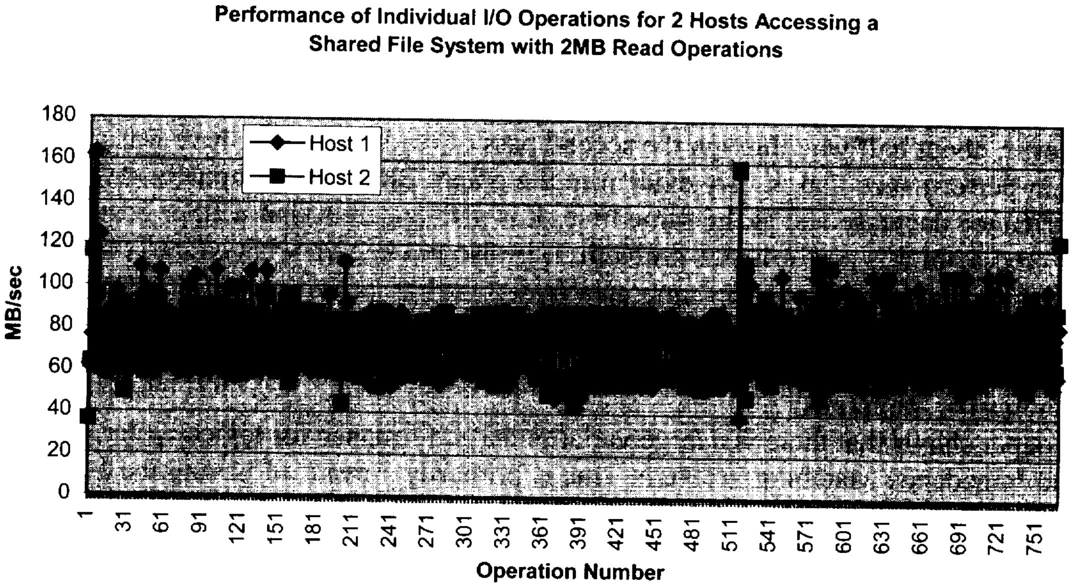


Graph 12.

8.8 Storage Area Network Performance Results

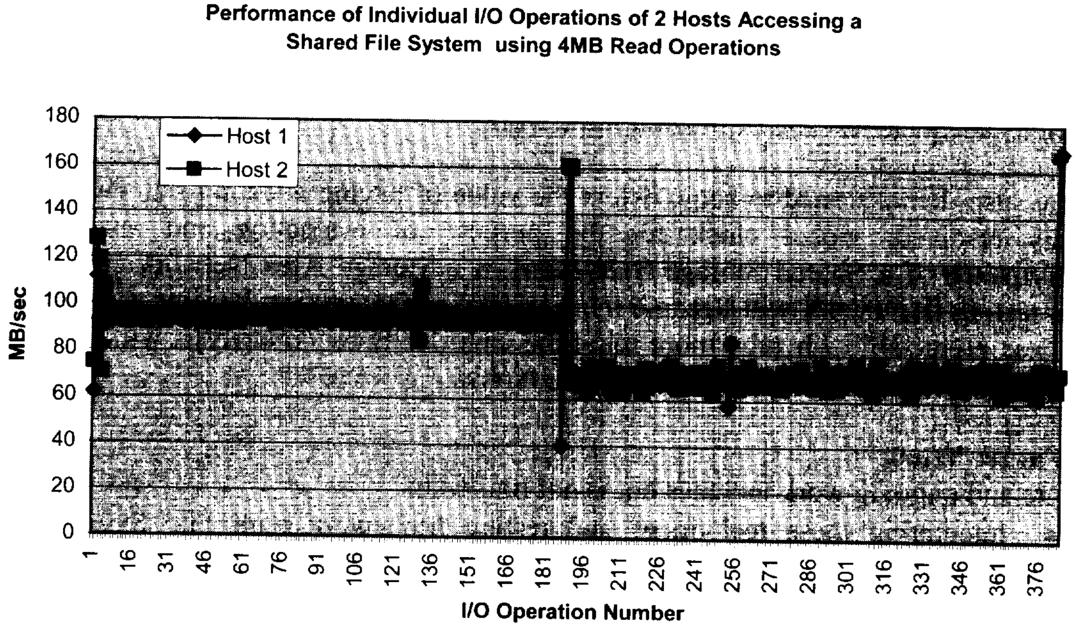
At this point the testing and evaluation process becomes more complex. When testing a storage subsystem on a single, isolated computer system, it is possible to correlate events (I/O requests, interrupts, data transfers, ...etc.) in *time* at a very high resolution. This is possible because all the performance benchmark application runs on a single computer using a single "reference clock" where all events are based on that single reference clock. In a Storage Area Network however, it is necessary to run the benchmark application from multiple computers simultaneously, each accessing the same Storage Subsystem. Each computer system has its own reference clock from which events local to a computer system can be correlated. However, the notion of a "global" reference clock must be established in order to cross correlate events in time over all the systems. In other words, there must be a single reference clock on which to base all the events that occur on the Storage Area Network in order to understand the *interactions* between computer systems accessing a single Storage Subsystem. The generation of this global clock, discussed in section 7, is therefore critical to the evaluation testing process of these SAN configurations.

A simple test was run using two hosts accessing a single set of 16 disk drives configured as a single logical volume through a file system shared between two Windows NT PC computers. Each PC computer had two Fibre Channel connections to the logical volume. The first test consisted of sequentially reading a 1.6GB file using 2MB per request from 2 hosts reading the same file. The file was read three times with results reported for each



Graph 13.

pass. The net result showed each host was able to read the entire file at an aggregate rate of 73MB/sec. Graph 13 shows the instantaneous bandwidth performance of each I/O operation for both hosts. The graph is crowded but it does show that the performance limits of each host remained in a well-defined band from 50-90MB/sec/op.



Graph 14.

Graph 14 however shows the utility of displaying the time-stamped operation data. In this test the same file was being read by the same two hosts but with 4MB read requests. The first half of the I/O operations look very consistent. (There is a small "blip" at the 1/3rd and 2/3rd points on this graph that indicate when the second and third read passes started.) However, about half way through the second pass of reading the file there was an unusual drop in performance that is very evident in the graph. Both host computers saw the same performance decrease and at the same time. It is also apparent that neither host computer recovered from this performance problem. It is also not known what caused this anomaly but further analysis of the timestamp data may reveal an access pattern issue related to a caching idiosyncrasy of the disks.

9 Concluding Remarks

This paper shows that there is a large variation in performance for logical volumes caused by the Impedance Matching problem. This is primarily a result of having the I/O request traversing too many levels in the Storage Subsystem Hierarchy. The I/O request at each level can get resized and/or re-aligned in space and time. By the time the I/O request gets to the storage subsystem, it appears are many smaller requests distributed across many devices. Furthermore, what the application sent over as a "parallel" request can be broken up into a series of smaller, serialized requests to the storage subsystem. The results are demonstrated in a series of graphs that show what happens to the performance as seen by the application when a series of large requests are made to subsystems with different configurations.

These are just some of many examples of the manifestation of the Impedance Matching problem within a Storage Subsystem. Other Impedance Matching-like problems occur in the caches used on the disks arrays and disk drives with respect to their size and caching algorithms, multi-host Storage Area Networks, and the ever-changing bandwidths and latencies of the subsystem interfaces. These are all areas that are ripe for investigation given an adequate test and evaluation framework.

It was also demonstrated to some extent the value of having a testing framework with a highly resolved, global clock for the purpose of evaluating and analyzing the performance of a Shared Storage Subsystem in a Storage Area Network environment. This testing framework will become more critical as the systems become more complex and less predictable whereby more real-time empirically-based analysis will be required to resolved problems in large SAN configurations.

10 Future Work

Future and ongoing work includes but is not limited to:

- Integrating these techniques and testing framework with File System testing efforts
- Developing ways to collect subsequently study "real world" storage system activity data
- Improving and expanding the capabilities of the testing software to other operating environments
- Incorporating other storage devices such as tape drives into this testing framework

References

[1] This work was supported in part by the National Science Foundation, under the NSF Cooperative Agreement No. CI-9619019, and by the Department of Energy through the ASCI Data Visualization Corridor Program under contract #W-7405-ENG-48.

[2] Ruwart, Thomas M., “Performance Characteristics of Large and Long Fibre Channel Arbitrated Loops”, *Proceedings*, 16th IEEE Symposium on Mass Storage Systems / 7th NASA Goddard Conference on Mass Storage Systems and Technologies, March 1999, IEEE Computer Society Press

[3] Thomas M. Ruwart and Matthew T. O'Keefe, "Performance Characteristics of a 100 MB/sec Disk Array", Storage and Interfaces '94, San Jose, CA

[4] Alex Elder et al., "The InTENsity PowerWall: A Case Study for a Shared File System Testing Framework", *Proceedings*, 17th IEEE Symposium on Mass Storage Systems / 8th NASA Goddard Conference on Mass Storage Systems and Technologies, March 2000, IEEE Computer Society Press

I/O and Storage Tuning
An Introduction to I/O and Storage Tuning Tips and Techniques

Randy Kreiser
SGI
12200-G Plum Orchard Road
Silver Spring, MD 20904
Tel: +1-301-572-8926
FAX: +1-301-572-3280
rkreiser@sgi.com

Computational power is getting cheap. Thus, it can be argued that the real cost of computing today lies in reliably storing, and rapidly moving, big data. This article will introduce some principles and methods with which to fine-tune I/O and storage in RAID (Redundant Array of Independent Disks) storage systems.

The existence of an important but unrecognized or under-appreciated RAID capacity/performance relationship should be noted here. It is common to recommend a hardware size for a system predicated on the predicted capacity required by the operation. However, users have performance requirements that must often supersede their capacity requirements. For this reason, in correctly analyzed and sized data storage configurations, one will often find more physical system capacity than is strictly required by the I/O and storage workload. The extra capacity is not unnecessary overhead; it is capacity needed to fulfill both the users' storage requirements and the users' performance requirements.

The statements above are expressed in the RAID hardware as:

- Capacity requirements dictate the number of RAID luns needed
- Performance requirements dictate the number of disks needed per RAID lun

With those important considerations in mind, we move on.

Traditional highly available RAID technology provides redundant disk resources in a number of disk-array configurations that render the storage system more available and improves reliability and performance. Each level of RAID offers a different mix of performance, reliability and cost. Which level of RAID to use is completely dependent on the individual situation. No single RAID level is best for every situation. However, five of the most commonly used configurations are:

Level 0: Striping. The various disks in the array each get portions of a file, which is reconstructed upon retrieval. In this way, RAID 0 is similar to XLV striping in that it stripes the data across all the drives but doesn't offer any parity or redundancy. Thus, in the event of a failed drive, all data across the stripe is lost. Advantage: faster access due to parallel operation of the accesses. Disadvantage: if there is a problem with one of the disks, no data is accessible.

In addition, RAID level 0 striping is done on the RAID hardware. XLV does the striping in software. This distinction is important, as some competitors conduct their RAID parity calculations via software on their CPU's, not in the RAID array.

Level 1: Mirroring. In this simple hardware-mirroring format, all disk contents are mirrored on another disk in a simple primary/secondary relationship. Usually done in conjunction with RAID level 0, this guarantees security and performance.

Advantage: parallel I/O requests can be fulfilled simultaneously.
Disadvantage: data storage costs are doubled.

Level 3: RAID 3 is supported in disk configurations of 4+1 and 8+1 due, primarily, to the architecture of most of the RAID controllers (Clariion and Ciprico) that support RAID 3. Parity is contained on one drive, with the data drive heads accessed in lock-step sequence. This promotes extremely high bandwidth to large, sequentially accessed files such as image, graphical, video and satellite data sets.

The performance-price paid for the extremely high bandwidth produced by the physical configuration (single parity drive and multiple head movements) is that RAID 3 will service 3-4 threads of concurrent I/O's very well, but will chock on 8-9 I/O threads. It's also the case with RAID 3 that random I/O's or smaller, more numerous I/O's will degrade performance.

Level 5: In RAID 5 all the drives operate independently. RAID 5 is good for reads, for small I/O's, for many concurrent I/O's, and for random I/O's. Thus, its characteristics are just the opposite of RAID 3 characteristics.

In RAID 5 the parity blocks are distributed across all disks, together with other data. RAID 5 spreads the parity among all of the drives, but within one single, physical I/O it writes parity to only one drive. The next physical I/O writes the parity to a different drive, thus rotating parity. Simultaneously, the data blocks are being written to the other drives which make up the RAID 5 lun. Thus, it might appear that parity disk bottlenecks would be minimized.

However, any advantage or efficiency gained would be offset by the RAID 5 parity and data distribution calculation on writes. Writes are particularly demanding for RAID 5. To offset this write-performance degradation, memory (cache) can be added to each of the storage processors (SP). The amount of cache is dependent upon the number of disk drives owned by the SP plus the size of the I/O's from the application, the number of concurrent I/O's from the application, and the mix or reads versus writes.

Physical Parity Needs Summary:

RAID 0 requires 9% extra space for parity because it doesn't offer any parity.
RAID 1 requires 100% extra space because it is simply direct hardware mirroring.
RAID 1/0 space requirements are almost identical to RAID 1 because it, too, is basically a mirroring system.
RAID 3 requires 20% extra space in a 4+1 configuration, and 11% extra space in an 8+1 configuration.
RAID 5 can run 3 to 16 drives, so the extra space required is calculated as:
1 / Total # drives in lun. So a 15+1 RAID 5 lun would need 6-1/4% extra space.

RAID levels 1 and 1/0 need 100% more space. RAID level 3 needs 11% or 20% depending on the configuration. RAID 5 needs 6-1/4% to 33% depending on the configuration.

40/30/30 Rule

Fine-tuning the RAID system begins with recognition of the 40/30/30 performance rule.

The 40/30/30 performance rule states that 40% of the performance it's possible to extract from the system is within the hardware set-up; 30% is found in the system software, and another 30% reside in the application software.

This document will concentrate on exploiting the 30% of the fine-tuning opportunity to be found within the application software. After analyzing the application to reveal the characteristics of the application, answers regarding the remaining 70% (40/30) should become more clear as well.

Analyze the Application

Inspection of the application reveals valuable data about the characteristics of the I/O load under which the application is operating. Specifically, the overview will reveal the following broad characteristics:

- Large or small I/O's
- Sequential or random I/O's
- Number of concurrent I/O's
- Percentage mix of Reads and Writes in the I/O
- Direct, buffered, or raw I/O to the filesystem/volume. That is, the method (calls) used within the application code (program) to actually execute the reads/writes/opens, etc.

A more detailed analysis of the application will reveal:

- Transaction I/O size and type (Fixed and large are easier)
- An indicated RAID level to use (5, 3, 1_0, or 1)(An instance of the 40% rule)
- Number of disks to use in each single RAID lun (lun = logical unit number)(4+1 versus 8+1, mirror, etc.)

- Write caching or write buffering (Caching is battery backed-up and protected, write buffering is not protected)
- Cache page size (transaction I/O)
- Percentage mix of reads and writes
- Number of concurrent I/O's
- Whether the I/O is network-based or local
- Whether the data to the filesystem/volume is raw, buffered, or direct. That is, the method (calls) used within the application code (program) to actually execute the reads/writes/opens, etc.

Of these, it has been common to find that the two most important considerations are:

- Sequential versus random I/O
- Raw, direct, or buffered type of I/O to the filesystem/volume

The aggregate weight of the above parameters should define the size of the RAID luns and the write caching parameters to use.

Application Analysis Applied

1. The number of concurrent I/O's bears a direct relationship to the RAID level chosen. Because there can't be one RAID lun for one application and another RAID lun for some other application, the choice must be made between optimizing the environment for fewer, larger I/O's or for more numerous, smaller I/O's.

   It follows from this that transaction size is an important factor in selecting the RAID level to use: small I/O's (<32K) are classically linked to RAID 5, RAID 1, or RAID 1_0 luns. Large, sequential I/O's (>256K) are classic to RAID 3. I/O's between 32K and 256K fall into a gray area and decisions are application-dependent.

2. RAID 3 is good for sequential I/O's, but probably not for more than 3-4 concurrent I/O's. RAID 3 heads are locked together and step out across the drives together to write sequential I/O very well. However, RAID 3 heads don't perform random I/O well and are particularly slow for random writes.

3. Large Sequential I/O is best suited for direct I/O to the filesystem. Near raw performance and the benefits of having a XFS filesystem can result.

4. RAID 5, RAID 1 and RAID 1_0 are generally the best choices for random I/O and for more than 70% of read-based applications.

Observations about RAID

1. Creation of an optiondrive (creating one giant partition encompassing the entire disk drive) is recommended.

2. Disk thrashing will probably be produced by:

   A. Creating an external xfslog on the same RAID lun when "fx'ing" RAID. Doing this will cause the heads to 'Ping-Pong' from the extreme inner portion of the drives (when updating the data in the xfslog) to the extreme outside of the drives (to continue writing or reading). ("fx" is an interactive, menu-driven disk utility that creates partitions sizes, disk drive parameters, and writes the volume label of the device.)

      (The author wrote a script that eases the "fx'ing" part. It allows one to "fx" in one execution; all RAID attached to the system as an optiondrive. The script will also perform the command tag queuing (CTQ) setup at the same time. Once "run_fx" has executed, it documents exactly what was just done and, if executed in 'query mode', provides configuration documentation regarding the partition setup on all luns. Part of the 'rktools' tolls set, the script works on Fibre RAID and JBOD and SCSI-2 RAID and JBOD.

   B. The creation of more than 3 concurrently active partitions on the same lun is not recommended. It doesn't matter how many partitions are created, the criticality lies in how many partitions are accessed simultaneously. Having more than 3 partitions active simultaneously can cause disk thrashing and thus, poor I/O performance.

More Observations about RAID

3. Always enable CTQ when "fx'ing" RAID. While setting the appropriate CTQ depth is very important when using threaded or buffered filesystem I/O, CTQ'ing is not applicable when the I/O is single-threaded.
   CTQ depth = 256 / Total # luns owned by the DPE (Disk Processor Enclosure).

4. It is monumentally important to select the appropriate stripe unit size (whenever possible, select an even stripe width I/O) and lun interleaving when creating XLV striped volumes. For instance, if there are two busses with four luns per bus, interleave on the volume element line. (XLV devices provide access to disk storage as logical volumes. A logical volume is an object that behaves like a disk partition, but its storage can span several physical disk devices. XLV can concatenate disks together to create larger logical volumes, stripe data across disk to create logical volumes with greater throughput, and plex (mirror) disk for reliability). It is also the case that with an XLV striped volume having multiple luns; each lun could have its own XLV thread of I/O. For instance if you have an XLV striped volume made up of 4 luns, then the stripe group = 4. If you create even stripe width I/O's an application I/O will fit evenly across all four luns in one physical I/O. This will create up to 4 threads of I/O one to each device. If you had two application threads doing this, you would have two application I/O threads feeding multiple XLV I/O threads.

5. Calculation for even stripe width I/O. Application I/O size = (# of luns * stripe unit). Thus, if the XLV stripe lun group is 4, and the stripe unit is 2048 blocks, the applications' I/O size = 4MB.

Two Ways to Scale I/O

1. Use even stripe widths from the application program.

2. Use more threads of I/O from the application program. Threads could be from the use of posix threads (a set of IEEE standards designed to provide application portability between Unix variants) in the application program, or they could be from running more application program processes.

Underlying Ur-Truth

Until the saturation point is reached, creating more sustained I/O will produce the best overall I/O results.

Summary

Despite, or because of, the massive proliferation of data and our ability to stockpile it in terabyte quantities, many data-intensive operations have trouble quickly and efficiently accessing the information they need. Such operations need to maximize data retrieval, optimize throughput performance, and enhance the performance of both I/O and storage systems. Fortunately, correctly tuned RAID I/O and storage systems can significantly enhance the availability, reliability and performance of the data storage system without significantly increasing the overall system cost.

Accelerated Aging Studies and the Prediction of the Archival Lifetime of Optical Disk Media

David E. Nikles and John M. Wiest
Center for Materials for Information Technology
The University of Alabama, Tuscaloosa, Alabama 35487-0209,
jwiest@coe.eng.ua.edu
tel 205-348-1727
fax 205-348-2346

Abstract

Data archivists expect information storage media to have a lifetime greater than ten years. Furthermore they desire the ability to predict when the media will fail in order to plan for its replacement. Archival lifetime predictions are based on accelerated aging studies, where the media are subjected to conditions of high temperature and high humidity. The rate of failure is measured and the data extrapolated to obtain rates of failure under ambient conditions. This extrapolation is reasonable provided the degradation process is activated and the Arrhenius relationship holds. However this may not be the case for the complicated materials packages in optical data storage media. A primary concern for the polymeric materials is any phase transition, such a glass transition or a beta relaxation, that may occur at temperatures between ambient and the accelerated aging conditions. It is not clear how one extrapolates through those transitions. These phase transitions can give rise to large changes in the rates of diffusion for water, oxygen and other agents of degradation. Furthermore, for polymers, such as polycarbonate, the mode of failure is often hydrolysis and the degradation products can catalyze further hydrolysis, an autocatalytic degradation. The polymer degradation will change the phase transition temperatures. The degradation products may also plasticize the polymer, causing further changes in diffusion rates. We provide here a simple analysis of accelerated aging techniques and discuss other factors that may be involved.

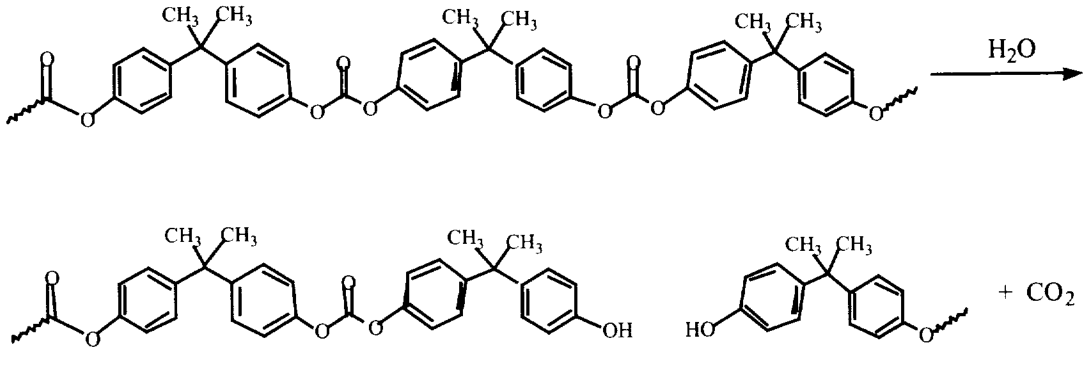
Optical Data Storage Media

DVD is an emerging optical data storage technology that may find application in data archiving. DVD disks are complicated materials packages consisting of a 0.6 mm polycarbonate substrate coated with a recording layer or layers. For double-sided disks, two disks are laminated with a polymeric adhesive layer. In DVD-R the recording layer is a dyed-polymer with a gold reflective layer. The recording layer in DVD-RW is a phase change alloy surrounded by dielectric layers. These material packages are similar to the corresponding compact disk formats, CD-R and CD-RW, except the materials are tuned to red lasers rather than 780 nm lasers. We may draw on the experience for the CD formats to gain a sense for the reliability of the DVD formats. The block error rate was measured as a function of time for CD-R disks exposed to 8% relative humidity and either 60°C, 80°C or 100°C [1]. The rate of degradation of the block error rate under these conditions led to a prediction of a data storage lifetime of 100 years, comparable to the best CD-ROM disks. The materials package for DVD-R is different from CD-R. The substrate is thinner which would make it more sensitive to any mechanical changes arising from polycarbonate hydrolysis. The recording layer contains a different dye, which absorbs the red laser instead of the 780 nm laser. The chemistry of the degradation of the dyes used for DVD-R has not been systematically studied. However accelerated aging studies of naphthalocyanine or cyanine dyes, sensitive to infrared lasers, suggested that the rate of degradation was reaction kinetics limited [2-3]. If the dyes for DVD-R are sensitive to degradation by oxygen or to moisture, it is expected that the degradation would be reaction kinetics limited. However, for more aggressive penetrants, such as ozone, the degradation may very well be mass transport limited. A systematic study of dye degradation by different penetrants would allow a rational accelerated aging strategy to be developed.

In DVD-RW the phase change alloys (In-Ag-Sb-Te) are sensitive to oxidation. The penetrants must diffuse through the polycarbonate and through pinholes in the protective layer ($ZnS-SiO_2$) to get at the alloy. Water would adsorb on the alloy surface, creating an electrolyte for the reduction of oxygen and the oxidation of the alloy [4]. This process disbonds the protective layer, exposing more of the alloy surface to corrosion. The result would be random defects that would increase the bit error rate as the corrosion sites grow. Clearly, this mode of degradation would be mass transport limited.

Polycarbonate

Under exposure to high temperature and high humidity conditions polycarbonate substrates hydrolyze to break the carbonate ester linkage, Eq. (1). The products are carbon dioxide and two new phenol end groups. Ester hydrolysis can be catalyzed by acids or by bases. Although phenol and carbon dioxide are acids, they are very weak acids. It is expected they would have a minimal effect on the rate of hydrolysis. The kinetics for polycarbonate substrate hydrolysis under high temperature and high humidity conditions was reported earlier [5]. Plots of the degree of polymerization as a function of aging time were linear, indicating the degradation process was not autocatalytic. The activation energy for hydrolysis was 70 ± 4 kJ/mol, which was close to the activation energy (59 kJ/mol) for the hydrolysis of diphenyl carbonate [6]. In the course of accelerated aging experiments, the polycarbonate was degrading. There was no determination of the effect of this degradation on the physical properties on the substrate, such as the mechanical properties or the rate of diffusion of penetrant molecules. There is a report that the bisphenol A monomer, liberated by hydrolysis, can diffuse to the surface [7]. The effect of bisphenol A on the degradation of optical disk recording layers is not understood.



(1)

The diffusion of small molecule penetrants into polycarbonate has been studied [10]. At low partial pressure water shows a dual mode soption, consisting of Henry's law and Langmuir contributions. Eq. (2) [8]:

$$A = Kp + \frac{abp}{1 + bp} \tag{2}$$

where p is the partial pressure, K is the Henry's law constant, a is the Langmuir capacity and b is the Langmuir affinity. The Henry's law contribution is temperature dependent as is the solubility of the penetrant in polycarbonate. The Langmuir contribution arises from occupation of excess volume. The excess volume depends on how the substrate was processed. At 25°C the solubility of water in polycarbonate is 0.35%, while the solubility of oxygen is 0.056%. The activation energy for diffusion of water is 26 kJ/mol, while the activation energy for oxygen diffusion is 32 kJ/mol. This information should be collected for other penetrant gases, such as ozone or the gases in the Battelle class II environment. Note that the activation energy for transport is significantly lower than the activation energy for the hydrolytic degradation. In extrapolating from high temperature conditions to ambient conditions, one wonders whether the mode of degradation may change from mass transport limited to reaction kinetics limited conditions.

Aging Studies: Analysis

Accelerated aging studies are used to predict the archival lifetime of information storage media. The strategy is to determine the rate of degradation at elevated temperatures or in environments containing elevated levels of potential degradants. Often the degradation is measured by a system metric, such as increase in bit error rate. The data are then extrapolated to ambient conditions. Typically, this involves a variety of assumptions such as an Arrhenius temperature dependence for the rates. However, these assumptions may be invalid depending on the underlying physical phenomena governing degradation or if the extrapolation passes through a phase transition, such as a glass transition in a polymeric component. Another problem is that many degradation processes, such as polymer hydrolysis or corrosion can be autocatalytic, *i.e.* the product of the degradation process can catalyze further degradation. A predictive model of archival lifetime must be based on an understanding of the chemical and physical processes leading to failure. This model must also account for the effect of these processes on the degradation of the system performance.

To demonstrate possibilities for analysis of accelerated aging strategies, we consider an example in two extreme limits: mass transfer limited degradation and kinetically limited degradation. The geometry that we examine consists of a large (effectively infinite in the lateral directions) plate or disc of the protective material of thickness δ (*i.e.*, the polycarbonate substrate, backed by the data storage material in a recording layer). We consider only degradation of the data storage materials as the results of reaction with a penetrant that must move through the substrate. In addition we make some general assumptions: (1) The system is at steady state. (2) Mass transfer is one-dimensional. (3) The system can be treated as containing only two components: the penetrant and the substrate material and, therefore, adequately modeled with binary Fickian diffusion. (4) The substrate material is stationary (not diffusing). (6) Only a single degradation reaction occurs, and reaction products can by neglected in the analysis. (7) The degradation reaction occurs at the interface between the substrate and the data storage layer. The propriety of these assumptions is discussed below.

First we consider the case where degradation is kinetically limited. This means that there is always sufficient penetrant at the interface between the substrate and the data storage layer for the rate of degradation to be governed by reaction kinetics. Hence, we need only examine the kinetics of the degradation reaction. If we assume that the reaction exhibits Arrhenius temperature dependence, then the rate of degradation, R, shows the same temperature dependence, and we have:

$$R \propto e^{-E_r/T} \tag{3}$$

where E_r is the activation energy for the reaction, and T is the absolute temperature. This simple relation prescribes that accelerated aging tests at temperature T_0 can be used to predict the degradation rate at temperature T via:

$$R(T) = R(T_0)\left\{\exp\left[E_r\left(\frac{1}{T_0} - \frac{1}{T}\right)\right]\right\} \tag{4}$$

If the penetrant concentration at the outer surface of the substrate, c_p, is varied in an accelerated aging test (*e.g.*, by varying the ambient humidity) in the kinetically limited system, then Eq. (4) will be modified simply by including the ratio of concentrations raised to the appropriate reaction order, n, on the right-hand side. That is:

$$R(T,c_p) = R(T_0,c_{p0})\left\{\left(\frac{c_p}{c_{p0}}\right)^n \exp\left[E_r\left(\frac{1}{T_0} - \frac{1}{T}\right)\right]\right\} \tag{5}$$

Hence, knowledge of the reaction order (n) and the activation energy for the degradation reaction (E_r) is all that is required for aging analysis.

If degradation is mass transfer limited, then the rate of degradation is governed by the rate at which penetrant can diffuse to the interface between the substrate and the data storage layer. The degradation reaction can be viewed as occurring spontaneously. The degradation rate is then equal to the mass transfer rate of penetrant. Therefore, under the assumptions given above we have [9]:

$$R = \frac{-D_{pc}}{\delta}\ln\left(1 - \chi_{ps}\right) \tag{6}$$

where D_{pc} is the effective binary diffusivity of the penetrant and χ_{ps} is the ambient mole fraction of the penetrant at the outer surface of the substrate. The diffusivity in Eq. (5) is a relatively strong function of temperature. It is reasonable to assume that [10]:

$$D_{pc} \propto e^{-E_d/T} \tag{7}$$

where E_d is the activation energy for diffusion. Hence, the analogue of Eq. (5) for predicting aging at temperature T and ambient mole fraction χ_{ps} from data at T_0 and χ_{ps0} in this limiting case is:

$$R(T,\chi_{ps}) = R(T_0,\chi_{ps0})\left\{\frac{\ln(1-\chi_{ps})}{\ln(1-\chi_{ps0})}\exp\left[E_d\left(\frac{1}{T_0}-\frac{1}{T}\right)\right]\right\} \tag{8}$$

That is, knowledge of the activation energy for diffusion (E_d) is sufficient for the accelerated aging analysis. Eq. (8) also suggests the possibility of accelerated aging tests, valid under our fairly restrictive assumptions.

The dependencies on ambient penetrant concentration in Eqs. (5) and (8) are quite different indicating the importance of knowing which mode of transport and degradation dominates. The temperature dependencies in the two equations are very similar, although the difference in magnitudes of the two activation energies should make it possible to distinguish whether the process is mass transfer or reaction limited.

In reality, the degradation process probably occurs somewhere between the two extremes of mass transfer limitation and reaction kinetics limitation. In addition, it is important to examine the applicability of our general assumptions. The assumption that the system is at steady state and that transport is one-dimensional are acceptable for the case where the degradation is heterogeneous – occurring only at the interface between the substrate and the recording layer. The assumption that the system can be treated as containing only two components is clearly over-restrictive. Many species can diffuse through the substrate and cause degradation. The diffusion rates for each of these species will be different. In addition, the reaction products generated by the degradation reactions may actually promote further degradation. Including all of the possible degradation reactions and species involved would complicate the analysis considerably. The basic forms of the results (Eq. (5) and (8)) would be similar but with a spectrum of activation energies and reaction orders. Furthermore, it may be necessary to account for some movement of the substrate itself as it swells under the influence of the penetrant.

The assumption that the degradation occurs only at the interface between the substrate and the recording layer is also overly restrictive. Penetrants will diffuse into the recording layer and degradation will occur throughout the material. Furthermore, degradation of the substrate (*e.g.*, hydrolysis) will also occur. Including this homogeneous degradation will necessitate performing a non-steady state analysis of the process. That is, the analysis would require examination of the basic mass transfer equation in the material. Under the assumptions listed above (*i.e.*, Fickian transport, constant diffusivity, binary system, stationary substrate) that is:

$$\frac{\partial c_p}{\partial t} = D_{pc}\nabla^2 c_p + r_p \tag{9}$$

where r_p is the rate of reaction of the penetrant. However, it may be possible to include the diffusivity give by Eq. (7) in such a way as to provide a type of time-temperature superposition for aging studies. For example, if we assume that the rate of reaction of penetrant is first order in the concentration of the penetrant (*i.e.*, $r_p=-kc_p$), then the solution of Eq. (9) may be formally written as:

$$c_p = -k\int_0^t e^{kt'} f(\mathbf{r},t')dt' + e^{kt} f(\mathbf{r},t) \tag{10}$$

where we have assumed that there is no penetrant in the material initially, and $f(\mathbf{r},t)$ is the solution of:

$$\frac{\partial f}{\partial t} = D_{pc}\nabla^2 f \tag{11}$$

with boundary conditions as specified for c_p and a homogeneous initial condition. That is, $f(\mathbf{r},t)$ is the solution of the mass transfer problem in the absence of reaction. Now we imagine conducting an aging experiment at a temperature T_0, with the goal of extracting information on the degradation that would occur at (lower) temperature T. The rate of degradation should be proportional to the concentration of penetrant in the sample, so we need to examine how the solution given by Eq. (10) varies with temperature. Temperature enters parametrically in this solution in two places: through the rate 'constant', k, and through the effective binary diffusivity, D_{pc}. For the former, it is reasonable to assume an Arrhenius dependence:

$$k(T) = k_0\left\{\exp\left[E_r\left(\frac{1}{T_0}-\frac{1}{T}\right)\right]\right\} \tag{12}$$

where k_0 is the rate constant at temperature T_0. For the diffusivity, we define a new time scale, s through:

$$ds = \frac{D_{pc}(T)}{D_{pc}(T_0)}dt \tag{13}$$

This is a simple re-scaling of time if the temperature T is constant, and Eq. (11) can be thought of as defining a material time, if T is varying with time. From Eqs (11) and (13), we find:

$$f(\mathbf{r},t,T) = f(\mathbf{r},s,T_0) \tag{14}$$

That is, in the absence of reaction the mass transfer scales with temperature, so that we can conduct experiments on the accelerated time scale s at temperature T_0 and extract information about degradation on (longer) time scale t at temperature T. The relation between the time scales is given simply by Eq. (13) — requiring only knowledge of the temperature dependence of the effective diffusivity. Furthermore, the transformation remains valid irrespective of any phase changes that may occur in the material between T and T_0; all that is required is that the mass transfer remain Fickian.

In the presence of reaction, we must combine Eqs. (11) through (14) to obtain:

$$\begin{aligned} c_p(r,t,T) = & -k_0\left\{\exp\left[E_r\left(\frac{1}{T_0}-\frac{1}{T}\right)\right]\right\}\int_0^t e^{k_0\left\{\exp\left[E_r\left(\frac{1}{T_0}-\frac{1}{T}\right)\right]\right\}t'} f(\boldsymbol{r},s',T_0)\,dt' \\ & + e^{k_0\left\{\exp\left[E_r\left(\frac{1}{T_0}-\frac{1}{T}\right)\right]\right\}t} f(\boldsymbol{r},s,T_0) \end{aligned} \tag{15}$$

where s is given by Eq. (13). Unfortunately, the simple time re-scaling that we have for the no-reaction case does not apply here. That is, we cannot write the right-hand-side of Eq. (15) as a simple transformation of $c_p(\boldsymbol{r},t,T_0)$.

Finally, although our assumption that the mass transfer behavior is Fickian would at first seem to be innocuous and generally acceptable, mass transfer in polymeric materials is known to be anomalous — displaying a wide variety of nonlinear behaviors that are not described by Fick's law [11]. Incorporating nonlinear behavior into an accelerated aging analysis strategy presents many difficulties and possibilities for research. Given all of the above, it is clear that a more complete (and complicated) analysis is required.

Aging Studies: Experimental Needs

High temperature and high humidity accelerated aging studies are undertaken under the assumption that water and oxygen are the reactants that are causing failure. This may be true in many cases, however there are other trace substances in the atmosphere, particularly air pollutants, that may cause degradation. In response to this, Battelle has specified testing conditions to accelerate failure for the copper in electronics in an office environment — Battelle Class II Environment [12]. These conditions have been used in accelerated aging studies on metal particle magnetic tape [13]. The saturation magnetization was measured as a function of time exposed to Battelle Class II conditions. There was a concern that the iron particles in MP tape would be susceptible to corrosion and this would limit

Table 1. Battelle Class II Conditions [1]

Temperature	30°C
Humidity	70%
NO_2	200 ppb
Cl_2	10 ppb
H_2S	10 ppb

archival lifetime [14]. The corrosion problem has largely been solved by coating the particles with an amorphous aluminum oxide [15-17]. However tapes from different vendors can have vastly different rates of corrosion [18]. Furthermore, there was a concern about the reproducibility of this experiment [19]. Sides and Spratt used an impinging jet geometry that gave more control of the delivery of the corrosive gases to the tape samples [20]. However, no one has systematically determined the role of each component in the gas mixture plays in the degradation. Furthermore, these conditions do not include some of the more aggressive substances, *e.g.* ground level ozone, present in many urban environments. Similar studies should be performed on optical data storage media.

Conclusions

A fundamental understanding of the chemical and physical processes that lead to failure in optical disks must underpin the accelerated aging studies used to predict the archival lifetime. Only with an understanding of the kinetics of degradation and mass transfer can rational models be developed for lifetime prediction.

Acknowledgement

The authors gratefully acknowledge financial support provided by the National Science Foundation through the Materials Research Science and Engineering Center at the University of Alabama (DMR-9809423).

References

1. D. G. Stinson "Durability of Kodak Writable CD Media" Archival Storage Workshop, Center for Magnetic Recording Research, University of California at San Diego, July 2, 1997.

2. D. E. Nikles, K. Chiang, H. A. Goldberg, R. E. Johnson, R. S. Kohn, and F. J. Onorato "Accelerated Aging Studies for Organic Optical Data Storage Media" *Proc. SPIE-Int. Soc. Opt. Eng.* **1078**, 189-198, 1989.

3. D. E. Nikles, K. Chiang, H. A. Goldberg, R. S. Kohn, and F. J. Onorato "Naphthalocyanine Chromophores for WORM-Type Optical Data Storage Media" *Proc. SPIE-Int. Soc. Opt. Eng.* **1248**, 65-73, 1990.

4. D. A. Jones "Principles and Prevention of Corrosion" Macmillan Publishing Company: New York; 1992.

5. D. E. Nikles and C. E. Forbes "Accelerated Aging Studies for Polycarbonate Optical Disk Substrates" *Proc. SPIE-Int. Soc. Opt. Eng.* **1499**, 38-41, 1991.

6. G. D. Cooper and B. Williams "Hydrolysis of Simple Aromatic Esters and Carbonates" *J. Org. Chem.* **27**, 3717-3720, 1962.

7. H. E. Bair, D. R. Falcone, M. Y. Hellman, G. E. Johnson and P. G. Kelleher "Hydrolysis of Polycarbonate to Yield BPA" *J. Appl. Polym. Sci.* **26**, 1777-1786, 1981.

8. J. A. Barrie "Water in Polymers" in "Diffusion in Polymers: J. Crank and G. S. Park, Eds., Academic Press, New York, NY, 1968.

9. R. B. Bird, E. N. Lightfoot, W. E. Stewart *Transport Phenomena* John Wiley and Sons, New York (1960).

10. F. J. Norton "Gas Transport Through Lexan Polycarbonate Resin" *J. Appl. Polym. Sci.* **7**, 1649-1659, 1963.

11. G. Astarita and Sarti *Polym. Sci. Eng.* **18**, 388-395, 1978.

12. W. Abbott "Corrosion of Electrical Contact: Review of Flowing Mixed Gas Test Development" *Br. Corros. J.* **24**,153 1989.

13. A. Djali, D. Seng, W. Glatfelter, H. Lambropoulos, J. Judge Study of the Stability of Metal Particle Data Recording Tape" *J. Electrochem. Soc.* 138, 2504-2509, 1991.

14. D. Speliois "Corrosion of Particulate and Thin Film Media" *IEEE Trans. Magnetics* **26**, 124-126, 1990.

15. T. Yamamoto, K. Sumiya, A. Miyake, M. Kishimoto, T. Taniguchi "Study of Corrosion Stability in Metal Particulate Media" *IEEE Trans. Magnetics* **26**, 2098-2100, 1990.

16. Y. Okazaki, K. Hara, T. Kawashima, A. Sato, T. Hirano "Estimating the Archival Lifetime of Metal Particulate Tape" *IEEE Trans. Magnetics*, **28**, 2365-2367, 1992.

17. M. C. A. Mathur, G. F. Hudson, L. D. Hackett "A Detailed Study of the Environmental Stability of Metal Particle Tapes" *IEEE Trans. Magnetics*, **28**, 2362-2364, 1992.

18. M. R. Parker, S. Venkataraman and D. DeSmet "Magnetic and Magneto-Ellipsometric Evaluation of Corrosion in Metal-Particle Media" " *IEEE Trans. Magnetics*, **28**, 2370-2372, 1992.

19. J. Corcoran "Archival Stability of Metal Particle Tape" NASA Conference Publication 3165, Vol. II, 187-203, 1991.

20. P. Sides and G. Spratt "An Investigation of the Archivability of Metal Particle Tape" *IEEE Trans. Magnetics*, **30**, 4059-4064, 1994.

Functionality and Performance Evaluation of File Systems for Storage Area Networks (SAN)

Martha Bancroft, Nick Bear, Jim Finlayson,
Robert Hill, Richard Isicoff and Hoot Thompson

Storage Technologies Knowledge Based Center,
Department of Defense

Patuxent Technology Partners, LLC
11030 Clara Barton Drive
Fairfax Station, VA 22039
703-250-3754 Voice
703-250-3742 Fax
hoot@patuxent-tech.com

Abstract

The demand for consolidated, widely accessible data stores continues to escalate. With the volume of data being retained mounting as well, a variety of markets are recognizing the advantage of shared data in terms of both cost and performance. Traditionally, common access has been addressed with network-attached fileservers employing data sharing protocols such as the Network File System (NFS). A new approach, poised to deliver high bandwidth access by multiple, heterogeneous platforms to a common storage repository at reduced cost, is beginning to emerge. Storage Area Networking (SAN) is an open-storage architecture designed to eliminate many of the traditional bottlenecks associated with secondary and tertiary storage devices. Conventional high performance computing (HPC) sites and compute-intensive production sites can benefit from such architectures as the need to share computational input and output data sets expands and the mix of computational platforms continues to diversify.

Recognizing the potential value of SAN solutions in their overall data management roadmap, the Storage Technologies Knowledge Based Center of the Department of Defense commissioned a research project in mid-1999 to evaluate the functionality and performance of emerging SAN technologies. The initial focus has been on SAN file systems that offer management of disk-resident data. The desire, however, is to expand the effort to include other traditional data storage functions such as backup, hierarchical storage and archiving using tape technologies. The underlying goal is high bandwidth and reliable access to data with guaranteed long-term retention while presenting a seamless and transparent interface to the users regardless of data location. Operational stability and ease of administration are key requirements as is overall data integrity. When complete solutions will be available and just how robust the family of products will be remains unclear. The magnitude of this challenge is realized when considering that production use of these technologies will entail serving numerous, likely

heterogeneous clients managing a variety of file sizes (tens of kilobytes to multiple gigabytes) and dealing with a mix of applications and access patterns.

As a starting point for the testing, the Center established an environment that features a pair of SGI™ Origin™2000s, two SGI 320 Windows NT® platforms and a fibre channel switch fabric with shared connectivity to over one terabyte of RAID storage. This configuration is expected to grow in number and types of computers (operating systems) as well as with the addition of fabric-attached tape technologies. This preliminary report deals with using the environment to evaluate third-party SAN file systems and related infrastructure technologies. It is a snapshot in time with only initial testing completed. More comprehensive, on-going status and plans, observations and performance data are available on-line at

http://www.patuxent-tech.com/SANresearch

During this stage of the evaluation, each file system product is being exercised to determine its performance under load, its operability and scalability as a function of clients and traffic, and its overall functionality and usability. The motivation is to assess the readiness of SAN file systems to move into production and set realistic timeframe expectations for making such a transition. Although this initiative is conducted under the auspices of the Department of Defense, this research should prove relevant to any large data center operation.

1 Introduction

Several definitions of a Storage Area Network (SAN) exist as related to common, shared repositories of data. The implementation of interest is one that permits true data and/or file sharing among heterogeneous client computers. This differentiates them from SAN systems that permit merely physical device sharing with data partitioned (zoned) into separate file systems. Refer to Figure 1 for a depiction of a notional SAN system. The architecture is broken into three basic elements: SAN clients, a switch fabric and shared storage. The software orchestrating the architecture is what unites the components and determines exactly how these elements behave as a system. The optimum vision is a single file system managing and granting access to data in the shared storage with high bandwidth fibre channel links facilitating transfers to and from the storage.

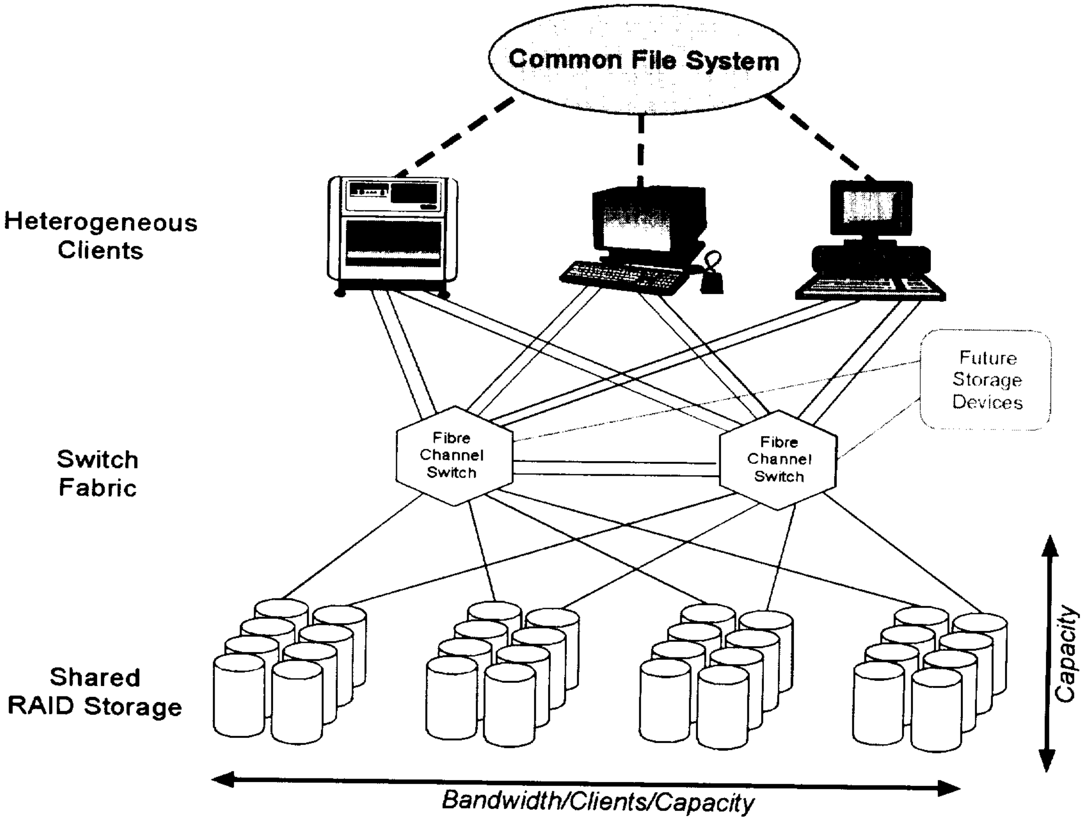


Figure 1. Notional Storage Area Network (SAN)

The advantages of the topology are readily apparent:

- File transfer performance as seen by the client compares with that of directly attached storage.
- The switch fabric can be expanded horizontally by adding switches (client and storage ports) to increase overall system bandwidth.
- Individual fibre channels can be added, combined and striped across to increase bandwidth between an individual client and storage.

- Multiple routes through the fabric between the clients and storage avoid single point failures and/or isolating data.
- Storage depth can be increased by adding or using higher density devices.
- The fabric topology can be expanded to include other storage technologies such as tape drives either directly or by using bridges.

The functioning of the common file system along with how files are opened, closed, read, written, etc. is fundamental to the operation of the SAN. File system control and metadata can co-exist with one of the application clients or be hosted on a dedicated computer. Metadata and locking information can be stored locally or on the SAN itself. A variety of implementations are technically feasible, each with its own functionality and performance implications.

2 Requirements Analysis and Test Planning

Recognizing the potential value of SAN solutions in their overall data management roadmap, the Storage Technologies Knowledge Based Center of the Department of Defense commissioned a research project in mid-1999 to evaluate the functionality and performance of emerging SAN technologies. The initial focus has been on SAN file systems that offer management of disk-resident data. The desire, however, is to expand the effort to include other traditional data storage functions such as backup, hierarchical storage and archiving using tape technologies. The underlying goal is high bandwidth and reliable access to data with guaranteed long-term retention while presenting a seamless and transparent interface to the users regardless of data location. Operational stability and ease of administration are key requirements as is overall data integrity. When complete solutions will be available and just how robust the family of products will be remains unclear. The magnitude of this challenge is realized when considering that production use of these technologies will entail serving numerous, likely heterogeneous clients managing a variety of file sizes (tens of kilobytes to multiple gigabytes) and dealing with a mix of applications and access patterns.

2.1 Requirements Drivers

A SAN file system, when deployed in the production environment, will be expected to maintain a very high level of performance, interoperability, maintainability and availability. Accordingly, the research effort is evaluating the attributes presented in Table 1 relative to the file system products under test. Note that this list reflects the current testing bias. Future activity will stress the interaction of the disk-based SAN technologies with a broad range of other storage functions such as Hierarchical Storage Management (HSM) software, backup software and magnetic tape devices.

2.2 Product Selection

The initial focus has been on researching and testing currently available third-party SAN file systems. Although on the surface the market appears rich with SAN file system offerings, only four products currently are ready for evaluation that meet the Center's criteria and configuration restrictions. They are listed in Table 2.

Table 1 - Requirements Drivers

Item	Parameters
1	Shared concurrent reading and writing of a single file
2	High performance throughput for a wide range of file sizes, with an emphasis on small files
3	Appropriate locking mechanisms at file and sub-file level
4	Sustainable client bandwidth ranging from 500 megabytes/sec to 1 gigabyte/sec
5	High aggregate bandwidth through entire fabric (effectively equal to the number of clients times the desired per-client bandwidth)
6	Low latency for data access
7	Scaling in terms of number of clients, amount of storage, metadata management and maximum number of files supported
8	Transparent integration of file system into existing systems, allowing ease of use
9	Existing user base with support for a variety of common applications
10	Heterogeneous mix of operating systems
11	Ability to serve clients not directly attached to the SAN fabric
12	Additional file system functionality such as executable support, ability to use file system to boot from, etc.
13	SAN volume management features
14	HSM support
15	Backup support
16	Comprehensive set of administrative tools for configuration, monitoring and troubleshooting, allowing ease of maintainability and operation
17	Full range of security features
18	Highly available and high-integrity overall operation

Table 2 - SAN File System Products

Product	Developer
CentraVision™ File System (CVFS)	MountainGate Imaging Corporation/ Advanced Digital Information Corporation (ADIC)
SANergy™	Mercury Computer Systems, Inc./ Tivoli Systems
DataPlow™ SAN File System (SFS)	DataPlow, Inc.
Global File System (GFS)	University of Minnesota with support from NASA, the Department of Defense and several corporations.

A separate initiative is evaluating SGI's Clustered SAN Filesystem (CXFS™). Note too that the market is already experiencing consolidation as evidenced by ADIC's acquisition of MountainGate, Tivoli's acquisition of the SANergy unit of Mercury, and Hewlett® Packard's acquisition of Transoft Networks, Inc.

Selection for this round of testing was based on a combination of factors. The primary criteria used were:

- Architectural diversity and technical approach.
- Support for heterogeneous clients running the most recent versions of target operating systems with emphasis on the latest versions of IRIX™.
- Existence of a product roadmap noting client operating support plans and addressing operational issues.

Given the overall excitement about SAN technologies and the projected growth of the market [1], other products will warrant evaluation as they mature. Candidates include the Concurrent Data Networking Architecture™ (CDNA)™ by DataDirect Networks, Inc. and FibreNet by Transoft Networks. Also under review are products from the VERITAS® Software Corporation and the EMC Corporation.

2.3 Testbed Configuration

As a starting point for the testing, the Center established an environment that includes two SGI Origin2000s and two dual controller SGI RAID systems (over 1 terabyte of raw storage) interconnected via two 16-port switches: one Storage Technology Corporation unit (Brocade Communication Systems, Inc., OEM) and the one Brocade unit (reference Figure 2).

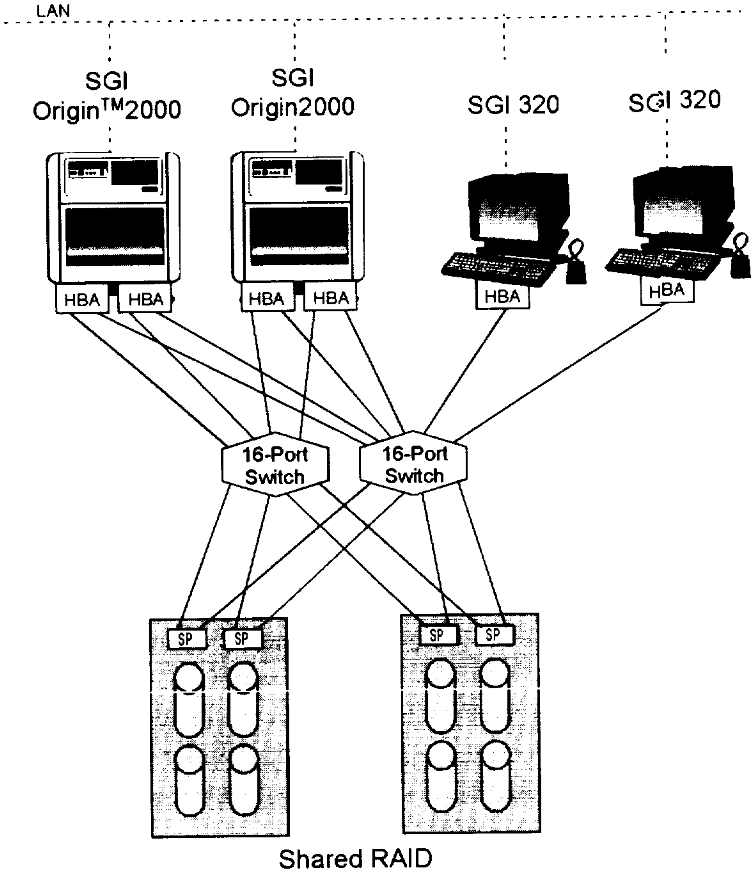


Figure 2. SAN Research Testbed Configuration

Each SGI Origin2000 has a pair of dual channel Prisa host bus adapters (HBA) for connectivity to the switch fabric. Two SGI 320 NT systems also are included for those file system products dependent upon a separate, NT-based metadata controller. They also facilitate heterogeneous SAN client testing. One of the SGI 320s uses an Emulex HBA; the other uses a Qlogic card. Both SGI 320s can, as an option, be booted under Linux. Low-bandwidth communication between the various computers is via traditional 100BASE-T LAN technology. Overall connectivity is flexible and changeable to support the testing requirements as they evolve.

Each RAID system (two total) is configured with four 8+1 RAID 3 logical units (LUN), with two LUNs assigned to each controller. Sustainable bandwidth peaks at 75 megabytes/sec per LUN. Configured usable storage is 576 gigabytes with some disks left unbound.

Table 3 provides a list of the key components with respective product numbers.

Table 3 – Research Testbed Hardware and Software Components

Vendor	Component
Origin™2000	IRIX Operating System
	Prisa NetFX-XIO64 HBA
SGI 320	Windows NT
	Red Hat™ Linux
	Emulex LP7000 HBA
	Qlogic 2200F
Storage Technology Corporation	Fibre Channel Switch 4000
Brocade Communication Systems, Inc.	SilkWorm® 2800
SGI RAID	SP THOR Disk Controller
	9GB Barracuda (ST1917FC)

2.4 Test Planning

The test planning is being shaped by the following objectives:

- Characterize the performance of the individual SAN file system products as a function of file access demands including the ability to stripe files across HBAs, switches and storage elements.
- Explore hot spots and scalability of the products as a function of load and file system fragmentation.
- Compare the performance of SAN file systems to the native file system and traditional file sharing techniques.
- Evaluate operational attributes of the different SAN configurations with respect to administration, availability and maintenance.
- Investigate mechanisms for serving SAN-based data to clients indirectly attached to the fabric via a server (such as NFS).

The projected outcome of the SAN testing is a qualitative and quantitative critique of the products under review measured against the requirements drivers outlined in Section 2.1.

The experiments are being conducted over a range of operating conditions. The test cases envisioned range from the simplest of constructs—single channel writes and reads from a single Origin2000—to multi-channel, multi-client mix load scenarios. In some cases the tests purposely overextend the capability of the system in order to assess the functionality and performance during saturation or when limited bandwidth is forced to be allocated across several active client channels.

2.4.1 Qualitative Testing

Qualitative review will consider the predictable list of product attributes. Of interest is:

- Quality of the documentation
- Ease of installation and configuration
- Ease of use
- Availability of administrative tools for monitoring and troubleshooting
- Transparency to user
- Fault tolerance
- Diagnostic capabilities
- Security features
- Volume management features
- File locking capabilities

2.4.2 Quantitative Testing

Quantitative testing on the other hand will be more performance oriented and is focused on calibrating two fundamental characteristics of the SAN file systems: metadata management and file system throughput as a function of load. The tests are being designed to present stressful yet operational-like conditions. Where possible, industry recognized benchmarks will be used. Several variables, many of which interact, will likely affect the performance of the different products. Most important perhaps are those that are administrator definable when building and instantiating a given file system. Given that the number and type of client access patterns will vary greatly by installation, it is critical to understand how and whether a file system can be tuned to optimally handle the expected workload. Adjustable parameters typically include the following:

- Record (block) size or the subdivision of file
- Stripe width or the size of the data block written to a given logical (or physical) disk in a group of disks that compose a file system
- Mapping of logical (or physical) disks to RAID controllers and HBAs.

2.4.2.1 Metadata Management

The metadata management tests are being designed to measure the number and type of metadata operations that can be accommodated in a given time for single and multiple-client scenarios. This is critical given the assumption that a single, common file system is responsible for data flow in a SAN with potentially a large number of users. The key issue is whether there are any hard scaling limitations in terms of number of clients or number of files. Also important is determining under what conditions latency becomes unacceptable from an access-to-first-byte perspective.

These tests are coming from two sources. One source is project-specific scripts run from single, isolated clients and/or from multiple clients concurrently. The scripts will initiate a large number of metadata-related operations without the associated data I/O while calculating the time per operation. Examples of metadata operations include:

- File open/close
- Get/set file attributes
- Create/delete file
- Rename file
- Make/delete directory

Third party benchmarks are also being considered as the second source. For instance, PostMark, a benchmark by Network Appliance, Inc., is a candidate. It is publicly available at

http://www.netapp.com/

2.4.2.2 File System Throughput

Throughput tests are being developed to measure sustainable transfer rates as a function of number of clients and access patterns, both directly to clients on the SAN, and also to clients not directly attached to the SAN fabric. A mix of test programs will be used, some publicly available, such as SGI's lmdd, while others will be simple C programs written specifically for this project. Also being considered is taskMaster, vxbench and lmbench. taskMaster is useful for simultaneously running variants of the same command on multiple computers.It is available on the GFS website:

http://www.globalfilesystem.org/

vxbench, developed by the VERITAS Software Corporation, provides for multi-threaded testing. Lmbench is a performance analysis tool distributed by BitMover, Inc., at:

http://www.bitmover.com/lmbench

Data will be gathered to measure the behavior of the file systems under normal conditions as well as stress in the midst of allocates, de-allocates, reads and writes, and fragmentation. The method for exercising a file system is multi-step:

1. Measure data transfer rates for a small subset of file sizes, transfer sizes, and access patterns using nominal file system build parameters. Repeat the test while adjusting the build parameters until an optimum performance point is determined.
2. Once the optimum build parameters are set, exercise the file system for individual and multiple clients by initiating:
    a. Single client, single process operations using different file and host block sizes for both reads and writes, sequential and random.
    b. Single client, multiple process operations to either the same or different files, for a predetermined subset of file and host block sizes for sequential versus random accesses, read contention and write contention, and the classic single writer, multiple readers.
    c. Multi-client operations running the same basic script against the same or different files for a predetermined subset of file and host block sizes for sequential versus random accesses.

3. Execute a final set of tests to determine the benefit of configuring multiple file systems with different build parameters as a method to increase total SAN throughput in mixed workloads.

3 SAN File Systems Overview

The SAN file system products being evaluated share certain fundamental characteristics that under optimal conditions tend to even out their performance. The objective of all the SAN file systems, at least from the Center's perspective, is to eliminate file servers between clients and storage with minimum or no impact to the controlling applications. Control information is typically separated from data traffic and in some architectures the two are isolated on completely separate networks. Clients have connectivity to storage via a switch fabric layer that provides the performance of directly attached disks. This allows data to be transferred at relatively high percentages of peak fibre channel bandwidth (100 megabytes/sec per link). All the approaches under test permit multiple HBAs per SAN client, increasing the potential bandwidth per client to a multiple of the base fibre channel rate. Also, the file systems are typically exportable, providing access to SAN resident data by clients that are not directly connected to the SAN switch fabric. Figure 3 depicts generic SAN data and control flow. The diagram shows the fundamental transactions that usually occur—exchange of metadata between requesting SAN client and a third-party metadata manager followed by the data transfer between the client and shared storage via the fibre channel fabric.

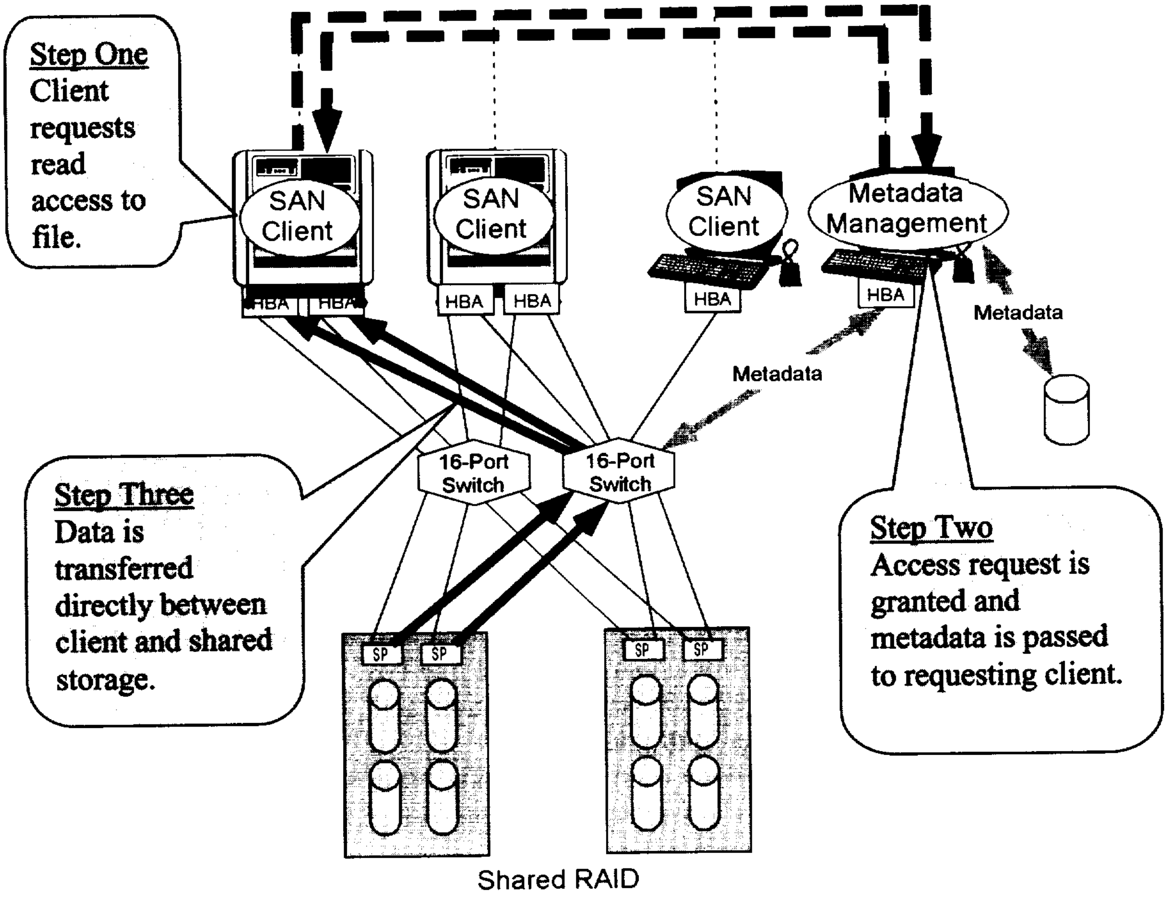

Step One
Client requests read access to file.
SAN Client
SAN Client
SAN Client
Metadata Management
HBA
HBA
HBA
HBA
HBA
HBA
Metadata
Metadata
Step Three
Data is transferred directly between client and shared storage.
16-Port Switch
16-Port Switch
Step Two
Access request is granted and metadata is passed to requesting client.
SP
SP
SP
SP
Shared RAID

Figure 3. Generic File System Data and Control Flow

Differences in the products show up in two primary aspects of the designs. The first aspect is the approach taken to deal with the file system metadata both in terms of where it is stored (locally or on the SAN) and whether it is centralized or distributed. The metadata design has direct effects on performance, scaling and availability. The second aspect is the relation of the SAN client software to the host operating system. How client software is positioned in the software stack impacts performance and also ties directly to the ease of porting it to new revisions and/or to different operating systems. Table 4 summarizes the key attributes of the products being tested. Subsequent sections elaborate on the overall design approach of each

Table 4 - Product Summary of Key Attributes

Product	SAN File System Design	Metadata Management	1.1.1.1 Supported Operating Systems
CentraVision File System	Proprietary	Centralized	IRIX 6.2 to 6.5 NT 4.0
SANergy	Proprietary	Centralized	IRIX (all current releases) Solaris (all current releases) Mac 8.0+ NT 4.0 AIX (all current releases) Compaq Tru64 UNIX™ (all current releases)
DataPlow SFS	Proprietary	Centralized/ Distributed	IRIX 6.2, 6.3, 6.5 Solaris 7 and 8
GFS	Open Source	Distributed	Linux

GFS is notably not heterogeneous but inclusion is warranted given the current popularity of the open source model of software development. To date, CVFS and SANergy have been installed and initial testing has started.

3.1 CVFS (Version 1.3.8)

CVFS is a distributed file system designed specifically [2] for fibre channel and SAN technology. CVFS provides sharing of common network storage across multiple heterogeneous systems. The CVFS file system is a hybrid implementation transferring data directly between fabric-attached storage and the SAN client's application, while using TCP/IP transports under a client/server model for control and metadata. CVFS is designed for sequential bulk-data file transfers (megabyte or greater) that are typically streamed into an application. This exploits the read-ahead capabilities and serial nature of the I/O schema. Performance equals or surpasses that of the local file system for well-formed I/O.

The key element of the CVFS is the File System Services (FSS). The FSS is a user-level application that acts as a server for the file system clients. It is responsible for the file system's name space, file allocation, bandwidth management, virtual file management and configuration. The FSS is a POSIX compliant (IEEE Std 1003.1-1990), multi-threaded application that runs on either an IRIX or NT-based host. SAN clients communicate with the FSS for allocates, reads, writes, etc. over a typical LAN to obtain access to SAN-resident data in a fashion similar to interchanges with the local operating system. Once acknowledged, file extents are passed from the FSS to the requesting client via the LAN, then data is transferred directly between the client and the shared storage via the fibre channel fabric. All communication packets between the FSS and its clients conform to network endian with 64-bit extensions. It does not need to run on a workstation that is physically connected to Fibre Channel fabric because it communicates with the clients via TCP/IP sockets. Metadata is stored using the FSS host's native file system and local system disk. Note also that the FSS host also can be a SAN client.

On the client side, CVFS is written as a file system driver operating at the kernel level in order to transparently attach CVFS managed storage to the client operating system. In IRIX, this is the Virtual File System (VFS) layer; in Windows NT, it is the File System Driver (FSD) layer. Each port provides a completely native interface and is written specifically for the candidate platform. The remainder of client software, however, provides for significant code re-use. Each client operates as if it is directly attached to local storage. The data resides on the managed storage in CentraVision file format. In general, the stored data format can be considered raw data. CVFS uses 64-bit "containers" and accommodates both "big-endian" and "small-endian" file structures. CVFS looks like a local file system with utilities such as cvfsck to check the file system for consistency. Currently, CVFS mounts the NT file system as a network drive. However, in a forthcoming release, the NT version will have a local drive implementation. On IRIX, it currently appears as a local-drive. The final result is that all clients (no matter what platform) perceive the data as native.

Several administrative decisions that directly impact performance must be made when building a CVFS file system:

- Disks (LUNs) are specifically labeled as CVFS entities.
- Disks (LUNs) are assigned to Stripe Groups. This assignment allows for increasing both the bandwidth and storage depth of a given file system.
- Block size and Stripe Group Breadth are adjustable, permitting tuning of the file system versus the application/user access patterns.
- Affinities can be established so that specific files can be stored in the most performance favorable fashion.

Another important operational consideration is CVFS behavior in the event of failures. When a client fails, transactions by the client in transit are accepted into the FSS and are committed to the metadata files. All connections are then cleaned-up with the failed client. When the client re-establishes contact, the client's picture of the SAN is re-established through normal system recovery operations. To the user and to the file system there are no apparent seams to the FSS picture other than the possible transactions lost on the client (that didn't make it to the server) during the failure.

Currently, FSS switchover to a redundant server is a manual operation. However, the release of a more resilient version is imminent. The new FSS design requires that the metadata be placed on a shared storage device, either the SAN itself or any device accessible by at least two servers. Also in the new version, the FSS becomes a journaled file system. This feature provides for hard-crash integrity and very rapid recovery time. Any platform that supports the FSS can be a participant in the fault tolerant configuration. NT and IRIX servers can freely exchange server responsibilities. When a primary and one or more secondary FSSs are configured, the secondary FSSs are poised to take over the service. They are fully operational and have complete access to file system metadata including in-process I/O transactions. If the primary fails, a vote is executed to determine which secondary can take over. There are two ways the vote is stimulated:

- Lack of response from the primary server—if a client or administrator tries to access the FSS and it is does not respond.
- No update to the Arbitration Control Block on the shared metadata Stripe Group – a running FSS must update its respective "heart-beat" block on the metadata Stripe Group.

For additional information regarding CVFS refer to
http://www.centravision.com/

3.2 SANergy (Version 1.6)

SANergy is a hybrid of conventional networking and direct attached storage [3]. Now patented, it is an operating system extension built on standard system interfaces. SANergy fully supports the user interface, management, access control, and security features of the native host file systems, providing all the file system management, access control and security expected in a network. SANergy clients can be heterogeneous with data being freely shared by all clients attached to shared storage.

SANergy operations center around the Metadata Controller (MDC) that provides centralized metadata management. The Version 1.6 SANergy MDC is based on a Windows NT environment and the NT File System (NTFS). NTFS inherently provides key features such as security, transaction logging and journaling. SANergy intercepts data transactions, then separates and accelerates them using high-bandwidth transports typically fibre channel. Metadata is intertwined with the real data on the shared storage system. Hence, metadata traffic is mixed with data transfers through the switch fabric. The metadata is exchanged between the MDC and SAN clients using standard LAN technologies. NFS is a UNIX client requirement necessitating the NT-based MDC to run an NFS server application. CIFS is used to communicate with NT clients. When a file operation is requested by a SAN client, extent information is retrieved from the appropriate NTFS volume and is passed back to the requester via the MDC. SANergy supports locking primitives down to the byte level with coordination provided by the MDC.

On the client side, SANergy acts as a layered filter driver. It sits on top of the file system(s) either handling an I/O request directly, or passing it on to its natural path, or both. The code is kernel/driver code and is loaded like any other device driver. Since it is wrapped around the primary drivers supplied by the operating systems, SANergy's exposure to any major systems internal change is minimized. Clients have no prerequisite knowledge of NTFS. Rather, all they need is the block location and order, information that is provided by the MDC. Ultimately data is delivered in a format acceptable to and usable by any application built for cross platform environments.

When building a SANergy file system several operational considerations are worthy of note:

- Disks (LUNs/volumes) are labeled, partitioned and formatted as NTFS file systems using the NT Disk Administrator, a process that writes over any disk resident file and/or configuration information. The MDC must be connected to the switch fabric regardless of whether it is also participating as a SAN client.
- Disks (LUNs) can be assigned to Stripe Sets that allows for both increasing the bandwidth and storage depth of a particular file system. Stripe size is fixed at 64KB.
- NTFS supports multiple partitions (file systems) per volume.
- File record size is adjustable, permitting tuning of the file system versus the application/user access patterns.

The SANergy architecture is flexible in that the MDC can also be an active SAN client. Perhaps the biggest differentiator for SANergy however is the range of supported SAN client operating systems as noted in Table 4. Also, a new version of SANergy (2.0) recently has been released. It supports failover, a critical requirement in operational environments, and also a Sun UFS-based version of the MDC. Failover is handled by an additional product called XA. Any machine running SANergy software also can run the XA software with any XA machine watching any number of MDCs. Should one fail, it will become the MDC for whatever volumes that were owned by the failed machine. Plus, it will send "remap" messages to other SANergy clients (with or without XA software) to remap any mapped shares to the new MDC. The new Sun MDC reportedly provides the key features of the NT version while improving greatly on the striping options allowed when establishing the SAN file system. Although SANergy is most powerful in large file applications, a version is being developed that will be more amenable to small file applications.

For more information, refer to the SANergy web site at
http://www.sanergy.com/

3.3 DataPlow (Version 1.2)

The DataPlow SAN File System (SFS) is a distributed file system with full operating system integration. A key design feature of DataPlow SFS is the separation of metadata into two fundamental components – the higher level namespace-oriented information managed by a metadata server and the more detailed, extent-level data stored directly on the shared disks. File operations require a SAN client to communicate with the metadata server to obtain the location of the more fine-grained information that the client reads directly from the shared storage. In order to facilitate heterogeneous environments, SFS software stores metadata on the server and shared disks in a format that is operating system independent.

The metadata server can be hosted by any one of the SAN clients or it can be free standing. In either case, all SAN clients must have TCP/IP connectivity with the metadata server. SFS clients are able to share SAN file data with LAN and WAN-based clients of any platform through use of traditional protocols such as NFS, CIFS, and HTTP.

If configured for high-availability, metadata server functionality can failover to a secondary server should the primary fail. Just as critical, the failure of an individual SFS client should not harmfully affect the entire SAN. The metadata server simply disconnects the client and releases locks held by the client. Traditional techniques (journaling, file system utilities, etc.) help ensure overall data integrity.

Several administrative options are available when building an SFS file system:

- SFS is able to utilize various commercial volume managers. This flexibility permits numerous striping and mirroring configurations that accommodate a wide range of bandwidth, scalability, cost, and availability requirements. Volume managers that support multiple operating system platforms can be used in conjunction with SFS software to enable heterogeneous file sharing.
- File system block size is adjustable. The block size parameter is used when tuning for small files and reduced fragmentation.
- File systems may be partitioned into several segments in order to exploit parallelism during block allocation and de-allocation. Depending upon the physical device configuration, segmentation further enhances parallelism during data transfers. Segmentation is hidden from users and applications.

DataPlow SFS supports common operations such as synchronous and asynchronous buffered I/O. Additionally, SFS provides support for direct I/O, a caching policy that bypasses the system buffer cache in order to achieve near raw performance. SFS invokes direct I/O either after an explicit system call request by the user application or automatically once file request sizes reach a predetermined size.

Currently, SFS operates in IRIX and Solaris environments. Additional client implementations are in development. Also in development are HSM interfaces such as DMAPI to improve backups, restores, etc.

For additional information refer to

http://www.dataplow.com/

3.4 GFS (Antimatter Anteater)

GFS is a distributed file system based on shared, network-attached storage [4]. GFS is built on the premise that a shared disk file system must exist within the context of a cluster infrastructure of some kind for proper error handling and recovery and for the best performance. SAN clients service only local file system requests and act as file managers for their own requests; storage devices serve data directly to clients. GFS uses callbacks from clients requesting data held exclusively by another client, so that the client holding the data exclusively releases it some time after the request. This implies direct client-to-client communication. Overall the design permits aggressive metadata and data caching resulting in GFS performance being on a par with local Linux file systems like ext2fs.

GFS provides transparent parallel access to storage devices while maintaining standard UNIX file system semantics—user applications still see only a single logical device via the standard *open, close, read, write* and *fcntl*. This transparency is important for ease of use and portability. However, GFS allows some user control of file placement on physical storage devices based on the appropriate attributes required such as bandwidth, capacity, or redundancy.

The GFS structure and internal algorithms differ from traditional file systems, emphasizing sharing and connectivity in addition to caching. Unlike local file systems, GFS distributes file system resources, including metadata, across the entire storage subsystem, allowing simultaneous access from multiple machines. *Device Locks* are mechanisms used by GFS to facilitate mutual exclusion of file system metadata [5]. They also are used to help maintain the coherence of the metadata when it is cached by several clients. The locks are implemented on the storage devices (disks) and accessed with the SCSI device lock command, *Dlock*. The Dlock command is independent of all other SCSI commands, so devices supporting the locks have no awareness of the nature of the resource that is locked. The file system provides a mapping between files and Dlocks.

To allow recovery from failures, each GFS machine writes to its own journal. When a GFS machine modifies metadata, this is recorded as a single transaction in that machine's journal. If it fails, other machines notice that its locks have timed out, and one of the other machines replay the failed machine's logs and re-boots the failed machine. Other machines in the GFS cluster can keep accessing the file system as long as they do not need any metadata in the failed client's journal.

As an alternative to disk-based locks, GFS also can use a lock daemon running on any machine accessible to the GFS cluster over IP. Hence, special SCSI disks with DLOCK firmware are not required to run GFS. GFS can also be run without locks as a local file system. Lastly, lock handling has been modularized so that GFS can use almost any globally accessible lock table. This positions GFS to exploit the coming developments in Linux clustering, where highly scalable clusters will be available (to thousands of nodes) with fully recoverable, distributed lock manager technology.

Currently GFS is only operational in a Linux environment. An open source operating system, such as Linux, is ideal for developing the new kernel code required to implement the GFS constructs [6], [7]. However, development of other UNIX variants is likely in the future, including FreeBSD and IRIX.

For additional information on GFS refer to
http://www.globalfilesystem.org/

4 Initial Observations

Testing to date has dealt largely with establishing the basic functionality of the SAN environment and understanding the nuances introduced by the switch fabric environment. Some key activities have included:

- Learning the capabilities and restrictions of the "plug and play" functionality of fibre chanel switches, HBAs and storage devices.
- Establishing the most advantageous RAID configuration with the objective being to maximize the disk throughput available to the various file systems.
- Determining proper procedures for sequencing equipment on-line to ensure that the fabric is operational.
- Using the information available from the fibre channel switches to manage and monitor the fabric activity and status.

Time also has been spent investigating the benchmarking products commonly available for the various areas of quantitative testing to be carried out. By using standard benchmarking products, results can be presented in a way allowing comparison with other industry-sanctioned testing and evaluation efforts.

The CentraVision File System and SANergy have been installed on the testbed and preliminary experiments have been conducted. CVFS has been exercised hosting the FSS both on the SGI IRIX and Windows NT computers. SANergy has been tested exclusively with a Windows NT-based MDC. Performance testing of simple read/write operations has yielded similar results with both CVFS and SANergy delivering a relatively high percentage of peak bandwidth for large sequential file operations. Additionally both seem to operate as advertised and data sharing across heterogeneous platforms works as evidenced by a rather simple test of exchanging a PDF file. More extensive testing is required and planned, as detailed earlier.

5 Future Testing

Testing beyond the initial configuration and file system products is already being planned. A greater emphasis on archiving and backup technologies is envisioned. Items currently being considered are:

- Additional/different SAN file systems. Notably absent from the discussion are offerings from some of the more prominent companies in the storage and networking industry, specifically the VERITAS Software Corporation and the EMC Corporation. Developments by these and other companies are being monitored for possible inclusion in future testing.
- Additional/different client hardware and operating systems.
- Additional/different disk storage devices.
- Additional fibre channel switch devices.
- Data flow to/from tape systems attached to the switch fabric.

Future activities will rely in part on an expanded test environment. Several technologies – hardware and software – are under consideration.

6 Test Results

Given the continuing and evolving nature of this research effort, a web site has been established to deliver a variety of timely information on-line at

http://www.patuxent-tech.com/SANresearch

It will provide operational reviews of each of the products under test including a pro/con style evaluation as well as any future evaluations that are planned. Also available will be relevant vendor comments regarding the evaluations in addition to public domain plans for future product feature sets especially as they pertain to any noted shortcomings. Market impressions and links to relevant websites also will be provided.

7 Acknowledgments

The authors would like to thank several individuals who provided technical content for the write-ups describing the SAN file systems currently in evaluation. They are:

- MountainGate Imaging Corporation/Advanced Digital Information Corporation
  - Brad Kline, Software Architect
- Mercury Computer Systems, Inc./Tivoli Systems
  - Chris Stakutis, VP-Engineering/ CTO
- DataPlow, Inc.
  - Steve Soltis, CEO

Appreciation is also extended to Matt O'Keefe, University of Minnesota, for his GFS insight and overall support in the editorial process.

References

[1] Robert C. Gray. The 1999 Outlook for Storage Area Networks. Presented at Getting Connected '99 hosted by the Storage Network Industry Association Board. May 19 - 20, 1999.

[2] Brad Kline, Pete Lawthers. CVFS: A Distributed File System Architecture for the Film and Video Industry, a MountainGate White Paper, June, 1999. MountainGate Imaging Systems, Inc. and Prestant Technology, Inc., http://www.centravision.com/

[3] SANergy Users Guide. GS-06-12 7/16/99. Mercury Computer Systems, Inc. Chelmsford, MA 01824-2820.

[4] Kenneth Preslan *et al.*, Implementing Journaling in a Linux Shared Disk File System. In *The Eighth Goddard Conference on Mass Storage Systems and Technologies in cooperation with the Seventeenth IEEE Symposium on Mass Storage System*s, College Park, Maryland, March 2000, (these proceedings) http://www.globalfilesystem.org/Pages/gfspapers.html

[5] Kenneth Preslan *et al.* SCSI Device Locks Version 0.9.5. In *The Eighth Goddard Conference on Mass Storage Systems and Technologies in cooperation with the Seventeenth IEEE Symposium on Mass Storage System*s, College Park, Maryland, March 2000, (these proceedings)

http://www.globalfilesystem.org/Pages/dlock.html

[6] M. Beck, H. Bohme, M. Dziadzka, U. Kunitz, R. Magnus, and D. Verworner. *Linux Kernel Internal*s Addison-Wesley, second edition, 1998.

[7] Alessandro Rubini. *Linux Device Driver*s. O'Reilly & Associates, 1998.

April: A Run-Time Library for Tape-Resident Data

Gokhan Memik, Mahmut T. Kandemir, Alok Choudhary, and Valerie E. Taylor
Center for Parallel and Distributed Computing
Northwestern University, Evanston, IL 60202
{memik,mtk,choudhar,taylor}@ece.nwu.edu
tel: +1 847 467-2299
fax: +1 847 491-4455

Abstract

Over the last decade, processors have made enormous gains in speed. But increase in the speed of the secondary and tertiary storage devices could not cope with these gains. The result is that the secondary and tertiary storage access times dominate execution time of data intensive computations. Therefore, in scientific computations, efficient data access functionality for data stored in secondary and tertiary storage is a must. In this paper, we give an overview of APRIL, a parallel runtime library that can be used in applications that process tape-resident data. We present user interface and underlying optimization strategy. We also discuss performance improvements provided by the library on the High Performance Storage System (HPSS). The preliminary results reveal that the optimizations can improve response times by up to 97.2%.

1 Introduction

We address the problem of managing the movement of very large data sets between different levels of a hierarchical storage system. It is now widely acknowledged that the data set sizes manipulated by scientific codes are getting larger as programmers have access to faster processors and larger main memories. The data sets whose sizes exceed main memories should be stored in secondary and tertiary storages. Although the prices for secondary storage devices are decreasing, tertiary storage devices are becoming increasingly attractive especially for applications that require vast amount of storage capacity which cannot be satisfied by secondary storage devices and for applications which cannot afford the cost or system complexity of a large number of disk drives. There has been a considerable amount of work in addressing the flow of data to and from secondary storage devices (e.g., magnetic disks) [1, 2, 3, 4, 5, 6, 7, 8, 9]. There has also been a significant amount of work on the management of large scale data in a storage hierarchy involving tertiary storage devices (e.g., tapes devices) [10, 11, 12, 13, 14]. Striping has been studied to improve the response time of tertiary storage devices [15, 16].

The Department of Energy's ASCI plan draws an outline of the expected storage requirements for large-scale computational challenges. According to this plan, a large scientific application today is producing 3-30 terabytes of simulation datasets for a run, requiring 3 petabytes of archive capacity. These sizes are excepted to grow to 100-1000 terabytes per run and to 100 petabytes of archive capacity in the year 2004. Even with the assumptions of aggressive improvements in the evolution of storage devices, the data accesses in these applications will take a significant proportion of the overall execution time [17]. On top of this, aggregate data sizes may require the employment of tertiary storage devices. Many of these applications do not demand the entire datasets to be accessed at a given time. So, having means to access portions of the tape-resident datasets efficiently may decrease the time spent in data accesses significantly.

In this paper, we present APRIL, a parallel run-time library, that can be used to facilitate the explicit control of data flow for tape-resident data. Our library can be used by application programmers as well as optimizing compilers that manipulate large scale data. The objective is to allow programmers to access data located on tape via a convenient interface expressed in terms of arrays and array portions (regions) rather than files and offsets. In this sense the library can be considered as a natural extension of state-of-the-art run-time libraries that manipulate disk-resident datasets (e.g., [2, 18]). The library implements a data storage model on tapes that enables users to access portions of multi-dimensional data in a fast and simple way. In order to eliminate most of the latency in accessing tape-resident data, we employ a *sub-filing strategy* in which a large multi-dimensional tape-resident *global* array is stored not as a single file but as a number of smaller *sub-files*, whose existence is transparent to the programmer. The main advantage of doing this is that the data requests for relatively small portions of the global array can be satisfied without transferring the entire global array from tape to disk as is customary in many hierarchical storage management systems. In addition to read/write access routines, the library also supports pre-staging and migration capabilities which can prove very useful in environments where the data access patterns are predictable and the amount of disk space is limited.

The main contributions of this paper are as follows:

- The presentation of a high-level parallel I/O library for tape-resident data. APRIL library provides a simple interface to the tape-resident data, which relieves the programmers from orchestrating I/O from tertiary storage devices such as robotic tapes and optical disks.
- The description of the implementation of the library using HPSS [19] and MPI-IO [3]. We show that it is both simple and elegant to build an I/O library for tape-resident data on top of these two state-of-the-art systems. In this paper, however, we focus on a single processor performance.
- The presentation of preliminary performance numbers using representative array regions and sub-file sizes. The results demonstrate that the library is quite effective in exploiting the secondary storage – tertiary storage hierarchy without undue programmer effort.

Section 2 gives an overview of the APRIL library. Section 3 describes sub-filing and its use in the library. Section 4 briefly explains the implementation and Section 5 presents the user interface. Section 6 gives preliminary experimental results and Section 7 concludes the paper with a summary and an outline of future work.

2 Library Overview

The library provides routines to efficiently perform I/O required in sequential and parallel applications. It can be used for both in-core and out-of-core applications. It uses a high-level interface which can be used by application programmers and compilers. For example, an application programmer can specify what section of an array she wants to read in terms of lower and upper bounds in each dimension, and the library will fetch it in an efficient manner, first from tape to disk and then from disk to main memory. It provides a portable interface on top of HPSS [19] and MPI-IO [3]. It can also be used by an optimizing compiler that targets programs whose data sets require transfers between secondary storage and tertiary storage. It might even be possible to employ the library within a database management system for multi-dimensional data.

At the heart of the library is an optimization technique called *sub-filing*, which is explained in greater detail in the next section. It also uses collective I/O using a two-phase method, data pre-staging, pre-fetching, and data migration. The main advantage of sub-filing is that it provides low-overhead random access image for the tape-resident data. Sub-filing is invisible to the user and helps to efficiently manage the storage hierarchy which can consist of a tape sub-system, a disk sub-system and a main memory. The main advantage of the collective I/O, on the other hand, is that it results in high-granularity data transfers between processors and disks, and it also makes use of the higher bandwidth of the processor interconnection network.

In general, a processor has to wait while a requested tape-resident data set is being read from tape. The time taken by the program can be reduced if the computation and tape I/O can be overlapped somehow. The pre-staging achieves this by bringing the required data ahead of the time it will be used. It issues asynchronous read calls to the tape sub-system, which help to overlap the reading of the next data portion with the computation being performed on the current data set. The data pre-fetching is similar except that it overlaps the disk I/O time with the computation time.

3 Sub-filing

Each *global tape-resident* array is divided into *chunks*, each of which is stored in a *separate sub-file* on tape. The chunks are of equal sizes in most cases. Figure 1 shows a two-dimensional global array divided into 64 chunks. Each chunk is assigned a unique *chunk coordinate* (x_1, x_2), the first (upper-leftmost) chunk having (0,0) as its coordinate. For the sake of ensuing discussion we assume that the sub-files corresponding to the chunks are stored in row-major as depicted in the figure by horizontal arrows.

A typical access pattern is shown in Figure 2. In this access a small two-dimensional portion of the global array is requested. In receiving such a request, the library performs three important tasks:

- Determining the sub-files that collectively contain the requested portion,
- Transferring the sub-files that are *not* already on disk from tape to disk, and
- Extracting the required data items (array elements) from the relevant sub-files from

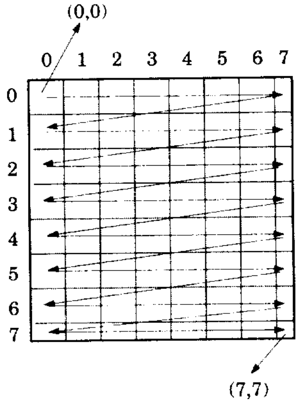


Figure 1: A global tape-resident array divided into 64 chunks.

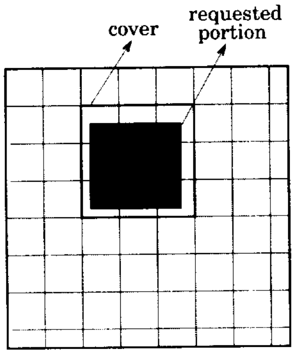


Figure 2: An access pattern (shaded portion) and its cover.

disk and copying the requested portion to a buffer in memory provided by the user call.

In the first step, the set of sub-files that collectively contain the requested portion is called *cover*. In Figure 2, the cover contains the sub-files (1,2), (1,3), (1,4), (2,2), (2,3), (2,4), (3,2), (3,3), and (3,4). Assuming for now that all of these sub-files are currently residing on tape, in the second step, the library brings these sub-files to disk. In the third step, the required portion is extracted from each sub-file and returned to the user buffer. Note that the last step involves some computational overhead incurred for each sub-file. Instead, had we used just one file per global array this computational overhead would be incurred only once. Therefore, the performance gain obtained by dividing the global array into sub-files should be carefully weighed against the extra computational overhead incurred in extracting the requested portions from each sub-file. Our preliminary experiments show that this computational overhead is not too much.

Parallel reads by multiple processors pose additional problems. Consider now Figure 3(a) where four processors are requesting four different sub-columns of a region. The underlying cover contains 28 sub-files. After bringing these sub-files from tape to disk, we have a problem of reading the required sub-portions (sub-columns) for each processor. As stated by del Rosario et al. [20], *collective I/O* is a technique in which processors perform I/O on

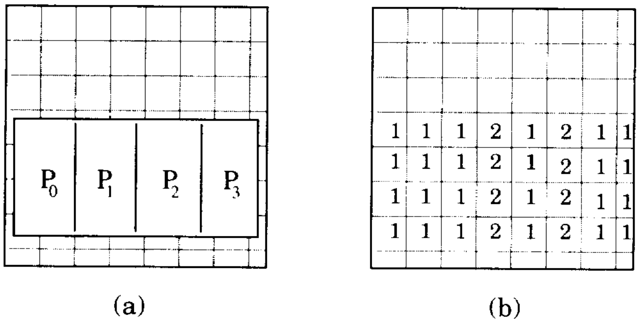


Figure 3: (a) An access pattern involving four processors. (b) The global array with each sub-file marked with the number of processors that share it.

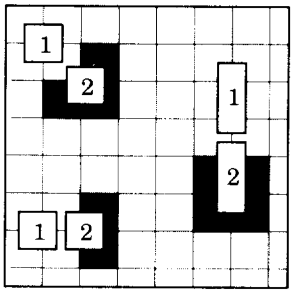


Figure 4: Successive array accesses.

behalf of each other in order to reduce the time spent in disk-I/O at the expense of some extra communication. Two-phase I/O is a specific implementation of collective I/O, which uses the information available about the access and storage patterns. It employs two phases. In the first phase, the processors access the data in a *layout conformant* way (to exploit spatial locality on disk as much as possible) and in the second phase they re-distribute the data in memory among themselves such that the desired access pattern is obtained. While it is quite straightforward how to use collective I/O when we have a single file, in our multiple file case it is not clear how to utilize it. One simple approach might be to use collective I/O for each sub-file on disk. In our example, that would mean calling a collective I/O routine 28 times. A better alternative might be to read the data from disk to memory in two steps. In the first step, the processors that have exclusive access to some sub-files perform these independent accesses. In the second step, for each of the remaining sub-files, we can perform collective I/O using only the processors that request some data from the sub-file in question. Considering Figure 3(b), this collective I/O scheme corresponds to first reading the sub-files marked '1' and then collectively (using two processors) reading the sub-files marked '2'. We plan to implement this last collective I/O strategy in the future.

During successive reads from the same global file it might happen that the same sub-file can be required by two different reads. Assuming that the sub-file in question still resides on the disk after the first read, it is unnecessary to read it again from tape in the second read. In such a case only the other (additional) sub-files required by the current access are read from tape. The situation is shown in Figure 4 for three scenarios. In each case, the

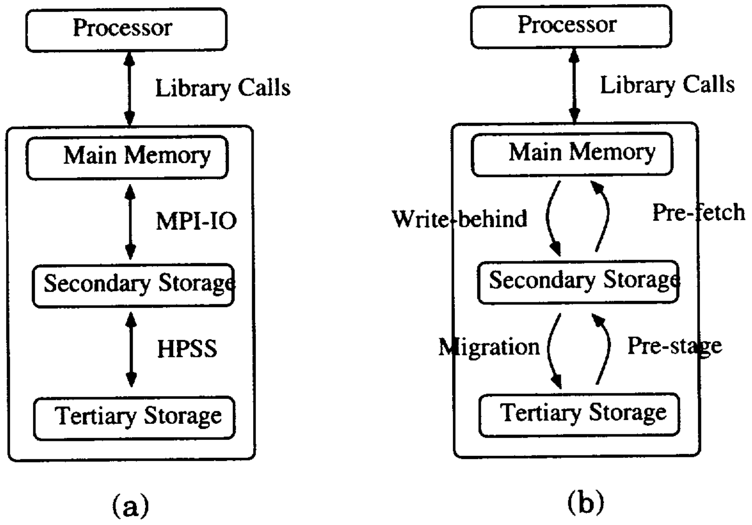


Figure 5: (a) Library architecture. (b) Pre-fetching, pre-staging, and migration.

first portion read is marked '1' and the second portion read is marked '2'. The shaded parts around the second portions correspond to additional sub-files that are needed for the second read. In other words, the library effectively uses the storage hierarchy.

4 Implementation

We are implementing the APRIL library on top of HPSS [19] and MPI-IO [3]. The connections between different components are shown in Figure 5(a). In a read call, the sub-files are first read from tape to disk using HPSS and then from disk to main memory using MPI-IO. As mentioned earlier, we employ collective I/O between disk and memory. In a write call the direction of data-flow is reversed. Figure 5(b) shows the corresponding storage levels for each action described in the following sections.

To store the information about the file and the chunks, we are currently using the Postgres95 database [21]. When a new file is created, the user may enter the necessary information about the chunks. The detailed information about the file creation is given in Section 5. Then this meta-data is stored in the database for later usage. When a user opens a previously created file, the corresponding meta-data about the file and the chunks are read from the database and cached in the memory. Then the following accesses uses this meta-data. The database is informed about the changes when the file is closed. In other words, the database is accessed only in file open and file close.

5 User Interface

The routines in the library can be divided into four major groups based on their functionality – Initialization/Finalization Routines, File Manipulation Routines, Array Access Routines, and Stage/Migration Routines. Table 1 lists some of the basic routines in the library and their functionalities. All the routines listed here are the low level instructions, which should explicitly be called by the user. We are currently adding high level routines to our library,

Table 1: Some of the library routines.

Initialization/Finalization Routines	
Routine	Functionality
T_INITIALIZE	Initializes the library structures
T_FINALIZE	Finalizes the library structures
File Manipulation Routines	
Routine	Functionality
T_OPEN	Opens a tape-resident global file for read/write
T_CLOSE	Closes a tape-resident global file
T_REMOVE	Removes both the sub-files of the file and the corresponding info.
Array Access Routines	
Routine	Functionality
T_READ_SECTION	Reads a rectilinear section
T_WRITE_SECTION	Writes a rectilinear section
Stage/Migration Routines	
Routine	Functionality
T_STAGE_SECTION	Stages a rectilinear file section from tape to disk
T_STAGE_WAIT	Waits for a Stage to complete
T_PREFETCH_SECTION	Pre-fetchs a rectilinear file section from tape (or disk) to memory
T_PREFETCH_WAIT	Waits for a Pre-fetch to complete
T_MIGRATE_SECTION	Migrates a rectilinear file section from disk to tape

which will call the low level routines implicitly.

Initialization/Finalization Routines: These routines are used to initialize the library buffers and meta-data structures and finalize them when all the work is done. The routine to initialize the system has the format

int T_INITIALIZE ().

This routine initializes the connections to the HPSS and the database. It returns a positive number upon successful completion. Similarly, int T_FINALIZE () closes the above mentioned connections.

File Manipulation Routines: These routines are used for creating files, opening existing files, closing open files and removing all the chunks and the information related to a global file. T_OPEN is used for creating new files and for opening existing files. It returns a file handle for later referral to the file. The synopsis of T_OPEN is as follows:

T_FILE T_OPEN (char *filename, char *mode, T_INFO *tapeinfo).

'Filename' stands for the name of the file to be opened. 'Mode' indicates whether the file is opened for read, write, or read/write. 'Tapeinfo' is the structure used for entering the necessary information about the file and chunks. It has fields for the elementsize, number of dimensions, the size of each dimension of the chunk and the size of each dimension of the global file.

Array Access Routines: These routines handle the movement of data to and from the tape

```
int main(int argc, char **argv)
{
 T_FILE exfile;
 T_INFO exinfo;
 int start[2];
 int end[2];

 /* Initialize the library */
 T_INITIALIZE();

 /* Open the file for read.  The exinfo will be filled by the library.
 For creating the file (i.e.  if the file is opened for the first time),
 information about the file should be supplied to T_OPEN via exinfo.*/
 exfile = T_OPEN ("file_1","r", &exinfo);

 start[0] = 0;
 start[1] = 0;
 end[0] = 24000;
 end[1] = 80;

 /* Perform the operation */
 T_READ_SECTION (&exfile, &buf, starts, ends);

 /* Close the file */
 T_CLOSE (&exfile);

 T_FINALIZE();
}
```

Figure 6: An example code for reading from a two dimensional file.

subsystem. An arbitrary rectilinear portion of a tape-resident array can be read or written using these access routines. Let us focus now on T_READ_SECTION. The signature of this routine is

int T_READ_SECTION (T_FILE *fd, void *buffer, int *start_coordinate,
int *end_coordinate)

'fd' is the file descriptor returned by T_OPEN. 'Start_coordinate' and 'end_coordinate' are arrays that hold the boundary coordinates for the section to be read. There are as many elements as the dimensionality of the associated tape-resident global array. This command reads the corresponding elements and stores them in 'buffer'. As discussed earlier, what actually happens here is that the relevant sub-files are read from tape to disk (if they are not on the disk already), and the required sections are read from these sub-files on disk and forwarded to the corresponding positions in the buffer in main memory. An example code for T_READ_SECTION is given in Figure 6. In this example, a 24000×80 portion of the file is read to the buffer. The syntax for the T_WRITE_SECTION routine is almost the same except that the direction of the transfer is reversed.

Stage/Migration Routines: These routines are used to stage and migrate the data between the tapes and the disk sub-system. The command

int T_STAGE_SECTION (T_FILE *fd, int *start_coordinate, int *end_coordinate)

immediately returns and starts a data staging operation in the background from tape to disk. It returns an integer to the application which can be interpreted as a descriptor for the associated pre-stage operation. Note that what is actually performed here is to bring the relevant *sub-files* from tape to disk. Note also that there is no 'buffer' parameter in the signature. The routine

int T_STAGE_WAIT (int pre-stage_descriptor)

can be used to wait for a previously initiated pre-stage operation to complete.

int T_PREFETCH_SECTION (T_FILE *fd, void *buffer, int *start_coordinate,
int *end_coordinate)

is used to start a pre-fetch operation from disk to memory. The parameters are the same as for T_READ_SECTION. It returns an integer which can be used as a pre-fetch descriptor in a later T_PREFETCH_WAIT call.

int T_MIGRATE_SECTION (T_FILE *fd, int *start_coordinate, int *end_coordinate)

starts to migrate the relevant sub-files (i.e., those corresponding to the section described in the signature) from disk to tape. It should be used with care as these sub-files may contain portions of data that will be requested by a later library call.

6 Experiments

The experiments are performed using the HPPS at the San Diego Supercomputing Center (SDSC). We have used the low level routines of the SDSC Storage Resource Broker (SRB) to access the HPSS files. SRB is a client-server middleware that provides a uniform interface for connecting to heterogeneous data resources over a network and accessing replicated data sets [22].

We experimented with different *access patterns* in order to evaluate the benefits of the library. Table 2 gives the start and end coordinates (on a two dimensional global array) as well as the number elements read/written for each access pattern (A through H). Note that the coordinate (0,0) corresponds to the upper-left corner of the array. In each case, the accessed array consists of 50000×50000 floating point elements (10 GB total data). We used two different sub-file (chunk) sizes: small (1000×1000 elements) and large (2000×2000 elements).

Table 3 shows the performance results obtained. For each operation (read or write) we give the response times (in *seconds*) for a naive access strategy and the gains obtained against it using our library which employs sub-filing. The naive strategy reads/writes the required portion from/to the array directly, i.e., it does not use sub-filing and the entire 50000×50000 array is stored as a single large file. For the sub-filing cases we show the *percentage reduction* in response time of the naive scheme. For example, in access pattern A, the sub-filing with small chunk size improved (reduced) the response time for the read operation by 85.2%. Figures 7 and 8 show the results obtained in graphical form. Note that the y-axes on the figures are *logarithmically scaled.*

Table 2: Access patterns.

	Pattern Information		
Access Pattern	Start Coordinate	End Coordinate	Total floating points
A	(0,0)	(1000,1000)	$1 * 10^6$
B	(0,0)	(4000,1000)	$4 * 10^6$
C	(0,0)	(24000,1000)	$24 * 10^6$
D	(5000,5000)	(6000,6000)	$1 * 10^6$
E	(0,0)	(50000,80)	$4 * 10^6$
F	(0,0)	(80,50000)	$4 * 10^6$
G	(0,0)	(1000,4000)	$4 * 10^6$
H	(6000,6000)	(8000,8000)	$4 * 10^6$

Table 3: Execution times and percentage gains.

	Write Operations			Read Operations		
Acc. Ptr.	Times w/o chunking	Small Chunk Gain (%)	Large Chunk Gain (%)	Times w/o chunking	Small Chunk Gain (%)	Large Chunk Gain (%)
A	2774.0	96.1	94.5	784.7	85.2	77.1
B	2805.9	83.8	84.9	810.1	43.2	55.6
C	2960.3	8.8	37.9	793.3	-240.5	-172.4
D	3321.2	96.7	95.4	798.4	84.1	79.7
E	151.7	-3525.1	-2437.6	165.2	-3229.3	-2623.9
F	138723.3	96.0	97.2	39214.1	85.9	88.5
G	11096.3	95.9	96.4	3242.9	88.3	88.6
H	5095.2	91.2	96.5	1612.9	76.6	89.9

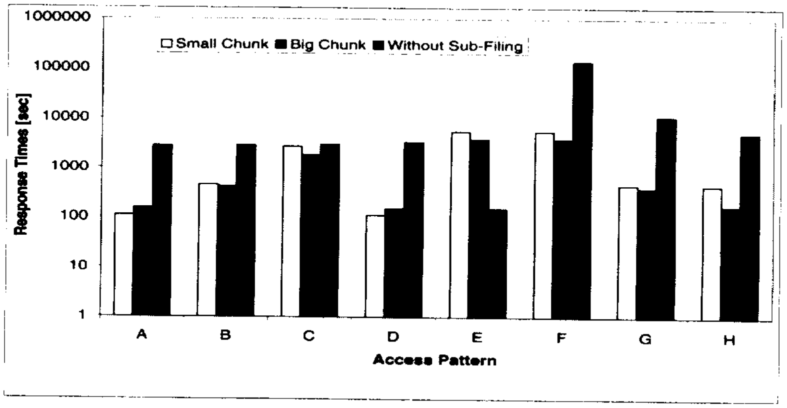


Figure 7: Execution times for **write** operations.

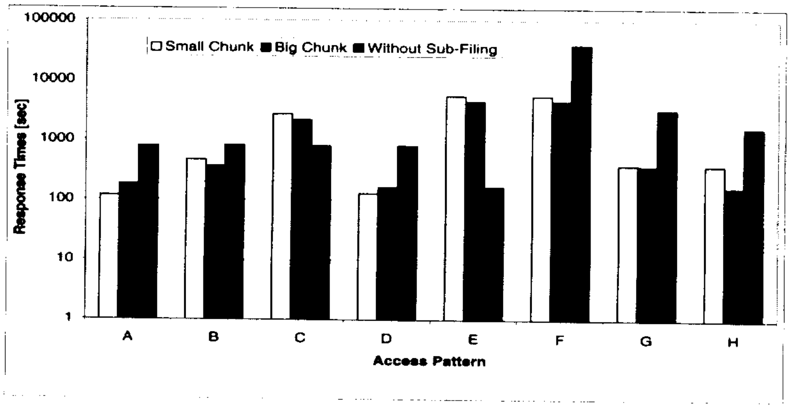


Figure 8: Execution times for **read** operations.

In the patterns A and D, where a 4 MB square chunk is accessed on the left corner and around the middle, respectively, the small chunk size outperforms the large chunk size as the latter accesses extra data elements that do not belong to the required portion. In the pattern H, on the other hand, increasing the chunk size reduces the number of I/O calls which in turn results in the best response time. In B and G, 16 MB of data are accessed in orthogonal directions. In G, since we access a sub-column portion of a row-major array, we need to issue 4000 I/O calls in the naive case. In B, the naive strategy issues only 1000 I/O calls to access the same volume of data. Consequently, the impact of sub-filing is more pronounced in G. By comparing the response times of A, B, C, and E, we note that the response times are dominated by the number of I/O calls (in the naive version) and of chunks (in the sub-filed versions–that also corresponds to I/O calls–) rather than by the volume of data accessed. Finally, in the pattern F (whose response time in the naive case was calculated using interpolation from A and G), the sub-filing strategy has the best performance of all and brings a 97.2% improvement in write calls.

In access pattern E, however, the naive strategy outperforms the sub-filing. The sub-filing strategy has two drawbacks for this access pattern. First, the naive strategy completes the whole access with a single I/O call, whereas the sub-filing strategy requires 50 calls to different sub-files to satisfy the access. Secondly, a high percentage of data read by the sub-filing is not used to satisfy the request. As a result of these two drawbacks, the naive strategy performs better than the sub-filing for this access pattern. However, we show in Section 6.1 that by chosing an appropriate sub-file size, the sub-filing strategy can perform as good as the naive strategy even for the access pattern E. Note that, HPSS allows the users to access portions of the data residing in tape. The response time of the naive solution for the access pattern E will increase dramatically for the tape architectures, where the granularity of access is a file, because the whole file should be brought to the disk from tape.

An important aspect of our library is its handling of random I/O accesses. When we compare the times for the access patterns A and D, we see that there is a 20% increase in the response time of the naive strategy for the write operation. On the other hand, the times for sub-filed versions remains the same.

Overall, the sub-filing strategy performs very well compared to the naive strategy which performs individual accesses to a large file, except for the cases where the access pattern and the storage pattern of the array match exactly. For large chunk size the average improvement for writing is 93.48%, and for reading it is 73.48%. These data show that, in average our library brings substantial amount of improvement over the naive strategy. The next section shows even in the case where the access and storage pattern match exactly, a suitable chunk shape allows our scheme to match the response time of the naive strategy.

6.1 Adaptive Chunk Size

The preliminary results show that our library can bring substantial amount of improvement over the naive case. In the access pattern E, however, the naive strategy performs better. In this section, we experiment with a different chunk size to explore the possibility of matching the performance of the naive scheme in this access pattern.

In the new experiments, the chunk size is set to 50000×80 floating points, which is similar to the access pattern E. The other parameters remain as in Section 6. The response time for write operation drops to 148.9 seconds, which is 1.85% better than the naive scheme. For read operation, the response time is 166.2 seconds, which is 0.61% worse than the naive scheme. These results show that our library can perform as good as the naive scheme even in cases, where the access pattern and storage pattern exactly match.

7 Conclusions and Future Work

We presented a portable interface to the tape-resident data. The interface makes it easier for the user to specify the tape I/O required in sequential and parallel applications. The experience gained during its design and development will, hopefully, also help in reaching a set of standard routines for accessing the tape-resident data. We are in the process of implementing the library. We completed the read, write, pre-stage, and pre-fetch routines and made some initial experiments with them. We are currently implementing different migration routines and collective I/O strategies and will later experiment with I/O-intensive applications that manipulate tape-resident data.

Acknowledgements

We would like to thank NPACI for providing the hardware and software components used in this study. This work is supported by Department of Energy under the Accelerated Strategic Computing Initiative (ASCI) Academic Strategic Alliance Program (ASAP), under subcontract no. (W-7405-ENG-48) from Lawrence Livermore National Laboratories. We would also like to thank Regan Moore and Micheal Wan of SDSC for their help with the SRB system.

References

[1] P. Cao, E. Felten, and K. Li. Application-controlled file caching policies. In *Proc. the 1994 Summer USENIX Technical Conference*, pages 171–182, June 1994.

[2] A. Choudhary, R. Bordawekar, M. Harry, R. Krishnaiyer, R. Ponnusamy, T. Singh, and R. Thakur. PASSION: parallel and scalable software for input-output. *NPAC Technical Report SCCS-636,* 1994.

[3] P. Corbett, D. Fietelson, S. Fineberg, Y. Hsu, B. Nitzberg, J. Prost, M. Snir, B. Traversat, and P. Wong. Overview of the MPI-IO parallel I/O interface, In *Proc. Third Workshop on I/O in Paral. and Distr. Sys.*, IPPS'95, Santa Barbara, CA, April 1995.

[4] J. del Rosario and A. Choudhary. High performance I/O for parallel computers: problems and prospects. *IEEE Computer,* March 1994.

[5] C. S. Ellis and D. Kotz. Prefetching in file systems for MIMD multiprocessors. In *Proc. of the 1989 Intl. Conf. on Paral. Proc.*, pages I:306–314, St. Charles, IL, August 1989.

[6] J. F. Karpovich, A. S. Grimshaw, and J. C. French. Extensible file systems (ELFS): An object-oriented approach to high performance file I/O. In *Proc. the Ninth Annual Conference on Object-Oriented Prog. Sys., Lang., and Appl.*, pp. 191–204, Oct 1994.

[7] D. Kotz. Multiprocessor file system interfaces. In *Proc. the Second Intl. Conf. on Paral. and Distr. Info. Sys.*, pages 194–201. IEEE Computer Society Press, 1993.

[8] R. H. Patterson, G. A. Gibson, and M. Satyanarayanan. A status report on research in transparent informed prefetching. *ACM Operating Systems Review*, V 27(2), pp 21–34, April 1993.

[9] R. Bordawekar, A. Choudhary, K. Kennedy, C. Koelbel, and M. Paleczny. A model and compilation strategy for out-of-core data parallel programs. *Proceedings of the ACM Symp. on Prin. and Prac. of Paral. Prg.*, pages 1–10, July 1995.

[10] S. Sarawagi: Execution reordering for tertiary memory access. *Data Engineering Bulletin* 20(3): 46–54, 1997.

[11] B. Hillyer and A. Silberschatz. Random I/O scheduling in Online Tertiary Storage Systems. In *Proc. of the 1996 ACM SIGMOD Conference*, pages 195–204, 1996.

[12] T. Johnson and E. L. Miller. Performance measurements of Tertiary Storage Devices. In *Proc. of 24rd International Conference on Very Large Data Bases*, pages 50–61, August 1998.

[13] T. Johnson and E. L. Miller. Benchmarking Tape System Performance. In *Proc. of the 15th IEEE Mass Storage Systems Symposium*, 1996.

[14] B. Kobler, J. Berbert, P. Caulk and P. C. Hariharan. Architecture and Design of Storage and Data Management for the NASA Earth Observing System Data and Information System (EOSDIS). In *Proc. of the 14th IEEE Mass Storage Systems Symposium*, pages 65–78, 1995.

[15] A. L. Drapeau and R. H. Katz. Analysis of Striped Tape Systems. In *Proc. of the 12th IEEE Mass Storage Symposium*, Monterey, CA, March 1993.

[16] L. Golubchik, R. R. Muntz, and R. W. Watson. Analysis of Striping Techniques in Robotic Storage Libraries. In *Proc. of the 14th IEEE Mass Storage Sys. Symp.*, pages 225–238, 1995.

[17] P. H. Smith and J. van Rosendale. Data and visualization corridors. *Technical Report CACR-164*, CACR, Caltech, Sept 1998.

[18] S. Toledo and F. G. Gustavson. The design and implementation of SOLAR, a portable library for scalable out-of-core linear algebra computations, In *Proc. Workshop on I/O in Paral. and Distr. Sys.*, May 1996.

[19] R. A. Coyne, H. Hulen, and R. Watson. The high performance storage system. In *Proc. Supercomputing 93*, Portland, OR, November 1993.

[20] J. del Rosario, R. Bordawekar, and A. Choudhary. Improved parallel I/O via a two-phase run-time access strategy. In *Proc. the 1993 IPPS Workshop on Input/Output in Paral. Comp. Sys.*, April 1993.

[21] A. Yu, and J. Chen. *The POSTGRES95 User Manual.* Dept. of EECS, University of California at Berkeley, July 1995.

[22] S. Baru. Storage Resource Broker (SRB) Reference Manual. Enabling Technologies Group, San Diego Supercomputer Center, La Jolla, CA, August 1997.

Performance of an MPI-IO implementation using third-party transfer

Richard Hedges, Terry Jones, John May, R. Kim Yates*
Lawrence Livermore National Laboratory
P.O. Box 808, Livermore, CA 94551
rkyates@llnl.gov
tel +1-925-423-5535

Abstract

We present a unique new implementation of MPI-IO (as defined in the recent MPI-2 message passing standard) that is easy to use, fast, efficient, and complete. Our implementation is layered over the High-Performance Storage System, using HPSS's third-party transfers and parallel I/O descriptors.

1 Introduction

The MPI-2 standard [9] includes a chapter devoted to parallel I/O functions, often called "MPI-IO." Several partial or complete implementations have appeared (e.g., ROMIO [13], PMPIO [2], Sun MPI [11], and Fujitsu's MPI-2 [7]). This paper describes a new MPI-IO implementation that has several important and unique features:

- It provides large-scale scientific applications with an easy-to-use, high-performance, portable interface to petabyte archival storage systems.
- It shows good performance and very high utilization of the underlying storage system. Our tests have demonstrated peak MPI-IO bandwidth of up to 197 MB/s for collective read operations and up to 173 MB/s for write operations, out of a maximum available bandwidth of 207 MB/s on our test platform. Even though our test platform is small (see Sec. 3.1), this compares favorably with, for example, the 150 MB/sec maximum throughput reported for PDS/PIO on the Intel TFLOP [10], both in terms of absolute performance and efficiency.

*This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under contract number W-7405-Eng-48.

- It uses a unique I/O mechanism known as *third-party transfer* in the High Performance Storage System (HPSS) archival storage system [1, 12, 14]. That is, the transfer of data from one processor to another may be arranged by a third processor, which does not participate in the actual movement of the data.
- It fully implements every MPI-IO function, including shared file pointers, error handlers, and automatic conversion between data representations.
- It is designed to work with any MPI-1 library, provided user applications are compiled with `mpio.h`. This header file defines all needed MPI-2 extensions for MPI-IO, including macros that enable MPI-IO to have access to the arguments given to MPI datatype constructors.
- It is thread-safe (as long as the underlying MPI library is also thread-safe).

Our MPI-IO implementation is a new user interface first provided with release 4.1 of HPSS. HPSS is an archival storage system that is designed to manage very large files on both disk and tape. HPSS is a joint project of IBM and several U.S. national laboratories, with a significant number of production installations.

The focus of our implementation has been to provide an efficient and scalable standard interface to the HPSS file system, providing the full functionality of the MPI-IO specification. We exploit HPSS I/O descriptors (Sec. 2.2), file striping, MPI-IO file hints, and third-party transfers to parallelize collective I/O. We shelter the user from HPSS details and constraints as much as possible. We utilize a threaded and distributed work model to minimize the latency of interactions with HPSS and to support the potential concurrency of nonblocking I/O. Our results affirm the promise of scalable performance with low overhead costs for layering MPI-IO over HPSS.

2 MPI-IO implementation

Our implementation of MPI-IO is specifically designed to run over HPSS and to take advantage of its third-party transfer capabilities. We have described this implementation elsewhere [5]; this section summarizes the design.

2.1 MPI-IO background

Two significant features of MPI-IO are *collective I/O* and access to discontiguous data chunks[1] using MPI *datatypes*. In a collective transfer, a group of processes in an application each perform a special MPI-IO read or write call, which can be implemented in such a way that information about each of the separate calls can be shared. Hence the library

[1] We use the term "chunk" to refer to a contiguous sequence of bytes in a file or memory.

can coordinate requests from multiple processes, and merge these requests to improve the locality of accesses within the file, potentially eliminating many needless disk accesses. MPI datatypes provide a mechanism for describing common static memory access patterns (e.g., slices of multidimensional arrays, records with arbitrary gaps, etc.) [3]. Using one datatype to address discontiguous regions of a file and another to address regions of a memory buffer, a single call can effect the complex data movement between them.

2.2 MPI-IO/HPSS implementation

The basic HPSS client API includes a mechanism for specifying third-party transfers using data structures called *IODs* (for "I/O descriptors"). An IOD specifies a transfer between a file and one or more client processes, and has two sides: one side describes a sequence of file chunks, and the other describes a sequence of client process memory chunks. Hence, file chunks accessed by multiple client processes can be collected into a single IOD that can be passed to HPSS through an `hpss_ReadList` or `hpss_WriteList` call.

IODs are a very flexible mechanism for describing parallel data transfers, but constructing an IOD requires quite a bit of detailed coding, and the interface is only usable for accessing HPSS files. In addition, use of the `hpss_ReadList/WriteList` interface requires an application to manage *client mover threads,* which connect application processes to remote storage devices via sockets to carry out data transfers. To offer applications a simpler and more portable programming interface, we have implemented MPI-IO on top of the HPSS IOD mechanism. The main tasks of our MPI-IO library are to manage the client mover threads and to convert MPI-IO requests into IODs.

To optimize collective I/O, it is necessary to submit I/O requests to HPSS from a single IOD. This means that collective MPI-IO calls must forward requests from many tasks to a single thread, which merges them to form an IOD. Our implementation does this using a set of *server threads* with one server thread running within each process of a parallel application. Each MPI-IO file is a single HPSS file; HPSS handles striping internally. One of our implementation's server threads is assigned to handle all collective operations on a given open file handle. However, different open file handles may be managed by different servers, so no single process bears the burden of managing all the open files. The server thread for each process is created when the application initializes MPI-IO, and a server thread only manages file handles for the parallel job that spawned it.

When an application makes a collective data access request, each process in the application forwards its part of the request to the server thread managing the request's file handle, which assembles an IOD and submits it to HPSS. HPSS carries out the data transfer to all participating processes (in parallel if possible) and then returns. Note that although *control* of the request is centralized at the server thread, the *data* itself does not flow through the server; with third-party transfer the data can move directly between storage devices and processes in parallel.

Our implementation attempts to translate collective MPI-IO requests into IODs that transfer data as efficiently as possible. Since HPSS sequentializes separate access requests for a given file handle, describing as much of a transfer as possible in a single IOD improves the opportunities for parallelism and thereby improves performance. In the ideal case requests can be merged into a transfer whose file descriptor list contains a single element (i.e., the merged transfer accesses a single contiguous chunk of the file) and whose client descriptor list uniformly stripes the transferred data across the client processes. Furthermore, optimal performance is achieved if the client side striping exactly matches the HPSS file striping specified through the HPSS class of service.

Our implementation recognizes when it can merge collective transfer requests from multiple processes into a request to access a single contiguous region of a file. If the requests cannot be merged into a single contiguous access, the IOD will require a separate file descriptor for each discontiguous chunk of the access. However, the file descriptor list of an HPSS IOD is limited to 64 descriptor elements at the time of the tests reported here. When a transfer requires an IOD with more than 64 file descriptors, our implementation automatically divides the request into multiple IODs with 64 or fewer file descriptors.

Moreover, the implementation recognizes when the collection of client memory chunks can be described compactly by a regularly "striped" pattern. Our implementation also recognizes when it can describe the client transfer mapping as uniformly distributed across the participating client processes. This simplifies the client source/sink descriptor given to HPSS by using a "striped" address. If the HPSS file is striped across the same number of devices as the number of participating processes, and if the chunk size of the client distribution matches the chunk size of the HPSS file stripe, HPSS will be able to achieve a one-to-one connection between its movers and our MPI-IO client movers for the transfer, allowing maximal concurrency.

If a transfer is not uniformly distributed across the clients, each contiguous chunk of the transfer per client requires a separate descriptor in the IOD. For these cases, we arrange the transfer description so that if there are n client processes, a maximum of n client descriptors will be used. For each client that accesses discontiguous regions of the file, however, this will result in multiple file descriptors. To summarize, irregular client distributions and/or discontiguous accesses to the file (e.g., holes in the file) will result in suboptimal performance.

3 Performance

This section reports the performance of our implementation on the platform described in Section 3.1. We begin by describing the test methods, and then we report the results of these tests. We conclude the section with a discussion of these results.

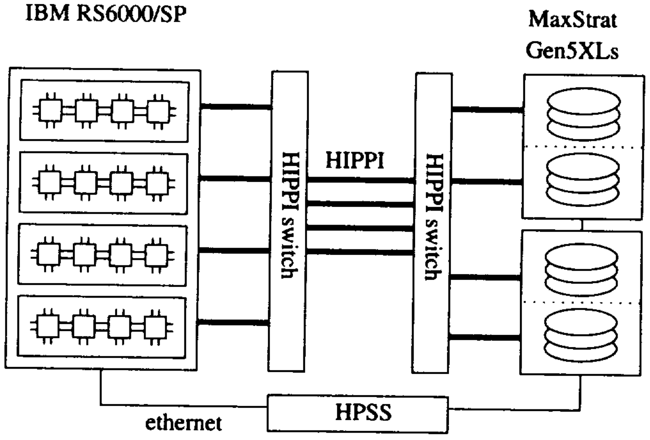


Figure 1: The test platform.

3.1 The test platform

An important feature of HPSS is its support of third-party data transfer. In this I/O model, a single process can issue a request to HPSS that will cause data to flow between the storage system and multiple processes on multiple compute nodes. (Recall that "third-party" refers to the fact that the process initiating a request need not be either the source or the destination of the data.) Using third-party transfer can simplify the management of parallel I/O transfers and reduce the number of times data is copied in a given operation. For example, a read request causes data to move between a single logical file and multiple destination processes. Third-party transfer allows this to happen with no need for intermediate buffering or shuffling of data between nodes. In this respect third-party transfer is similar to Kotz's disk-directed I/O technique [6]. When a file is striped over multiple storage devices, and different nodes are accessing different stripes, a single I/O request can initiate parallel transfer of data. Of course, the degree of parallelism will depend on how well the striping matches the data distribution on the nodes.

The main components of our hardware test platform are a parallel computer, two RAID storage devices, an interconnection network, and an HPSS server (see Fig. 1). Application code runs on a 16-processor SMP cluster consisting of four IBM RS/6000 SP 604 High Nodes that each contain four 112-MHz PowerPC 604 processors. Each node has its own HIPPI adapter card. The four HIPPI cards are connected through a crossbar switch to two Maximum Strategy Gen5 XL RAID systems. Each of these systems is configured to operate as two independent RAID devices, each with its own HIPPI connection. The maximum theoretical HIPPI bandwidth between the two RAIDs and the SMP cluster is 400 MB/second. However, the adapter cards on the SMP nodes do not stream data at full HIPPI rates, and the maximum observed bandwidth from a given card depends on the number of simultaneous connections. IBM reports that a single connection can sustain 31.5 MB/s, and that four connections through the same adapter sustain 51.8 MB/s [4]. Furthermore, MaxStrat has suggested there is a maximum theoretical transfer rate through a single HIPPI adapter channel on each RAID system of 80 MB/s [8], with less than 70 MB/s in practice. We discuss the effect of these limitations further in Section 3.4.

3.2 Test methodology

We tested the MPI-IO code using a program that let us vary four parameters:

- Stripe factor: The number of devices over which a file is striped. For these tests, we used striping factors of 1, 2, 4, and 8. As noted above, our two RAID systems behave as four independent RAID devices. For n-way striping, the HPSS stripes are evenly distributed across the four RAID logical devices. We use a striping unit (number of contiguous bytes within each stripe segment of a file) of 8 MB.
- File size: The size of the file was varied from 1 KB to 64 MB.
- Chunk size: The size of contiguous chunks that are interleaved in the file. We tested chunk sizes ranging from 1 to 16 MB in power-of-two increments.
- Number of MPI processes: We used 1, 2, 4, 8, and 16 processes; we limited the number of processes to stay within the number of CPUs available on our test system.

In addition to this test program, we evaluated MPI-IO with three more tests. The first compares native and nonnative data representation, the second compares blocking and nonblocking I/O, and the third measures performance tuning.

We configured the HPSS storage classes and used appropriate hints to `hpss_Open` so that the files we were writing would require minimum allocation and metadata overhead. We configured our test environment to enable the HPSS IPI protocol (third-party transfers), which sends data over the HIPPI network connection. Through particular choice of HPSS class of service and MPI-IO "hints," we configured HPSS to optimize transfers of large files for large-chunk accesses, with large chunks), knowing that this would be inappropriate for small files and small chunks. A production HPSS system would provide configurations appropriate for a variety of file and access needs, beyond what we used in our testing.

In the performance plots in the following section, each data point represents an average of five (in a few cases, four) test measurements. For write operations the test program overwrites a file that has already been created and written. Therefore, write timings do not include the time needed to allocate file blocks. Equivalently, we could have preallocated the file before writing.

One limitation of the experiments presented here is that the file size never exceeds 256 MB. However, HPSS does not cache file data, so caching effects should be inconsequential. It is reasonable to expect that transfer of larger files will perform similarly to the largest files reported on here; future experiments on a new testbed will be conducted to verify this.

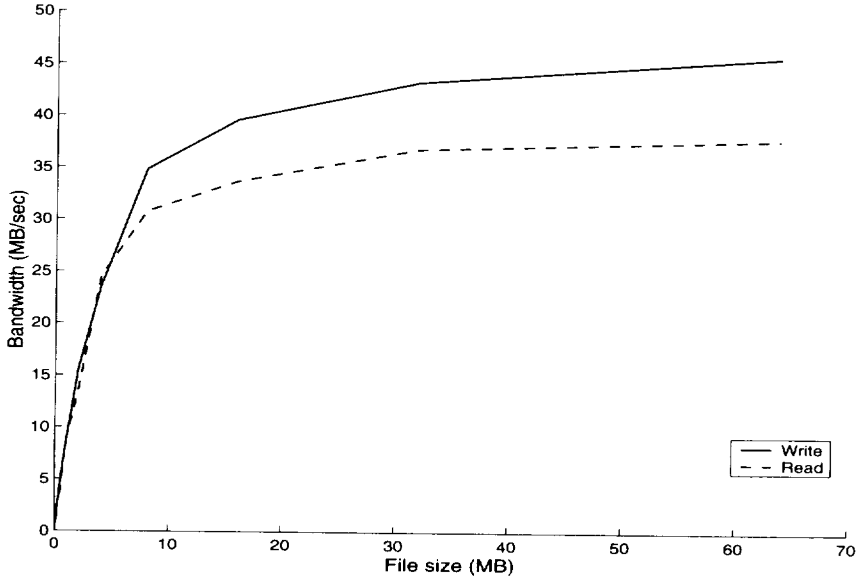


Figure 2: Baseline performance: 1 client process, stripe factor=1.

3.3 Performance results

We first show the read and write performance for independent operations with various file sizes. Figure 2 shows the baseline performance for MPI-IO: the throughput for a single process for a given file size with 1-way striping (i.e., no striping: only a single RAID is accessed). Thus, we can anticipate the maximum throughput of n processes using collective I/O with n-way striping to be n times this baseline performance.

For parallel or collective I/O, there are at least two possible ways to measure the I/O bandwidth. One is to take the average of the I/O times on the participating processes and divide this into the total amount of data moved in the operation. A second way is to take the I/O time as the interval between the earliest start time and the latest completion time of the processes. We chose the second method, which is more conservative. We inserted a barrier before each process began timing its portion of each collective I/O operation, so all the processes began each operation at about the same time.

Figure 3 shows the read and write performance for collective operations with various chunk sizes. These results are for parallel jobs with 16 MPI processes, where each process reads or writes a fixed amount of data. For these tests, we used only 8-way striping, but we vary the size of the contiguous chunk of data written by each process from 1 to 16 MB. The file size is 256 MB. At the largest reported chunk size of 16 MB there is just one chunk per process; all the other data points show transfers in which there are multiple chunks per process. These plots show the effectiveness of collective I/O, particularly as the chunk size is increased. Grouping the requests allows HPSS to handle accesses from multiple processes in parallel.

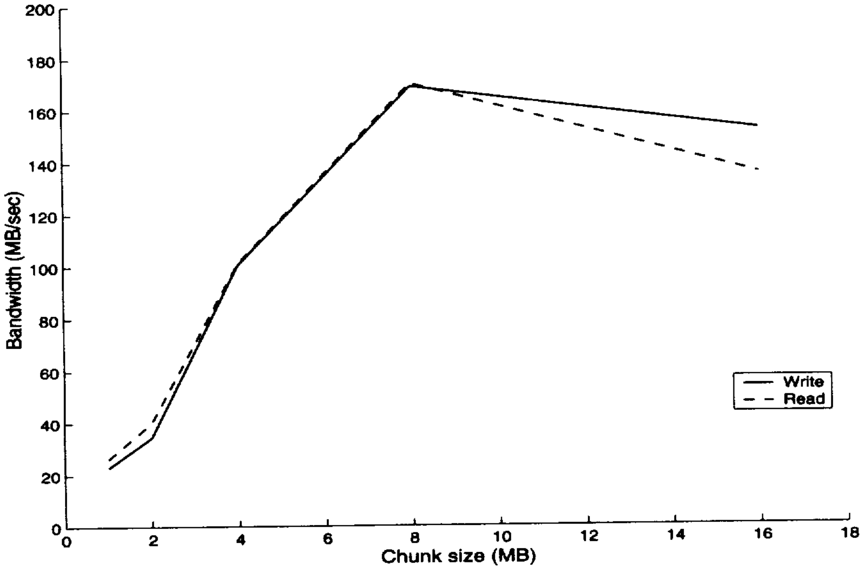


Figure 3: Collective write and read performance for varying chunk sizes (stripe factor = 8, number of processes = 16, file size = 256 MB).

In Figures 4 and 5 we show how varying the number of tasks and the stripe factor affects performance. The purpose of these tests was to explore what parameters gave the best performance on our system. For these tests, we fixed the chunk size at 16 MB and we only present data for collective operations. We vary the number of tasks from 2 to 16, and we vary the stripe factor from 2 to 8. The results show that collective I/O performance continues to improve as the stripe factor and the number of processes increases.

In a separate test we measured the performance of converting numeric formats during reading and writing. Using MPI_LONG_DOUBLE and converting between native format and MPI-2's "external32" representations, we found that the combined effect of the extra buffering and the numeric conversion proper resulted in a slowdown from 12.0 MB/s to 0.8 MB/s for writes and from 14.3 MB/s to 1.3 MB/s for reads (the chunk size was 8 MB for 64-bit native, 16 MB for 128-bit external32).

We measured the impact of using nonnative instead of native data representation. We used a chunk size of 8 or 16 MB (depending on the size of MPI_LONG_DOUBLE: in native representation, it is 64 bits; in external32 representation, it is 128 bits), one MPI process, and a stripe factor of 1. We used external32 as the nonnative representation to test. For our platform, only MPI_LONG_DOUBLE types require a data conversion from native to external32. However, conversion requires that the data be buffered, which in itself impacts the performance. We isolated the buffering costs by transferring MPI_BYTE data, which cause the data to be buffered but not converted. The difference in performance resulted in a slowdown from 12.0 MB/s to 1.7 MB/s for writes and from 14.3 MB/s to 3.0 MB/s for reads. We found the additional effect for conversion by transferring MPI_LONG_DOUBLE data. This resulted in a further slowdown to 0.8 MB/s for writes and to 1.3 MB/s for reads.

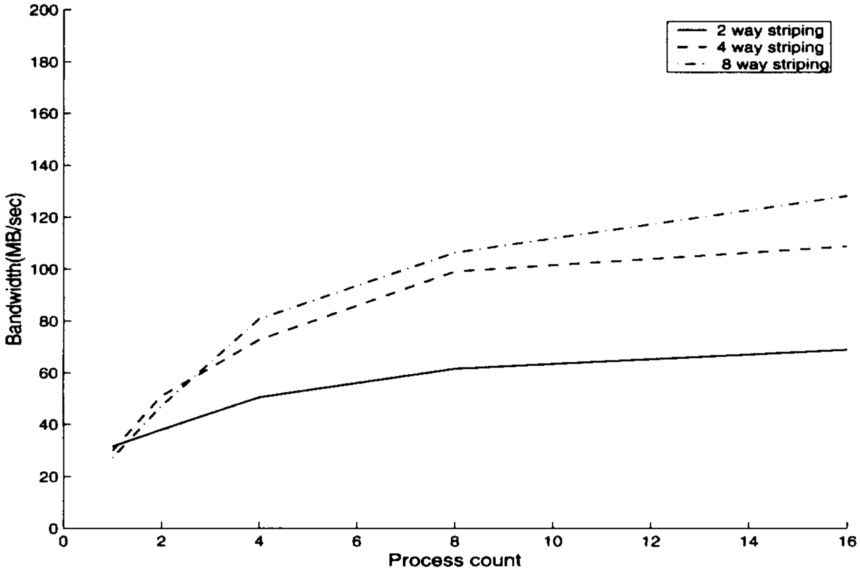


Figure 4: Collective read for varying number of processes
(stripe factor = 2/4/8, chunk size = 16 MB; file size = 16 MB / process).

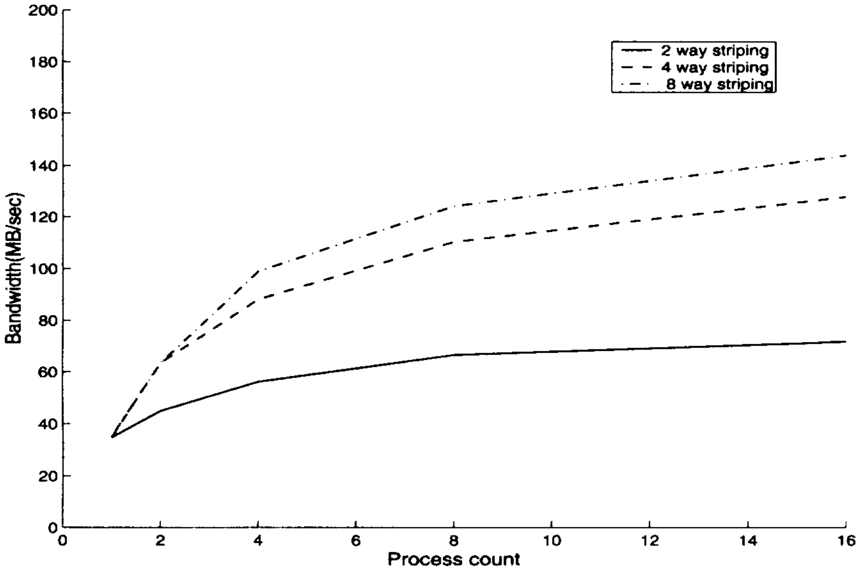


Figure 5: Collective write for varying number of processes
(stripe factor = 2/4/8, chunk size = 16 MB; file size = 16 MB / process).

We also examined the performance of blocking versus nonblocking I/O and the potential speedup of overlapping computation with I/O. We used a test that performed a floating point calculatation sequence for 16 million iterations, and then wrote the results of these calculations to a file. We constructed the calculation sequence so that the I/O and calculation times were approximately equal. We used 4 processes, a stripe factor of 4, and a chunk size of 16 MB, so the file size was 64 MB. We compared the time elapsed using sequential compute-then-write with the time elapsed when we overlap the compute and write phases using nonblocking I/O. We observed that the overlapped compute/write time (3.1 secs) was approximately 62% of the compute-then-write time (5.0 secs). We experimented with more MPI processes and stripe factors, but found that the best overlap was achieved with the four-process case. We believe this is due to contention among the threads that belong to each process when there are insufficient CPUs to assign to each active thread. That is, when there are more than 4 processes, each with multiple active threads, the processes are competing for the 4 CPUs per SMP node. When there is a single multithreaded process per node, there are 4 CPUs available for scheduling these threads.

Lastly, we constructed a test to utilize what we had learned about how to tune performance. We achieved maximum aggregate throughput for this test platform using collective I/O with 32 processes, 8-way striping and 8 MB chunks: we were able to read a 256 MB file at 197 MB/s and write it at 173 MB/s.

3.4 Analysis

The tests that vary file and chunk size show better performance for larger file sizes and chunks. This is expected since HPSS is designed for very large files and transfer sizes. It is worth examining what HPSS parameters contribute to performance variations, although we've alluded to some of these earlier.

Recall that we configured our test environment to use the IPI protocol for HPSS transfers. However, when the chunks are smaller than the HPSS stripe size, HPSS does not use the IPI protocol over HIPPI; instead, it uses TCP/IP over HIPPI. This is because HPSS defines its own blocks that are separate from, and typically larger than, the disk blocks that the storage devices uses. Transfers that do not fill a complete HPSS stripe block require special handling; to improve performance in this case we would need to collect data chunks into larger blocks. Performance peaked when the size of each chunk was 8 MB, our HPSS configuration's stripe size. For example, Fig. 3 shows that the performance of both reads and writes falls somewhat when the chunks grow from 8 to 16 MB.

Another source of performance variation is the contiguity of the data being accessed. For these tests, we deliberately avoided the use of MPI datatypes that contained holes (inaccessible regions) for the file types. That is, when the requests for all tasks are merged in a collective operation, they always form a single contiguous block of file data. Therefore, the limit on IOD length is not exceeded. Another discontiguity penalty is the metadata that is kept by the HPSS bitfile server and its limit on file fragmentation. For example, with even a single hole per filetype, the bitfile server is forced to keep two metadata records per filetype-sized chunk written, one describing the data and one describing the hole. Furthermore, the bitfile server has an upper bound (2K) on the number of fragments (metadata records) it can maintain per file. The creation and maintenance of the metadata hits the write performance severely and the read performance significantly. For the example of a single hole per filetype, writing and reading data with consecutive filetypes which are now discontigous, we see as little as only 10% of the contiguous performance.

Other HPSS configuration details that impact performance are how well the HPSS striping matches the distibution or striping of data over the client processes. This includes matching the size of a striping unit to the size of data being transferred by each client, as well as just matching number of clients and striping units. We see optimal performance when there is a 1-1 correspondence, as in the performance of 8 processes with 8-way striping which is better than the performance of 16 processes with 8-way striping.

Outside of HPSS, other limiting factors are the number of devices and connections available. Although there is a connection to each node, this is multiplexed to 4 CPUs on each node. Similarly, although there is a connection to each logical device, the IPI configuration of an 8-way stripe requires multiplexing of accesses to two stripe units per device.

On our testbed peak performance is limited by the HIPPI connections. Our measured read throughput of 197 MB/s is near the maximum aggregate performance of the HIPPI adapters on the compute nodes, which is 51.8 MB/s $\times$ 4 = 207.2 MB/s.

Although our results demonstrate scalability over the stripe factors and CPUs available for our tests, past performance is no guarantee of future returns. As the number of processors available increases, it is unlikely that all of those processors will have HIPPI interfaces available, and that would require additional data management, and possibly transferring data among nodes to maintain the collective I/O model we have implemented. HPSS is addressing this issue as well.

In summary, users will pay penalties for the flexibility of MPI-IO nonnative data representations and discontiguous accesses. The payoffs of collective I/O and concurrency of computation and I/O may ameliorate some of those penalties.

4 Future work

We are currently working on changes to improve the performance, and will be carrying out further experiments on larger testbeds and production systems.

5 Conclusions

We have examined the performance of a new MPI-IO implementation using third-party transfer and collective parallel I/O capabilities in the High Performance Storage System. Our implementation uses these capabilities to optimize file accesses from multiple processes in a parallel job. We have found the performance to be quite good, at least when reasonably large chunks are used.

Our implementation could be improved by optimizing it further to handle very finely interleaved data accesses. Other MPI-IO implementations use collective buffering to achieve this goal. Currently, our implementation does no data caching or buffering.

A further benefit of this implementation is that it adds to the list of platforms on which MPI-IO is supported efficiently, giving parallel programmers access to petabyte archives via a standard portable interface.

Acknowledgements

We thank John Allen, Rich Fischer, Dave Fox, Mark Gary, Robin Goldstone, Richard Mark, Dave Moffat, Ken Pollet, Joe Slavec, Stan Solbeck, and Steven Solbeck for helping us assemble and debug our test platform. HPSS developers Danny Cook (LANL) and Kurt Everson (IBM), along with Chris Wood of Maximum Strategy and Tom Torrico of IBM, offered several suggestions to help us maximize the performance of IPI transfers for our test platform. Jeanne Martin and Elsie Pierce contributed to earlier versions of this software.

References

[1] R. Coyne, H. Hulen, and R. Watson. The High Performance Storage System. In *Proceedings of Supercomputing '93*, November 1993.

[2] S. Fineberg, P. Wong, B. Nitzberg, and C. Kuszmaul. PMPIO—A portable implementation of MPI-IO. In *Frontiers '96, the Sixth Symposium on the Frontiers of Massively Parallel Computation*, October 1996.

[3] W. Gropp, E. Lusk, and A. Skjellum. *Using MPI: Portable Parallel Programming with the Message-Passing Interface*. MIT Press, Cambridge, Mass., 1994.

[4] IBM Corporation. IBM's MCA High Performance Parallel Interface (HIPPI). http://www.austin.ibm.com/hardware/adapters/hippi.html, February 1998.

[5] T. Jones, R. Mark, J. Martin, J. May, E. Pierce, and L. Stanberry. An MPI-IO interface to HPSS. In *Proceedings of the Fifth NASA/Goddard Conference on Mass Storage Systems*, September 1996.

[6] D. Kotz. Disk-directed I/O for MIMD multiprocessors. *ACM Transactions on Computer Systems*, 15(1):41–74, February 1997.

[7] W. Krotz-Vogel. private communication.

[8] MAXSTRAT Corporation. Gen5 XLE Product Overview. http://www.maxstrat.com/product_1.html, 1998.

[9] Message Passing Interface Forum. MPI-2: Extensions to the Message-Passing Interface. http://www.mpi-forum.org, July 1997.

[10] J. Sturtevant, M. Christon, P. Heerman, and P.-C. Chen. PDS/PIO: Lightweight libraries for collective parallel I/O. In *Proceedings of Supercomputing '98*, November 1998.

[11] Sun Microsystems Computer Company, Palo Alto, CA. *Sun MPI 3.0 Guide*, November 1997.

[12] D. Teaff, R. W. Watson, and R. A. Coyne. The architecture of the High Performance Storage System (HPSS). In *Proceedings of the Goddard Conference on Mass Storage and Technologies*, March 1995.

[13] R. Thakur, W. Gropp, and E. Lusk. Data sieving and collective I/O in ROMIO. In *Proceedings of the Seventh Symposium on the Frontiers of Massively Parallel Computation*, pages 182–189. IEEE Computer Society Press, February 1999.

[14] R. Watson and R. Coyne. The parallel I/O architecture of the High Performance Storage System (HPSS). In *IEEE Symposium on Mass Storage Systems*, September 1995.

Implementation of a Fault-Tolerant Real-Time Network-Attached Storage Device

Ashish Raniwala, Srikant Sharma, Anindya Neogi, Tzi-cker Chiueh
Experimental Computer Systems Laboratory
Computer Science Department
State University of New York at Stony Brook
Stony Brook, NY 11794-4400
{ashish, srikant, neogi, chiueh}@cs.sunysb.edu
tel +1-516-632-8436
fax +1-516-632-8334

Abstract

Phoenix is a fault-tolerant real-time network-attached storage device (NASD). Like other NASD architectures, Phoenix provides an object-based interface to data stored on network-attached disks. In addition, it features many functionalities not available in other NASDs. Phoenix supports both best-effort reads/writes and real-time disk read accesses required to support real-time multimedia applications. A standard cycle-based scan-order disk scheduling algorithm is used to provide guaranteed disk I/O performance. Phoenix ensures data availability through a RAID5-like parity mechanism, and supports service availability by maintaining the same level of quality of service (QoS) in event of single disk failures. Given a spare disk, Phoenix automatically reconstructs the failed disk data onto the spare disk while servicing on-going real-time clients without degradation in service quality. Phoenix speeds up this reconstruction process by dynamically maintaining additional redundancy beyond the RAID5-style parity on the unused space left on the disks. Phoenix attempts to improve the reliability of the disk subsystem by reducing its overall power consumption, using active prefetching techniques in conjunction with disk low-power modes. This paper describes the design and implementation details of the first Phoenix prototype.

1 Introduction

An emerging network file system architecture, called *Network-Attached Storage Device* (NASD) architecture, separates the processing of metadata such as access permission check and file directory lookup, from actual data movement between disks and client machines. Storage devices that are directly attached to the network off-load the data movement processing burden from network file servers, and thus improve the overall system scalability. This architecture contrasts with the conventional network file-systems in which there is no separation of metadata processing and data-storage. In NASD architecture, clients still send their access requests to network file servers, which after necessary checks and translations

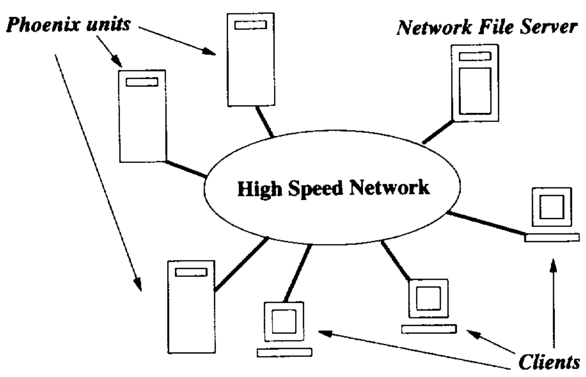


Figure 1: *Instead of attaching disk drives to the backplanes of the network file server machines, the NASD architecture uses storage devices that can be directly attached to high-speed LANs, and thus is able to exploit the aggregate bandwidth on the LAN for data transfers between disks and client machines.*

return cryptographically secure object capabilities. From this point on, clients use object capabilities to directly access the data residing on network-attached storage devices without involving network file servers. By distributing the bandwidth-intensive data transfer function across the network, the NASD architecture becomes more scalable than traditional server-attached storage architecture, both with the number of client machines as well as with the increasing link speed of the LANs. Figure 1 shows how NASD devices interact with client machines directly. *Phoenix* units constitute the storage system part of the NASD architecture. The complete NASD architecture is realized by augmenting *Phoenix* units with a file-server.

Phoenix is a Linux-based network-attached storage device built from off-the-shelf PC hardware, Fast Ethernet adapter and a set of Ultra-SCSI disks. *Phoenix* supports the following features:

- An object-based SCSI-like API.
- Bandwidth-guaranteed disk access, which is essential to real-time multimedia applications, *e.g.* MPEG streams in video-server applications.
- Both real-time disk reads and best-effort disk reads/writes.
- QoS guarantees that remain valid across single disk failures, specifically reconstruction of the contents of the failed disk onto a new disk while maintaining QoS for the existing streams.
- Utilization of unused space on disks to speed up the reconstruction process.
- Active prefetching and use of disk low-power mode to reduce disk failure probability.

This paper presents the detailed design and implementation decisions that went into the construction of the first *Phoenix* prototype. Section 2 reviews related projects in the area of NASD. Section 3 describes the data access interface which *Phoenix* provides to its client applications. Section 4 presents an overview of the design of *Phoenix* and its major architectural features. Section 5 discusses in detail the implementation of the *Phoenix* prototype under Linux. Section 6 discusses optimization features implemented in *Phoenix*. Section 7 reports performance measurements from the first *Phoenix* prototype. Finally, we conclude with a summary of major innovations in *Phoenix* and an outline of the planned work in Section 8.

2 Related Work

One of the early systems that adopted the idea of network-attached storage device is the RAID-II system built at Berkeley [1]. The focus of this work was to address the bus/memory bandwidth limitations of the disk array's host machine, by moving data directly between the network and the disks with minimal host involvement. Katz [2] discussed the concept of network and channel-based storage systems where networking and storage access are tightly integrated as a single entity. van Meter [3] provided a survey on the research areas of network-attached peripherals and the impacts of such devices on operating system design. Petal [4] uses a set of block-level storage servers that collectively manage a large set of physical disks to provide clients the abstraction of distributed virtual disks that tolerate and recover from disk, server and network failures. Frangipani [5] is a distributed file system that is built on top of Petal's distributed virtual disk service to provide scalable network file service. GFS [17] aims at providing a serverless file-system that integrates network attached storage and fibre-channel-based storage area network. This setup provides client computers full access to all storage devices on the network resulting in higher data availability.

The idea of separating high-level file system processing from low-level storage management opened up the possibility of customized optimization for file metadata processing and file data movement. The NASD project at CMU [6, 7] focused on the reduction of the file server load by providing clients an object-based access interface, which is more general and flexible than the file-based and block-based interfaces supported by file systems and disk devices, respectively. This project also addressed the important security issues in the NASD architecture. More recently, projects at U.C. Berkeley [8], CMU [9] and University of Maryland / U.C. Santa Barbara [10] all explored the idea of performing a limited form of computation inside disk drives to improve the overall system performance by reducing the data traffic between disk devices and clients. Similar ideas have been used to improve the efficiency of the disk storage system itself rather than that of the clients, for example, HP's AutoRAID system [11].

There have been several real-time storage server projects such as SUNY Stony Brook's SBVS [12], Microsoft's Tiger server [14], Starlight's StarWorks [13], and IBM's Tiger Shark parallel file system [15]. All the above systems took the more traditional network file system architecture rather than the NASD architecture. Some of these enhanced their scalability by deploying a clustered system architecture, but all data transfers had to go through the file servers.

Power management by reducing disk power consumption has been studied for mobile computers [20, 21], however, the primary goal there is to extend the battery life. Similar ideas applied to NASD can reduce heating effects by optimizing power consumption, potentially increasing the reliability of the disk subsystem [22].

Phoenix is heavily influenced by SBVS in terms of its overall architecture and internal design. It is one of the first, if not the first, NASDs that support fault-tolerant real-time object-based accesses. It provides high level of service availability as well as data availability. It also attempts to improve the reliability of the overall system by use of prefetching techniques. In addition, it supports both *server push* and *client pull* file accesses to accommodate the requirements of distributed multimedia applications.

3 Data Access Interface

The programming abstraction exposed to the clients by a *Phoenix* device is a set of logically contiguous objects whose internal structure such as disk layout is completely hidden from user applications. Clients may create, delete, access and modify objects. Each object has associated attributes like object-id, size, etc. The mapping from files and directories to objects is performed by a separate machine that serves as a network file server.

Phoenix supports both best-effort and real-time bandwidth-guaranteed disk accesses. The clients specify the data items of interest via a tuple: a unique object identifier, a block offset within the object and the number of blocks. For real-time disk accesses, an additional parameter, the bandwidth requirement in terms of 4K blocks/sec, must be specified. Clients can access data in either the *client-pull* or *server-push* mode. In the *server-push* mode, data may build up and thus exhaust buffers on the client side due to software/hardware glitches or mismatches in disk/network bandwidth scheduling granularities. To address this problem, *Phoenix* supports a general `skip` command interface with which a client application could request the *Phoenix* server not to send any data for N cycles, where N is a user-supplied parameter.

Table 1 summarizes the list of commands supported by *Phoenix*. `createsp` is used to create *special objects* at installation time and is the only one that cannot be done remotely. Executing this command is similar to creating a partition table on a fresh disk. Special objects maintain metadata information about a *Phoenix* device and the objects it contains. User objects are created and deleted with `create` and `delete`. Attributes of an object are set and read with `setattr` and `getattr` commands. All clients, real-time as well as non-real-time, use the `read` command to read data objects. The `type` parameter can have values `server-push`, `client-pull` and `best-effort`, denoting the mode of data access. For real-time clients in the client-pull mode, `read` command performs just the initial set-up for reads. To actually read the data in the object, they use the `pull` command. Data is written to an object using the `write` command. An object's size has to be declared in advance and cannot be changed dynamically. However, this restriction is not important

Command	Parameters	Return Value
`createsp/create`	attributes, perms	objid/status
`delete`	objid, perms	status
`read`	objid, offset, range, rate, perms, type	strmid/status
`write`	objid, offset, range, perms, data	status
`getattr`	objid, perms	attributes/status
`setattr`	objid, attribute name, value	status
`pull`	strmid, range, perms	data/status
`skip`	strmid, cycles, perms	status
`getdeviceinfo`	perms	deviceInfo
`shutdown/bootup`	perms	status

Table 1: *The set of commands supported by* Phoenix, *their arguments and return values.*

because a conventional file is organized as a chain of objects with new objects added on file growth. Commands `shutdown` and `bootup` perform the remote shutdown and bootup of a *Phoenix* system.

4 Phoenix System Architecture

4.1 Basic Design

In *Phoenix* each storage object is striped across a software-controlled disk array in a sequentially interleaved fashion, with a RAID-5 style of parity to protect data against single disk failures. Two *special objects* keep the metadata about a *Phoenix* device and individual objects on the device. The *DeviceInfo* object contains the device type, capacity, free space, block size, permissions, the starting location and size of the *ObjectList* object, etc. The *ObjectList* object contains a list of attributes for each object striped on the disk array including its size, starting offset, permissions, etc. *Phoenix* uses a fixed stripe unit size of 4 KBytes, which is independent of objects and the requested access rates to them.

Phoenix uses a cycle-based disk scheduling algorithm to provide disk bandwidth guarantees. In each I/O cycle, *Phoenix* retrieves from disks an amount of data for each real-time stream corresponding to its bandwidth reservation. Within an I/O cycle, initially real-time disk access requests are serviced in the scan order based on blocks accessed from the disks, and then the best-effort access requests are served in a partial scan order (explained in section 5.2). This ordering reduces the disk head seek overhead, simplifies the scheduling of non-real-time accesses and also makes it possible to perform I/O cycle utilization measurements required for admission control. A fixed percentage of the I/O cycle is reserved for best-effort traffic to guarantee that best-effort requests never starve. An explicit dynamic measurement-based statistical admission control, similar to the one used in SBVS, ensures that *Phoenix* can admit as many requests as possible while meeting the QoS guarantees to its clients. To maintain the continuity of data flow, *Phoenix* employs a double buffering scheme where the disk subsystem fills up one set of buffers with data while the other set is being emptied out onto the network.

4.2 Failure-Tolerant Real-Time Disk Service

An innovative feature of *Phoenix* is its ability to maintain the QoS guarantee to real-time clients across single disk failures. In contrast, conventional disk arrays put more emphasis on data availability and render the disks' service unavailable during the failure recovery period. *Phoenix*, on the other hand, continues to provide guaranteed disk bandwidth to real-time applications by treating reconstruction-related disk accesses as best-effort traffic.

Disk failures are detected by associating a timeout with each request issued to the disk array. On failure detection, *Phoenix* switches to *failure* mode. In failure mode, the reads which should be served by the failed disk are redirected to the corresponding block on the parity disk. After reading a complete stripe group, *Phoenix* re-builds the block on the failed disk through parity. The parity computation leads to an increase in the I/O cycle time. However, the parity computation is partially overlapped with the disk accesses to improve the performance.

4.3 Failure Recovery

While in failure mode, *Phoenix* sends periodic SCSI *inquiry* commands to detect the existence of spare disk. On successful detection, a switch is made to the *recovery* mode. To shorten the recovery phase, *Phoenix* denies all best-effort access requests in this mode. During the recovery period, a dummy stream called *reconstruction stream* is started to reconstruct the data of the failed disk onto the spare disk by making use of parity. Disk I/Os associated with the service of the client real-time streams also computes the portions of data on the failed disk using parity. The question is whether to write such computed data back to the spare disk (called the *piggyback* approach) or not (called the *non-piggyback* approach). Experiments with both the approaches were conducted and finally the *piggyback* approach was chosen for implementation [19]. The *piggyback* approach reuses the efforts involved in servicing real-time streams to do the disk reconstruction.

5 Implementation

5.1 Hardware Components

The first *Phoenix* prototype has been implemented on a PentiumPro 200-MHz PC with 128 MBytes of physical memory. The prototype has a 1-GByte IDE disk to hold the *Phoenix* kernel, swap space, and basic utilities programs. In addition, it is connected to an array of five Seagate ST34371W 4-GByte Ultra Wide SCSI disks physically mounted within an external disk case via an Adaptec 2940 Ultra-Wide SCSI adapter sitting on a 33-MHz PCI bus. Data is striped across the SCSI disk array, with a striping unit of 4 KBytes (which is also the minimum retrieval size for all disk accesses) and one of the disks designated as the parity disk. The prototype is connected to a switched Fast Ethernet through an Intel PRO/100+ PCI adapter.

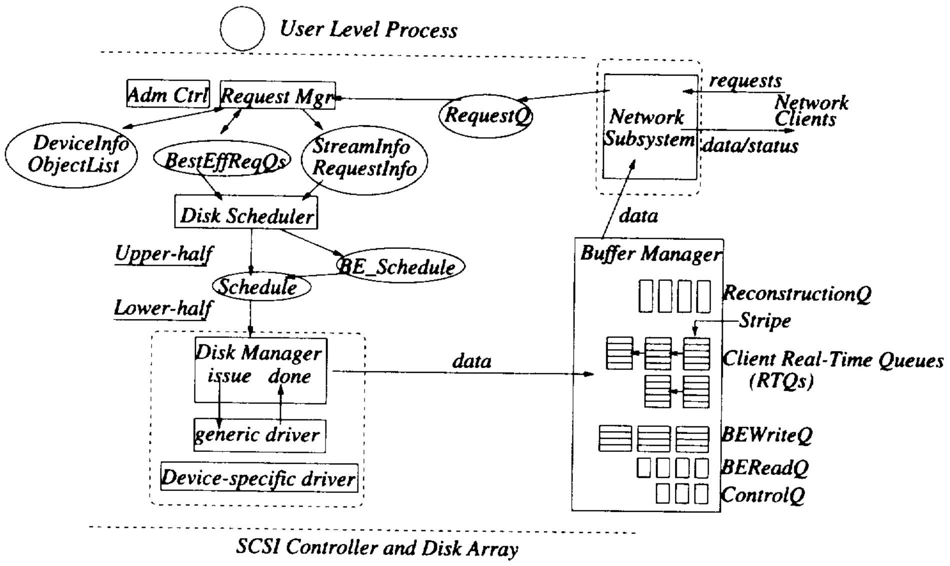


Figure 2: *Software Architecture of Phoenix depicting various modules, data structures and interactions between them. The arrows depict the basic data flow.*

5.2 Software Architecture

The *Phoenix* kernel is based on Linux 2.2.12. The interaction between the *Phoenix* subsystem with the Linux kernel is limited to memory management for allocation of buffers, scheduling of timers, kernel socket code for the network subsystem, and the generic SCSI controller driver for sending SCSI commands to the disk. The device-specific portion of the SCSI driver was left untouched. Because of the modular software architecture, it is expected that porting *Phoenix* to other hardware/OS platforms and Linux versions should be relatively straightforward.

The software architecture of the *Phoenix* kernel is shown in figure 2. *Phoenix* kernel code is activated by a startup user-level program that makes a system call with some configuration parameters. From this point onwards, *Phoenix* remains in the kernel mode. The kernel consists of a timer-driven upper-half which comprises the disk scheduler, the request manager and the admission controller, and the disk interrupt-driven lower-half comprising the low-level disk manager. The buffer manager supports other subsystems. The network subsystem is a timer-driven module that is invoked once every network cycle [16] to send data to the clients, and to accept new requests from the clients.

To implement cycle-by-cycle disk scheduling, the upper-half is invoked once every I/O cycle to prepare the disk schedule for every disk and initiate the lower-half to start disk request processing. Thereafter, the lower-half issues the next disk request from the SCSI callback function once the previous request finishes, until the access requests for all the disks are completed.

Since the disk scheduler can not determine in advance the number of non-real-time requests to be scheduled for service, it prepares a separate schedule, called best-effort-schedule, for these requests. Once the lower-half is done with the real-time requests for an I/O cycle, it invokes the best-effort scheduler to dynamically schedule requests from the best-effort schedule. Best-effort access requests that remain unserviced at the end of the current I/O cycle are processed in later I/O cycles. New best-effort requests are added to the best-effort schedule in scan order after these left-over requests. The ordering of left-over requests is not altered to avoid their starvation. Thus, the set of requests arriving within an I/O cycle are put in scan order and those arriving across I/O cycles are put in FIFO order. This ordering is termed as *partial scan order.*

5.2.1 Data Structures

`StreamInfo` and `RequestInfo` lists maintain the information regarding the on-going real-time streams, and best-effort read and write requests. The `DeviceInfo` and the `ObjectList` structures are in-memory copy of the *DeviceInfo* and *ObjectList* stored on the disks (refer to section 4.1). `RequestQ` is used by the network subsystem to queue new client requests. The corresponding `ReplyQ` is the `ControlQ` of the buffer manager. After processing the best effort requests, the request manager queues them up in the `BestEffReqQs` which are then picked up by the disk scheduler. `Schedule` is the schedule prepared by the upper-half to be used by the lower-half in the next disk I/O cycle. `BE_Schedule`, also prepared by the upper-half, is used to hold the best-effort requests scheduled to be sent to the disks. The various queues maintained by the buffer manager are discussed in section 5.2.5. The network subsystem, the upper-half and the lower-half all are executed from bottom halves of the timer or disk interrupt service routines. Since no two bottom-halves can execute concurrently, the consistency of any shared data structure among bottom-half processing modules is guaranteed.

5.2.2 Admission Control

The admission control module implements a measurement-based statistical admission control algorithm to determine whether to admit a new real-time stream. The module exports `admit_stream()` function which uses the following equation to predict the total service time after admitting the new $(N+1th)$ stream based on the past service time measurements for the on-going N streams.

$$\begin{aligned}\texttt{Pred\_Service}_{N+1} &= \texttt{Current\_Service}_N + \texttt{Std\_dev}_N + \texttt{Increase\_Seek\_Time} + \\ &\quad (\texttt{Current\_Service}_N * (\texttt{Requested\_Rate}/\texttt{Total\_Rate}_N))\end{aligned}$$

$\texttt{Std\_dev}_N$ is the standard deviation from the current service time (averaged over past few I/O cycles), $\texttt{Current\_Service}_N$, for N streams. `Increase_Seek_Time` is the increase in the seek time per I/O cycle if the new stream is admitted. $\texttt{Total\_Rate}_N$ is the summation of rates of all the on-going real-time streams. If the predicted service time for $N+1$ streams is less than the I/O cycle share reserved for real-time streams, then the new stream is admitted, otherwise it is rejected. While *Phoenix* is operating in the failure or recovery mode, the admission control simply rejects all new stream/best-effort read/write requests.

5.2.3 Request Manager

This module exports the function process_requests(), which takes client requests from the RequestQ and processes them based on their types. Table 1 lists all the possible requests. All requests are first validated. createsp, create, delete and setattr involve updating the DeviceInfo and ObjectList data structures. getattr and getdeviceinfo just access these structures for sending information to the clients. Best-effort read and write requests are processed and queued in the BestEffReqQs list and an entry is made in the RequestInfo list. Client Write requests are broken into stripe group writes and for each such stripe group write, OLD_DATA_READ and OLD_PARITY_READ requests are put into the BestEffReqQs. These reads in turn trigger the actual writes. The parity block is read to keep it updated with new block writes.

Every real-time read request is validated by the admission control and then an entry is inserted into the StreamInfo list. pull and skip simply update a counter in the StreamInfo structure, which is periodically checked by the network subsystem to decide whether to send data to the client or not. shutdown closes down *Phoenix* by writing the in-core copy of the DeviceInfo and ObjectList structures to the disks, cleaning up all required data structures. Bootup initializes *Phoenix* by reading the disk-copy of these data structures into memory.

5.2.4 Disk Scheduler

The disk scheduler exports the function update_schedule() which prepares the next I/O cycle's disk schedule to be served by the lower-half. The disk scheduler first puts the real-time requests in the Schedule data structure. It reads StreamInfo structure to retrieve the rate and current pointer information for on-going real-time streams. For each real-time stream, the disk scheduler schedules 2*data_rate-unconsumed_buf_size amount of reads (rounded off to complete parity groups). To reduce disk seek overhead, the disk scheduler tries to use a retrieval size as close to the maximum retrieval size (64 KBytes) as possible.

Unlike real-time requests, the exact number of non-real-time requests which will make the I/O cycle utilization optimal can not be pre-determined. To handle this, the disk scheduler fills up enough non-real-time requests in a separate BE_schedule in a partial scan order. When all real-time requests scheduled in an I/O cycle are completed, the disk scheduler invokes get_next_BE_request() to get the next best-effort request from the BE_schedule into the Schedule. This allows the lower-half to get as many non-real-time requests as it can serve and thus keep the I/O cycle optimally utilized.

In failure mode, the disk scheduler shifts the requests which should be served by the failed disk to the parity disk. It also puts inquiry commands in the disk schedule to probe pre-configured I/O locations to detect if a spare disk is available. On detection of a spare disk, the system switches to *recovery* mode. In the recovery mode, the disk scheduler schedules reads associated with data reconstruction, which in turn trigger reconstruction-related writes. The ranges of the disk blocks which the current streams are accessing are stored in the AutoReconstRanges and are termed as active blocks. The reconstruction of active blocks is piggybacked with the continual service of real-time streams. The disk scheduler schedules stripe-group reads for reconstruction of inactive blocks (not accessed by the existing real-time clients). Once the reconstruction of such inactive blocks is over and that of active blocks is not complete, the reconstruction stream starts with the reconstruction of the active blocks.

5.2.5 Buffer Manager

Figure 2 shows various queues maintained by the buffer manager. Each on-going stream has an RTQ structure, which is allocated using allocate_rtq(). This points to the linked list of the data buffers (shown in figure). Each node in the list can store a complete parity group, i.e., num_disks*retrieval_size bytes. These buffers are allocated using bmgr_get_buffer(). RTQ[0] is a special stream used to store the data read by the reconstruction stream. Write requests are allocated write buffers linked in the BEWriteQ using bmgr_allocate_write_buffer(). Each such buffer has 4 parts - OLD_DATA, NEW_DATA, OLD_PARITY and NEW_PARITY. Best-effort reads are allocated a data buffer linked in the BEReadQ. ReconstructionQ stores the reconstruction data to be written to the spare disk. Both the best-effort read buffers and the reconstruction buffers are allocated using bmgr_get_buffer() routine. ControlQ stores the control messages for the clients.

5.2.6 Disk Manager and Generic SCSI Driver

The disk manager exports the dmgr_start() function, which is called by the upper-half to trigger the next disk I/O cycle. This function issues the first set of requests to all the disks and then immediately returns. The completion of these requests is indicated by a call to scsi_done() which is the main part of this module. scsi_done() issues the next request and processes the reply received from the disk.

Issuing a disk request involves getting the next request from the disk schedule based on diskid and slotid of the reply, constructing the next command to be sent to the disk, and then sending out the actual disk request (with an associated timeout to detect possible disk failures). **Processing a reply** involves checking the reply for error_codes, switching to failure mode in case of disk failure detection, and then performing further reply processing termed as *post-processing*. Switching to failure mode involves schedule recomputation for the current I/O cycle where the real-time reads scheduled for failed disk are shifted to the parity disk. The next request is issued just before the *post-processing* stage so as to overlap the post-processing (e.g. parity computation, etc.) with the next disk access. The post-processing constitutes queueing up the required buffers with the buffer manager, updating relevant data structures, performing parity computations if required and queueing up further disk requests.

A subtle change was made to the disk request issue scheme described above based on reconstruction experiments. The request processing rate is not uniform across the disks. Thus, some disk might have processed more reconstruction-read requests than other ones, leading to accumulation of buffers. To avoid this buffer accumulation, *Phoenix* tries to balance the number of reads across the disks. In effect, in every I/O cycle, the first set of requests to the disks is sent out in the reverse order of the length of the disks' pending requests queues.

Another technique used to ensure uniform disk service progress is to slow down the *leading disk*. The *leading disk* is defined as the disk which has processed maximum number of requests from its schedule. This *leading disk* keeps changing dynamically as an I/O cycle proceeds. No more requests are sent to this disk as long as it remains the *leading disk*. Since 4 disks are sufficient to optimally utilize an UltraWide SCSI bus, not scheduling the fifth disk does not have significant impact on performance. In a typical setting, the number of *leading disks* can be determined based on supported SCSI bus bandwidth, the total number of disks and the difference between the processing rates of the disks. The main concept is to avoid request scheduling for the disks which are going faster than the other disks without affecting the overall performance.

5.2.7 Network Subsystem

The function `do_net_io_cycle()` of the network subsystem is invoked once every network cycle. It looks at the `StreamInfo` and `RequestInfo` structures, dequeues data from the buffer manager and sends it to the clients. It also sends them the new control messages queued in the `ControlQ` of the buffer manager. It processes the new client requests and puts them in the `RequestQ`. This module works by making socket-layer system calls from within the kernel to send out UDP packets over the network. This subsystem is relatively independent of the rest of the system and can be fairly easily replaced by other real-time network subsystems, e.g., Rether [16].

6 Optimization Features

6.1 Dynamic Replication to Reduce Reconstruction Time

To reduce the data reconstruction time, *Phoenix* employs a *dynamic replication* scheme that uses unutilized storage space in the disk array to *mirror* a part or all of the utilized portion of the disk array. The extent of the disk array's utilized portion that gets replicated depends on the size of the unused space. Here, a disk array consists of three parts *viz.* utilized & mirrored (UTM), utilized & parity-protected (UTP), and unutilized & mirrored (UUM) (figure 3). Both the UTM and UUM are reconstructed via $1 : 1$ reads and writes, whereas the UTP portion is reconstructed via $(N - 1) : 1$ reads and writes, where N is the parity group size, plus a parity computation. The mirroring scheme chosen, called *Declustered Replication*, distributes the replication for each disk across all other disks to increase read parallelism. To minimize seek overheads, replication unit is chosen to be the disk's maximum retrieval size (64 KBytes).

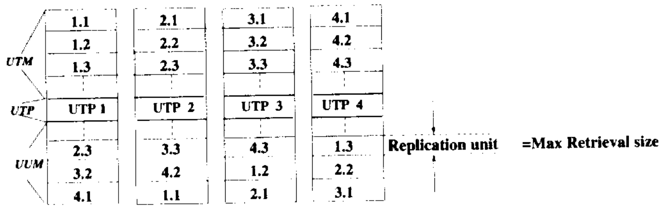


Figure 3: *Dynamic Declustered Replication. Each disk in the array is partitioned into 3 parts: UTM, which is reconstructed from the mirrored copy replicated across other disks, UTP, which is reconstructed from parity, and UUM, which stores mirror copies of other disks' UTM portions. UTM and UTP together represent the part of a disk that is being utilized.*

To implement dynamic replication, additional replication writes are scheduled for client writes to maintain the replication consistency. During the reconstruction phase, the disk scheduler tries to use the existing mirror copy to reconstruct the data. `ReconstructionQ` is used to temporarily store this data. The reconstruction of UTM and UTP is done in parallel to ensure optimal performance. Also, excessive UTM reads in short time may lead to write buffer accumulation and are therefore thwarted appropriately. Reconstruction related measurements on *Phoenix* prototype indicate significant performance gains achieved by the use of this approach. The benefits are expected to increase further as the number of disks in the parity group increases.

6.2 Active Prefetching to Lower the Power Consumption

To reduce the probability of disk failures due to overheating [22], *Phoenix* tries to reduce the overall power consumption of the disk subsystem. *Phoenix* employs an *active prefetching* technique by exploiting real-time applications' regular data access patterns. Rather than leaving the unused bandwidth in each I/O cycle wasted, *Phoenix* uses the spare bandwidth to prefetch data for each real-time stream, in order to skip some I/O cycles every once in a while. In these skipped I/O cycles, *Phoenix* puts the disks in the *low-power mode* and thus lowers the power consumption of the disk array. Switching between *low-power* and *normal* operating modes involves only electronic components rather than mechanical parts [18]. Therefore, mode switching power consumption is negligible as compared to power saving achieved. Consequently, active prefetching can ensure that the power consumption of a *Phoenix* device is proportional to the number of active streams being serviced at that time.

As *Phoenix* switches to the low-power mode, the upper-half no longer remains timer-driven. When the disk manager is done with its I/O cycle, it invokes the upper-half directly. The scheduler now schedules read requests for all the streams making sure that streams are prefetched fairly. When enough data is accumulated, the disk manager puts the disks in *low-power mode* and does not invoke the upper-half. The network subsystem keeps consuming the data and when the data level falls below a certain threshold, the network subsystem invokes the upper-half.

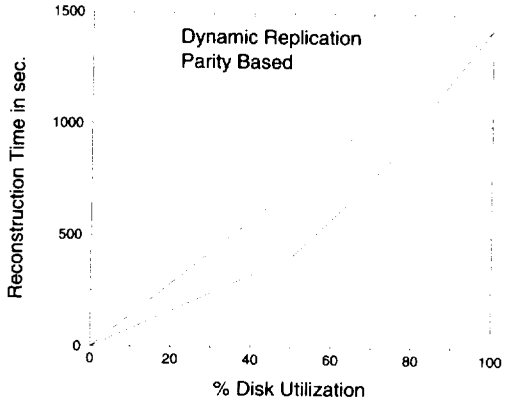


Figure 4: Variation of reconstruction time with increasing disk utilization.

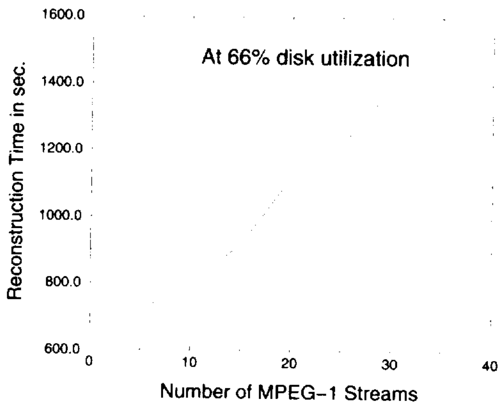


Figure 5: Variation of reconstruction time with increasing number of active streams

7 Performance Measurements

The throughput of the system was measured in terms of the number of MPEG-1 streams it can support. *Phoenix* supports a maximum of 52 streams in normal mode, 42 streams in failure mode and 36 streams in reconstruction mode. Thus, *Phoenix* services a maximum of 36 guaranteed RT streams across failures/reconstruction and an additional 16 guaranteed RT streams but not across failures.

Figure 4 shows the variation of raw reconstruction time (no streams in the system) as the disk utilization increases. A significant gain in reconstruction performance suggests use of *dynamic replication*. Reconstruction up to 50% disk utilization is totally based on replication and then onwards, reconstruction uses parity as well as mirrored data.

Figure 5 shows the variation of reconstruction time as the number of MPEG-1 streams increases. The length of these streams is constant and the streams are uniformly spread across the disks. The disk utilization is kept at 66% (equal UTM and UTP portions). When there are no client streams, the reconstruction is solely due to the reconstruction stream. The reconstruction time is minimum in this case. The reconstruction time increases with the number of real-time clients because of seek and request processing overheads.

Simulations were done to gauge the potential reduction in power consumption, and thus the increased reliability that can be achieved using active prefetching. The fraction of total time available for keeping the disks in low-power mode is shown in figure 6. As the number of streams reduces, the power consumption can also be reduced almost linearly.

8 Conclusions

This paper described in detail the design and implementation of a Linux-based network attached storage device, which exports an object based API, supports real-time reads and

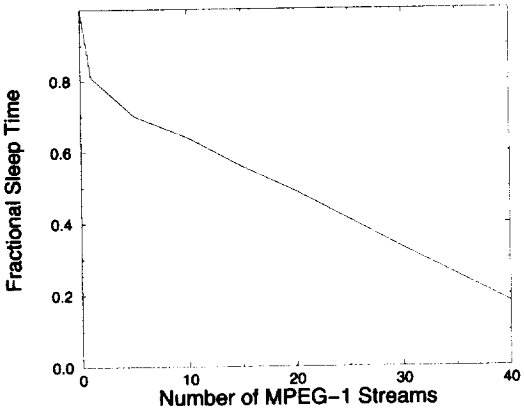


Figure 6: *Variation of* fractional sleep time *with increasing number of streams.*

best-effort reads/writes, provides uninterrupted real-time disk service in the event of a single disk failure, performs on-line disk reconstruction while using the active real-time streams, exploits full disk bandwidth and disk space all the time to speed up failed disk reconstruction, and increases the reliability of the disk subsystem by reducing its power consumption. Extensive measurements on the first *Phoenix* prototype were made to validate the design decisions (described in [19]). In future, the prototype will also be integrated with a real-time network subsystem [16] and a file-system.

References

[1] Drapeau A.L.; et. al., "RAID-II: a high-bandwidth network file server," Proc. of the 21st Annual Intl. Symp. on Computer Architecture, p. 234-44, Chicago, IL, 1994.

[2] Katz R.H., "High-performance network and channel-based storage," Proceedings of the IEEE, vol.80, no.8, p. 1238-61, Aug. 1992.

[3] Van Meter R., "A brief survey of current work on network attached peripherals," Operating Systems Review, vol.30, no.1, p. 63-70, Jan. 1996.

[4] Lee E.K.; Thekkath C.A., "Petal: distributed virtual disks," 7th International Conference on Architectural Support for Programming Languages and Operating Systems, p. 63-70, Cambridge, MA, Oct. 1996.

[5] Thekkath C.A.; Mann T.; Lee E.K., "Frangipani: a scalable distributed file system," 16th ACM Symposium on Operating Systems Principles, p. 224-37, Saint Malo, France, Oct. 1997.

[6] Gibson G.A.; et. al., "A cost-effective, high-bandwidth storage architecture," Eighth International Conference on Architectural Support for Programming Languages and Operating Systems, p. 92-103, San Jose, CA, Oct. 1998.

[7] Gibson G.A.; et. al., "File server scaling with network-attached secure disks," 1997 ACM International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS 97), p. 272-84, Seattle, WA, June 1997.

[8] Keeton K.; Patterson D.A.; Hellerstein J.M., "A case for intelligent disks (IDISKs)," SIGMOD Record, vol.27, no.3, p. 42-52, Sept. 1998.

[9] Riedel E.; Gibson G.; Faloutsos C., "Active storage for large-scale data mining and multimedia," Proceedings of the 24th VLDB Conference, New York, NY., Aug. 1998.

[10] Acharya A.; Uysal M.; Saltz J., "Active disks: programming model, algorithms and evaluation," Eighth International Conference on Architectural Support for Programming Languages and Operating Systems, p. 81-91, San Jose, CA, Oct. 1998.

[11] Wilkes J.; Golding R.; Staelin C.; Sullivan T., " The HP AutoRAID hierarchical storage system," ACM Trans. on Computer Systems, vol.14, no.1, p. 108-36, Feb. 1996.

[12] Chiueh T.; Vernick M.; Venkatramani C., "Integration of Real-Time I/O and Network Support in Stony Brook Video Server," IEEE Network Magazine, April 1999.

[13] Tobagi F.A.; Pang J.; Baird R.; Gang M., "Streaming RAID - a disk array management system for video files," Proceedings of First ACM International Conference on Multimedia, p. 393-400, Anaheim, CA, Aug. 1993.

[14] Bolosky W.J.; Fitzgerald R.P.; Douceur J.R., "Distributed schedule management in the Tiger video fileserver," 16th ACM Symposium on Operating Systems Principles, p. 212-23, Saint Malo, France, Oct. 1997.

[15] Haskin R.L., " Tiger Shark-a scalable file system for multimedia," IBM Journal of Research and Development, vol.42, no.2, p. 185-97, March 1998.

[16] Venkatramani C.; Chiueh T., "Design, Implementation, and Evaluation of A Software-Driven Real-Time Ethernet Protocol," ACM SIGCOMM, 1995.

[17] Perslan K. W.; et. al., "A 64 Bit, Shared Disk File System for Linux," IEEE Mass Storage Systems Symposium, March 15-18, 1999, San Diego, California.

[18] "IBM Travelstar 6GT," http://www.storage.ibm.com/hardsoft/diskdrdl/prod/6gtprod.htm.

[19] Neogi A.; Raniwala A.; Chiueh T., "Phoenix: A low-power fault-tolerant network-attached storage device," ACM Multimedia, 1999.

[20] Li K.; Kumpf; Horton P.; Anderson T., "A quantative analysis of disk drive power management in portable computers," Proc. of the 1994 Winter USENIX, p. 279–291.

[21] Douglis F.; Krishnan P.; Marsh B., "Thwarting the power-hungry disk," Proc. of the 1994 Winter USENIX Conference, Jan. 1994.

[22] Herbst G., "IBM's drive temperature indicator processor (drive-TIP) helps ensure high drive reliability,"
http://www.storage.ibm.com/hardsoft/diskdrdl/technolo/drivetemp/drivetemp.htm

Access Coordination Of Tertiary Storage For High Energy Physics Applications *

Luis M. Bernardo, Arie Shoshani, Alexander Sim, Henrik Nordberg
Scientific Data Management Research Group
NERSC Division
Lawrence Berkeley National Laboratory
Berkeley, CA 94720
{LMBernardo, AShoshani, ASim, HNordberg}@lbl.gov
tel +1-510-486-5171
fax +1-510-486-4004

Abstract

We describe a real implementation of a software component that manages caching of files from a tertiary storage management system to a large disk cache developed for use in the area of High Energy Physics (HEP) analysis. This component, called the Cache Manager, is a part of a Storage Access Coordination System (STACS), and is responsible for the interaction with a mass storage system that manages the robotic tapes (we used HPSS). The Cache Manager performs several functions, including managing the queue of file transfer requests, reordering requests to minimize tape mounts, monitoring the progress of file transfers, handling transient failures of the mass storage system and the network, measuring end-to-end file transfer performance, and providing time estimates for multi-file requests. These functions are described in detail, and illustrated with performance graphs of real-time runs of the system.

1 Introduction

Like so many other scientific disciplines, HEP experiments produce huge amounts of data that, given the usual budget constraints, need to be stored in robotic tape systems. For instance, the STAR experiment at Brookhaven National Laboratory that will start collecting data by mid 2000, will generate 300 TB of data over the course of three years. Storing such amounts of data in disks is certainly unreasonable and also a waste of financial resources since most of the data will not be used often, yet they need to be archived. In practice all the data will be stored in tapes and the amount of available disk space will amount to a few percent of the total space needed to store all the data. Given the fact that retrieval of data from tapes is much slower than from disk, the need for smart cache management

*This work was supported by the Office of Energy Research, Office of Computational and Technology Research, Division of Mathematical, Information, and Computational Sciences, of the U.S. Department of Energy under Contract No. DE-AC03-76SF00098.

systems, that coordinate both the retrieval of data from tapes and the use of the restricted disk cache, is real [3, 2, 1]. With this goal in mind we developed STACS (Storage Access Coordination System) [4] to be used by the STAR experiment. STACS was designed to take advantage of the fact that the particle collisions, recorded by the STAR measuring devices, are independent of each other, and therefore the processing of each collision's data can be done in any order. This provides the ability to choose the order of caching of data from tape to disk cache, so as to optimize the use of the cache by multiple users. In addition, since we know ahead of time all the files needed for processing for all the users currently in the system, we can order the scheduling of file transfers to minimize the number of tape mounts.

This paper is organized as follows. In section 2, we start by briefly describing the application domain of High Energy Physics and how the particular needs of that domain influenced the design of STACS. We briefly discuss the architecture of STACS, and describe the process of executing queries. In section 3, we describe in detail the component responsible for interacting with the system that manages the tapes (we used HPSS), called the Cache Manager. In this paper, we emphasize many of its features, including the support of a request queue, the reordering of file transfers to minimize tape mount, and the handling of errors and system failures. We conclude in section 4.

2 The STACS Architecture

We describe in this section the components of STACS, and the reasons for the modular architecture of the system. First, we need to describe briefly the application domain, the kind of queries applied to the system, and what is expected from the application's point of view.

2.1 HEP Application Domain

In the HEP STAR experiment, gold nuclei are collided against each other inside an accelerator and the results of such collisions are recorded by a very complex set of measuring devices. Each collision is called an event and the data associated with each event is in the order of 1-10 MB. It is expected that the experiment will generate 10^8 such events over 3 years. The raw data recorded by the measuring devices are recorded on tapes. They are organized in files, each about 1 GB in size. The data then undergo a "reconstruction" phase where each event is analyzed to determine what particles were produced and to extract summary properties for each event (such as the total energy of the event, momentum, and number of particles of each type). The number of summary elements extracted per event can be quite large (100-200).

The amount of data generated after the reconstruction phase ranges from about 10% of the raw data to about the same size as the raw data, which amounts to about 30 - 300 TBs per year. Most of the time only the reconstructed data is needed for analysis, but the raw data must still be available. It is against the summary data that the physicists run their queries searching for qualifying events that satisfy those queries. All queries are range queries (for example, 5 GeV $<$ energy $<$ 7 GeV, or 10 $<$ number of pions $<$ 20). For each query, STACS has to determine which files contain the reconstructed data (or the raw data if they are requested), and to schedule their caching from tape for processing.

Given the fact that the different events (collisions) are independent of each other, it is irrelevant for the physicists whether they receive the qualifying events in the order they were generated or any other order, as long as they receive all qualifying events. So, what the physicists need is a away to map their queries to the qualifying events stored in the tape system and to efficiently retrieve those events from tape to their local disk so that they can run their analysis programs. STACS was designed with this in mind. It is typical that physicists form collaborations, where 10-100 users study the same region of the data. Therefore, there is good likelihood that queries of different users will overlap in the files that they need. STACS is designed to maximize the use of files once they are cached to disk, by striving to make each file available to all application programs that need it.

2.2 STACS

The STACS architecture consists of four modules that can run in a distributed environment: a Query Estimator (QE) module, a Query Monitor (QM) module, a File Catalog (FC) module and a Cache Manager (CM) module. All the communication between the different modules is handled through CORBA [5]. The architecture of the system is shown in Figure 1. The purpose of this paper is to describe in detail the capabilities provided by the CM. However, to put this in context we describe briefly the function of each module next.

The physicists interact with STACS by issuing a query that is passed to the QE. The QE utilizes a specialized index (called a bit-sliced index) to determine for each query all the events that qualify for the query and also the files where these events reside. This index was described in [4]. The QE can also provide time estimates before executing a query on how long it will take to get all the needed files from the tape system to local disk. The estimate takes into account the files that are currently in the disk cache. If the user finds the time estimate reasonable then a request to execute the query is issued and the relevant information about files and events is passed to the QM. The job of the QM is to handle such requests for file caching for all the users that are using the system concurrently. Since the users don't care about the order they receive the qualifying events the QM is free to schedule the caching of files in the way that it finds most efficient (for instance, by requesting first the files that most users want). The QM uses a fairly sophisticated caching policy module to determine which files should reside in cache at any time. The QM marks each file requested by one or more queries with a dynamic weight proportional to the number of queries that still need that file. The caching policy uses this weight to maximize the usage of the cache by queries. Any files that happen to be in cache and can be used by an application are passed to the application as soon as it is ready to accept the data (i.e. when it is not busy processing the previous data). Files are removed from cache only when space is necessary. The files with the lowest weight are removed first. A more detailed description of the caching policy is also given in [4].

After the QM determines which files to cache, it passes the file requests to the CM one at a time. The CM is the module that interfaces with the mass storage system, which in the case of STAR is HPSS. It is the job of the CM to make sure that the files requested

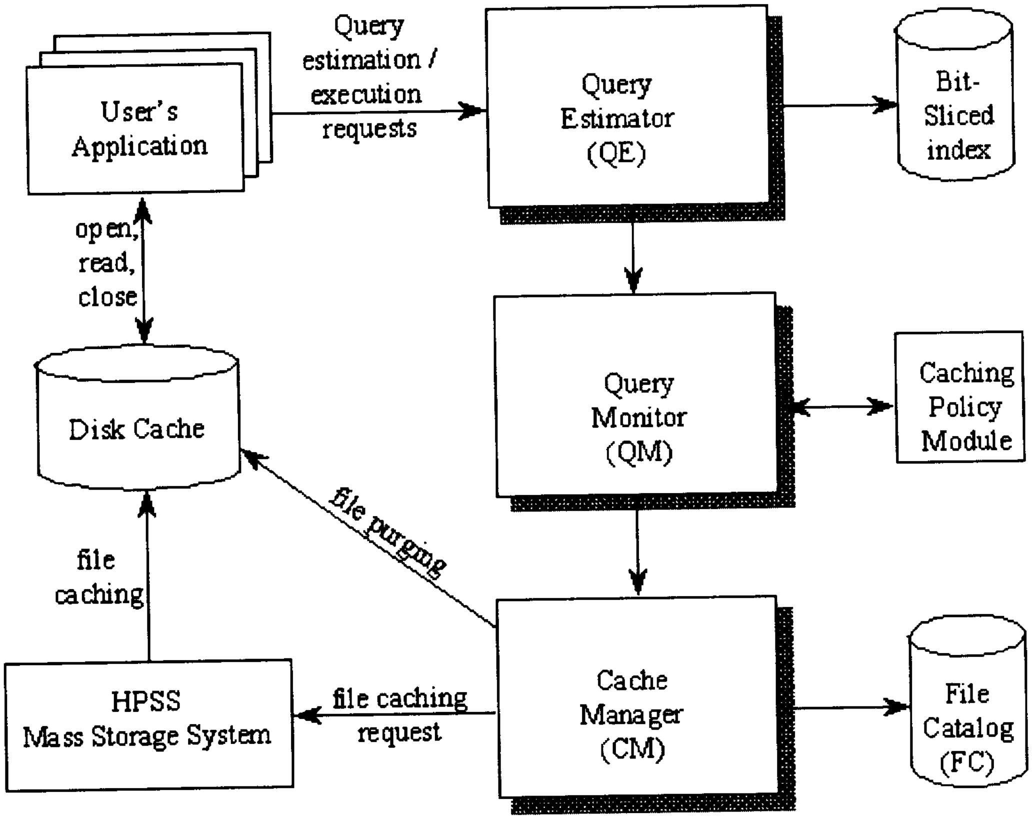


Figure 1: The STACS architecture.

by the QM are properly transferred to local disk. When a request reaches the CM a file is identified by a file id (*fid*), a logical name. To be able to transfer the file from HPSS to local disk the CM needs to convert the file logical name into a real physical name. This mapping can be obtained by consulting the FC, which provides a mapping of an *fid* into both a HPSS file name and a local disk file name (the full path of the file). It also includes information about the file size and the tape id (*tid*) of the tape where the file resides.

To visualize the operation of STACS, we include here a graph of a real run of the system processing multiple files (Figure 2) for a single query. The x-axis represents time. Each jagged vertical line represents the history of a single file. It starts at the bottom at the time it was requested, to the time it was cached to HPSS cache, to the time is was moved to the shared cache, to the time it was passed to the requesting query, and terminates (at the top) after the application finished processing all the events it needs from that file. As can be seen, initially a request for two files was made (one to process, and one to pre-fetch), and only after the first file was processed the application made a request to cache another file.

3 The Cache Manager

The CM performs mainly two functions: it transfers files from the mass storage system (HPSS) to local cache and purges files from local cache. Both actions are initiated by the QM. The transfer of files requires a constant monitoring. The CM performs a variety of

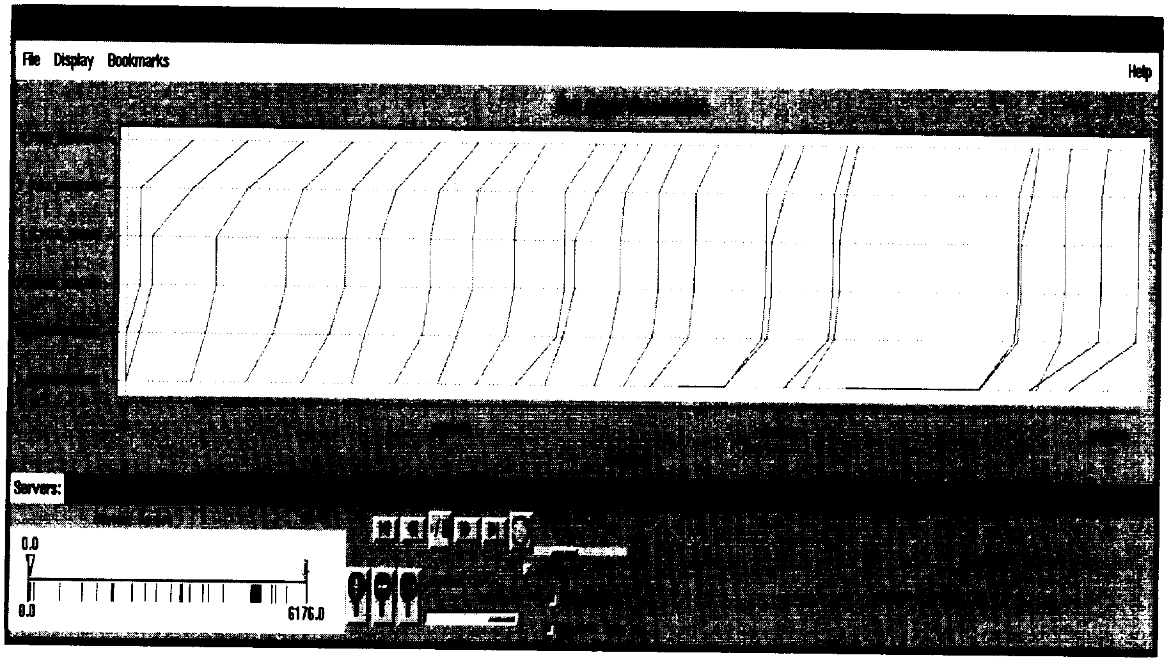


Figure 2: Tracking of files requested by a query.

different actions towards that end. It measures various quantities, such as the transfer rate of each file, it keeps track of the amount of cache in use, and (whenever a transfer fails) it detects the type of failure by parsing the PFTP output looking for errors.

3.1 File Transfers

The CM transfers files from the mass storage system (HPSS) to local cache using the parallel file transfer protocol (PFTP). The CM is multithreaded and can handle many file requests at the same time (in fact, there is a different thread for each PFTP request). Since the number of PFTPs that HPSS can handle concurrently is limited (by the memory available to HPSS) the CM needs to make sure that it doesn't swamp HPSS with too many concurrent PFTPs. This is a required feature because the HPSS system is a resource shared by many users and as such all users have to make sure they don't use more than their share. And even though the HPSS system administrator can block PFTP requests from any user, the system will work better if the users stay within their PFTP quotas. The CM handles this for all its users by queuing the file requests that it receives from the QM and never serving more than the number of PFTPs allocated to it. Thus, STACS and in particular the CM, performs the function of serving its users in a fair fashion, by not allowing any single user to flood the system with too many file caching requests. In STACS the number of allowed PFTPs can be changed dynamically by the system administrator, while the cache manager is running. If this limit is reduced, it simply stops issuing PFTPs until the number of PFTPs in progress reaches the new limit.

3.2 Queue Management

Since the CM builds up a queue of file requests that cannot be served while the number of PFTP requests is at its maximum, opportunities arise for rescheduling the order of requests in the queue so that files from the same tape are asked together one after another. The idea is that the transfer rate from HPSS to local cache will increase if the number of tape mounts is minimized. This is particularly important if the number of tape drives is small and the network bandwidth between HPSS and local cache is large. The goal is to have an aggregated transfer rate as high as possible and that can be achieved by minimizing the "idle" transfer periods during tape mounts. Obviously this gain obtained by rescheduling the queued requests comes at a cost, the cost of bypassing older requests in the queue and instead serving younger requests just because they happen to be from a more "popular" tape. We leave the responsibility of deciding how much rescheduling to do to the STACS administrator and that can be done by dynamically changing a "file clustering parameter" that characterizes the clustering of requested files according to the tape they reside in. Thus, choosing the parameter to be, say, 5 means that if a file from some tape was just transferred to local cache, then on average 4 more files from the same tape will be requested (this only holds true in an infinite queue, but it's a good approximation). Choosing the parameter to be 1 means that no rescheduling will be done and the files in the queue are served in a first come first serve order. Figures 3 and 4 show the order the files were requested versus the tape they reside in for two runs of the same set of queries. The "file clustering parameter"s used were 1 and 10 respectively. The important thing to notice is that in figure 3 there is a constant changing of tapes.

3.3 Query Estimates

One of the most interesting, and also the most difficult to implement, features of the CM is the capability of estimating how long the files needed for a query will take to transfer to local cache. Even though the users get the time estimate through the QE, the real estimates are done by the CM and passed to the QE. The estimates are done by checking which subset of the set of files needed for a query are in cache (call that X), which are already scheduled to be cached, and are in the CM queue (call that Y) and which still have to be requested (call that Z). The CM can use the current transfer rate to estimate how long the files needed will take to transfer. If the current transfer rate happens to be zero, either because no files are being transferred or because the network is temporarily down, then a default transfer rate is used. We describe in Section 3.5 how the actual transfer rates are obtained over time. So far, we used the maximum transfer rates obtained when the system is heavily loaded as the default transfer rate values. In the future, we plan to tune the default transfer rate dynamically, averaging the maximum transfer rates for the last 24 hours (or whatever default period is preferred).

To get a best case estimate, assuming this query gets top priority, we need only to divide the sum of sizes of files not in cache by the transfer rate Tr, i.e. $(s(Y) + s(Z))/Tr$ where $s(Y)$ and $s(Z)$ are the sum of sizes of files in set Y and set Z respectively.

However, we also want to get a realistic estimate. We achieve this as follows. For the X files that are in cache we assume they continue to be available to the application since they

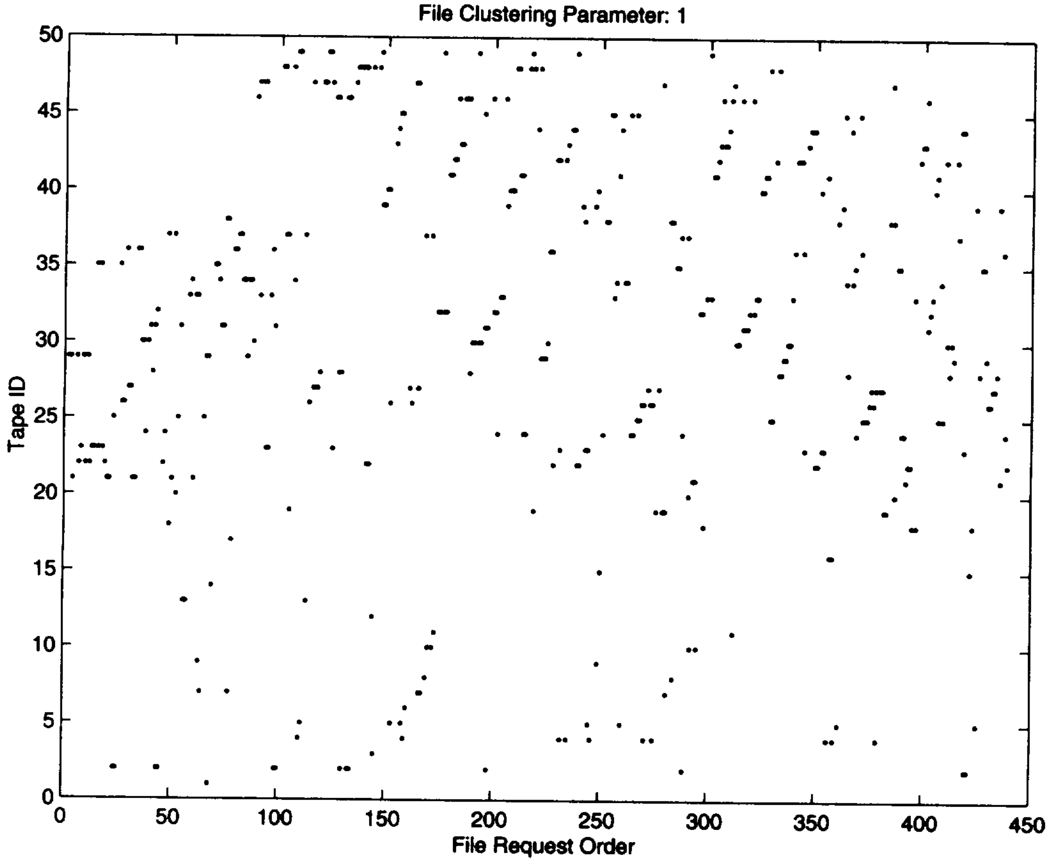


Figure 3: File request order in the absence of file clustering. Files are requested on a first come first serve basis. Each point in the x-axis corresponds to a new file request.

will be marked as needed. For the Y files in the CM queue, we have two cases to consider. If the set Z is empty then we don't need to consider the set of files in the queue that come after all the files in set Y. Call the set of remaining files in the queue Y' (we only need to consider the files in the queue from the first file to the last file in Y). Then the estimate is $s(Y')/Tr$. If on the other hand the set Z is not empty then we need to take into account that all the files in the queue need to be processed before any files in the set Z. We call the set of files in the queue T. Let then the number of queries in the system be q. For our estimate, we assume that each of the queries will be served in a round robin fashion, and that there is no file overlap between the queries. Then for the Z files we need $qs(Z)/Tr$, assuming that all files have similar sizes. So the total time estimate is $(qs(Z) + s(T))/Tr$.

Of course these estimates are only reasonably good if the system doesn't run out of cache space (in which case the file transfers have to stop until some files can be purged) and if the number of queries stays the same during the period that the query in question is being processed. Figures 6 and 5 show a comparison between the estimated time and the real time for the same set of twenty queries run from the same initial state (no files initially cached), with the difference that in one case the queries come 5 minutes apart and in the other case they come 20 minutes apart. In these runs the processing time per event (the time spent processing an event by the application) was chosen very small so that the amount of

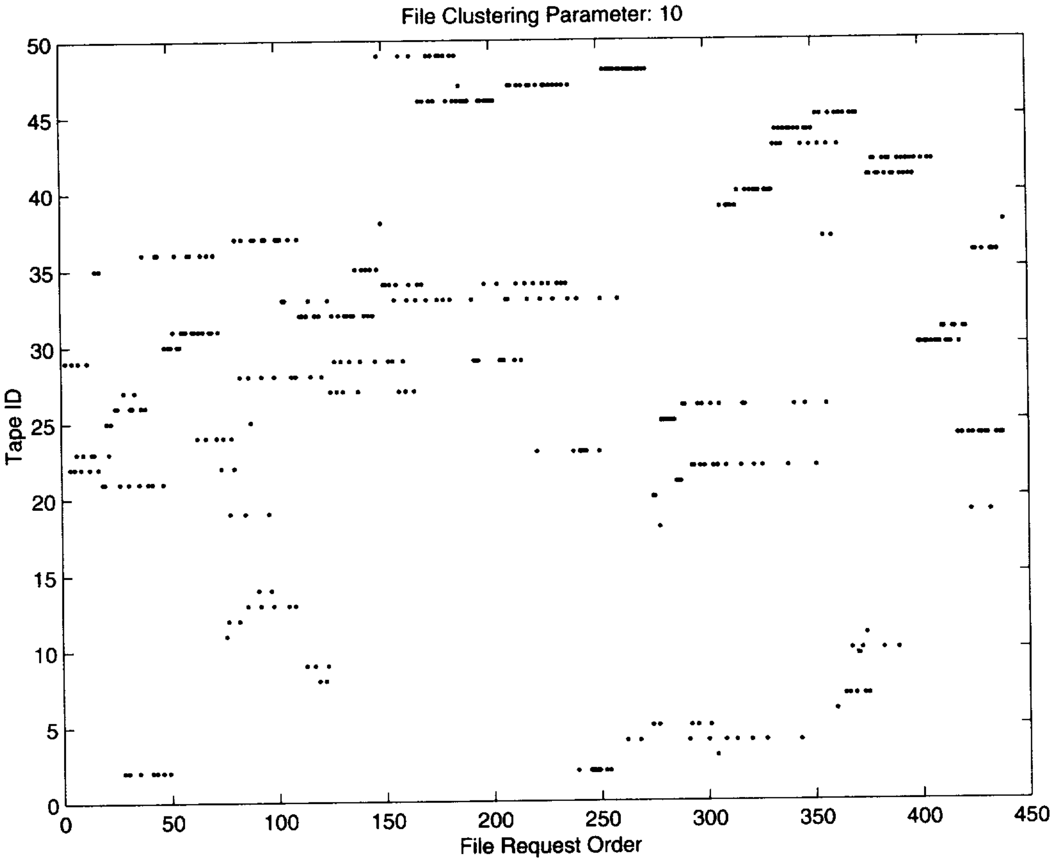


Figure 4: File request order with a file clustering parameter of 10 files per tape. As many as 10 successive requests from the same tape are made if they are found in the queue.

time the QM holds a file in cache is negligible when compared with the transfer time. The queries were designed to complete in about 20 minutes each. Figure 5 shows the estimates when the same set of queries arrive 20 minutes apart. This time is enough to transfer all the files needed by the query before the new query comes in. As a consequence the estimates are very accurate. They are biased towards shorter transfer times because the CM used the default transfer rate to calculate the transfer times, and the default transfer rate was chosen as the maximum transfer rate that the network supports. That default is not sustained for longer periods and hence the shorter time estimates.

On the other hand, in figure 6 the queries arrived 5 minutes apart. In addition, we did not take into account the number of queries that were in the system when a new query started. Since there was not enough time to finish a query before a new query arrives (we chose the queries so that they request approximately the same number of files every time), the requests for files pile up in the CM. This explains why successive time estimates grow larger and larger; the requests for files pile up faster than the CM can serve them. We can also see that the estimates were very poor and fell short of the real transfer times, because the estimate did not account for the number of queries in the system. This gave us the insight to take the number of queries into account, a feature that is now being implemented. We note that even so, the fact remains that when an estimate is done the CM knows nothing about

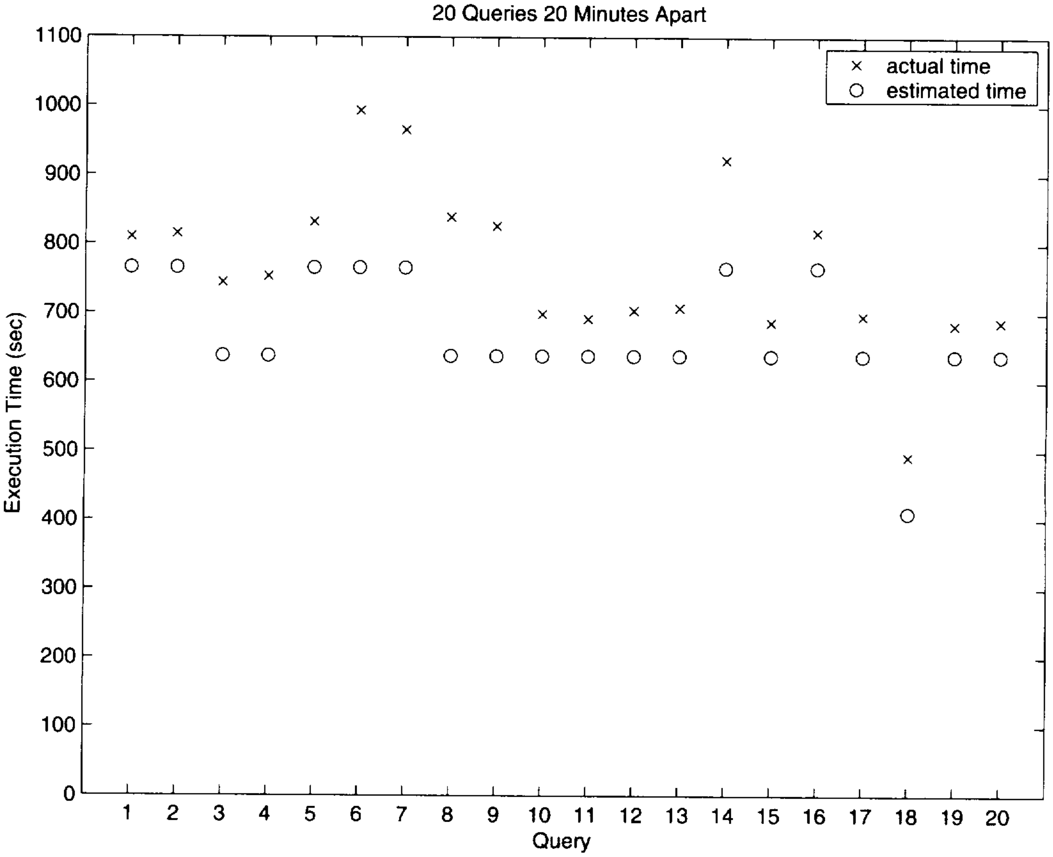


Figure 5: Comparison between estimated time and real transfer time when the queries run alone in the system.

the queries that will come in the future. Because of the round robin policy we currently use, such queries will request some files before all the files for previous queries were requested. Nevertheless, our estimates are pretty accurate since they are based on a measured transfer rate, the files in cache for that query, the number of files in the queue, the actual sizes of files, and the current load on the system, measured as the number of concurrent queries being processed.

3.4 Handling PFTP Errors

The most important functionality of the CM is the handling of transfer errors. Sometimes the PFTP transfer fails, either because HPSS misbehaves or breaks down, or because the network is down or even because the requested file doesn't exist in HPSS. So to make sure that the file was successfully transferred to local disk the CM starts by checking the PFTP output looking for the string "bytes transferred" (this string also appears at the end of a ftp transfer). If that string is not found the CM parses the PFTP output looking for possible error messages, and depending on the result different paths are taken. For instance, if the file doesn't exist on HPSS the CM just reports the fact to the QM. If on the other hand, the transfer error was due to some HPSS error (say, an I/O error) the CM removes the partially transferred file from disk, waits a few minutes, and then tries again to transfer the same file.

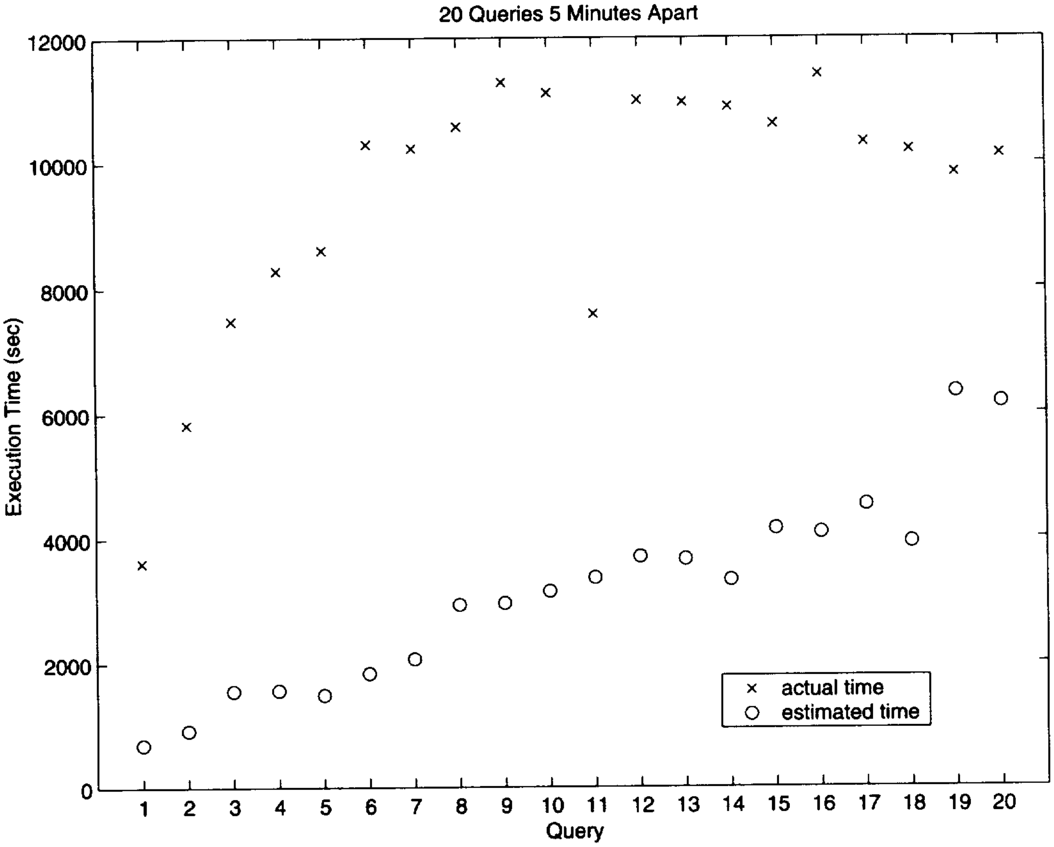


Figure 6: Comparison between estimated time and real transfer time when there is sharing of resources between queries.

This functionality of the CM is very important because it insulates the rest of the system and the user's application from HPSS and network transiént failures. All the user perceives is that the file may take longer to cache or that it doesn't exist. This situation is shown in Figure 2. It shows two gaps in the file transfers, one long and one shorter. This was due to an HPSS server failure that was then restored. The CM checked HPSS periodically till it recovered and then proceeded with file transfers.

The possible errors or reasons that cause a PFTP to fail are the following:

- File not found in HPSS. This is an irrecoverable error. The CM gives up and informs the QM.
- Limit PFTPs reached. This happens if other users use more than their share of allocated PFTPs. When this happens it is impossible to login to HPSS. The CM handles this by re-queuing the file request and trying again later.
- HPSS error. Some are recoverable (like an I/O error or a device busy error), others are not (a non existing file, or a wrong read permission). The CM handles the recoverable errors by trying again up to 10 times. This is a default number that can be changed dynamically. The assumption is that if a transfer fails 10 times then something is really wrong with the file. Another approach, which we did not implement, is to

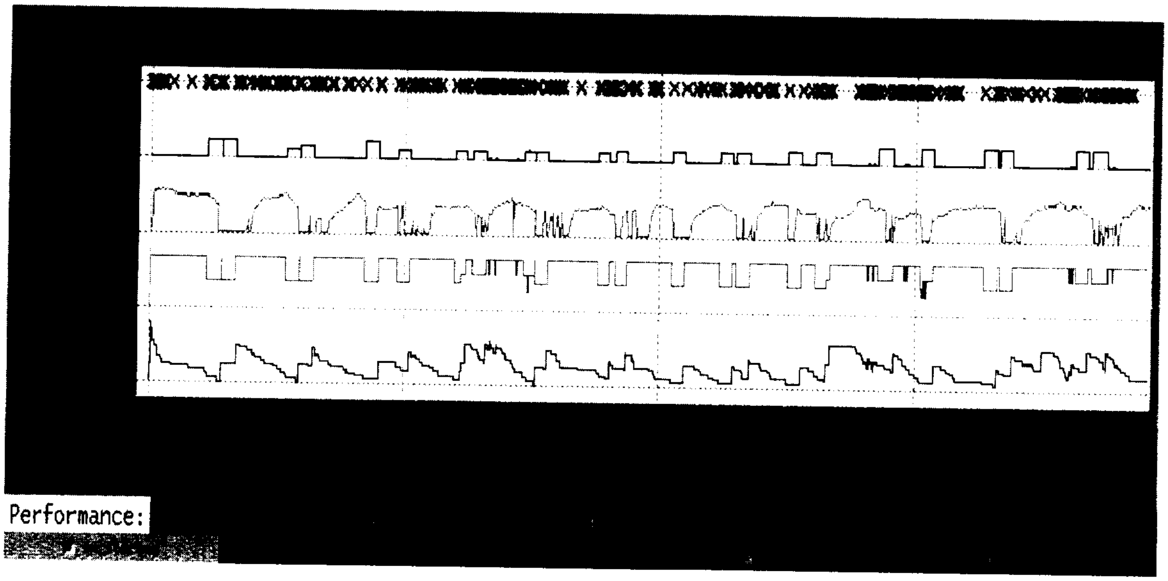


Figure 7: This graph shows several quantities that STACS can display dynamically and that characterize the overall status of the system.

have a timeout mechanism where no more PFTP retrials would be done once the timeout limit was reached.

3.5 Measurements

The CM keeps track of various quantities that characterize its status at any time. One of those, and probably the trickiest one to measure, is the transfer rate between HPSS and local cache. When a PFTP is launched the requested file transfer may not start right away. This is particularly true if the file happens to be on tape instead of being in the HPSS own cache. In that case the tape has to be mounted before the transfer can really start. This fact is not known to the CM. After the transfer occurs the CM can find out how much time was really used in transferring the file and how much time was used in mounting the tape and seeking to the right place on tape, but that information comes too late to be of any use in estimating the instantaneous transfer rate. The CM can give very accurate measurements of the instantaneous transfer rate by following a different approach: it periodically (say, every 15 seconds or whenever a file transfer ends) checks the local size of all the files currently being transferred. By measuring the total number of bytes transferred between now and the previous measurement and the amount of time elapsed, it can give an accurate value for the transfer rate. To smooth out quick fluctuations, it gives a moving average of the transfer rate measured over the last, say, 10 measurements.

Other quantities the CM keeps track of are the number of PFTPs pending, the amount of cache used by the files in local cache, and the amount of cache reserved for the requests currently in the queue. In addition to these measurements by the CM, the QM keeps track of information related to the status of queries. Specifically, it keeps also track of the number of queries waiting to be served or being served, and also the amount of cache actively being used, i.e., cache used by files that are being currently processed by some query. In this context, a query is considered as being served if it is currently processing a file, or if it has a file in local cache to process.

All these quantities can be displayed dynamically when the system runs and can be used by the STACS administrator to tune the policies of the system to overcome bottlenecks. For example, one of the parameters that can be set dynamically is how much pre-fetching to perform on behalf of each query. If there is a lot of available disk cache, and the PFTP queue is small, one can increase the number of pre-fetches, so that queries have at least one additional file in cache as soon as they finish processing a file. An example of such measurements displayed for a particular run are shown in Figure 7.

Another reason for keeping track of these measurements performance, is to observe whether the system resources are "balanced", i.e. used well for a typical query mix. In particular, it is important to understand where the bottlenecks are, and if some resources (tape drives, disk cache, and network resources) are underutilized. Accordingly, this can be used as a guide for adding the right kind of resources to the system to achieve better system performance.

3.6 Recovery from Crashes

One of the very important, even if rarely used, features of the CM is the capability to recover from crashes and return to its state before the crash. By crash we mean a real crash of the CM, which although very unlikely (we have run the CM for weeks without a glitch) cannot be put aside, but also the situation where the machine where the CM runs needs to be rebooted. Given the fact that a set of queries can take days to process it's of utmost importance that the system can return to its state before a crash without the users having to relaunch all the queries again. The CM does this by logging to a "recovery" file the list of requests that were not served yet. Once a new request arrives, information about it (file id and query id) is logged to a file, and after a request is served (a file is transferred) the associated information is removed from the same file. If the CM happens to crash or the system where it runs needs to be shut down, the CM can easily return to its previous state by reading the "recovery" file, and checking if the files were correctly transferred and are currently in cache. For any files not correctly transferred or not transferred at all, the CM relaunches the logged requests.

4 Conclusions

We described in this paper a real implementation of a storage access queuing and monitoring system to be used in high energy physics applications. The system is practically ready to be deployed and has been in a testing phase for the last few months. The system has been tested against a 1.6 TB federated database of synthetic data stored in 170 tapes. We have demonstrated the value of such a system in insulating the user's application from the details of interacting with a mass storage system. Specifically, the system enables the user to submit a query of what is needed, and the system finds all the files that need to be read from tape, schedules their caching so that files can be shared by multiple users, minimizes tape mounts, handles transient errors of the mass storage system and the network, and monitors performance. Such a system is particularly valuable for long running tasks (many hours) of 100's of files, where restarting a job because of a failure is not a practical option. Future plans include the application of the system in distributed multi-site grid infrastructure. In this setup, there can be multiple sites that have mass storage systems, and each site may have a shared disk cache for its local users. We envision the Cache Manager's functions to be associated with each storage resource in the system. An open (and difficult) problem is how to coordinate these distributed resource managers in order to support multiple users at various sites in the most efficient way. We also plan to apply this technology to application areas other than high energy physics.

References

[1] D. Düllmann. Petabyte databases. In *Proceedings of the 1999 ACM SIGMOD*, page 506, Philadelphia, Pennsylvania, 1-3 June 1999. ACM Press.

[2] A. Hanushevsky. Pursuit of a scalable high performance multi-petabyte database. In *Proceedings of the 16th IEEE Symposium on Mass Storage Systems*, pages 169–175, San Diego, California, 15-18 March 1999. IEEE Computer Society.

[3] J. Shiers. Massive-scale data management using standards-based solutions. In *Proceedings of the 16th IEEE Symposium on Mass Storage Systems*, pages 1–10, San Diego, California, 15-18 March 1999. IEEE Computer Society.

[4] A. Shoshani, L. M. Bernardo, H. Nordberg, D. Rotem, and A. Sim. Multidimensional indexing and query coordination for tertiary storage management. In *Proceedings of the International Conference on Scientific and Statistical Database Management*, pages 214–225, Cleveland, Ohio, 28-30 July 1999. IEEE Computer Society.

[5] A. Sim, H. Nordberg, L. M. Bernardo, A. Shoshani, and D. Rotem. Storage access coordination using CORBA. In *Proceedings of the International Symposium on Distributed Objects and Applications*, pages 168–175, Edinburgh, UK, 5-6 Sept. 1999. IEEE Computer Society.

DataCutter: Middleware for Filtering Very Large Scientific Datasets on Archival Storage Systems *

Michael Beynon[†], Renato Ferreira[†], Tahsin Kurc[†], Alan Sussman[†], Joel Saltz[†‡]

[†] Department of Computer Science
University of Maryland
College Park, MD 20742

[‡] Department of Pathology
Johns Hopkins Medical Institutions
Baltimore, MD 21287

{beynon,renato,kurc,als,saltz}@cs.umd.edu

Abstract

In this paper we present a middleware infrastructure, called DataCutter, that enables processing of scientific datasets stored in archival storage systems across a wide-area network. DataCutter provides support for subsetting of datasets through multi-dimensional range queries, and application specific aggregation on scientific datasets stored in an archival storage system. We also present experimental results from a prototype implementation.

1 Introduction

Increasingly powerful computers have made it possible for computational scientists and engineers to model physical phenomena in great detail. As a result, overwhelming amounts of data are being generated by scientific and engineering simulations. In addition, large amounts of data are being gathered by sensors of various sorts, attached to devices such as satellites and microscopes. The primary goal of generating data through large scale simulations or sensors is to better understand the causes and effects of physical phenomena. Thus, the exploration and analysis of large datasets plays an increasingly important role in many domains of scientific research. Simulation or sensor datasets generated or acquired by one group may need to be accessed over a wide-area network by other groups. Software support is needed to allow users to obtain needed subsets of very large, remotely stored datasets.

We present a middleware infrastructure, called DataCutter, that enables processing of scientific datasets stored in archival storage systems across a wide-area network. DataCutter provides support for subsetting of datasets through multi-dimensional range queries, and application specific aggregation on scientific datasets stored in an archival storage system. We discuss an implementation of the Virtual Microscope application [2] using DataCutter. The Virtual Microscope is representative of data-intensive applications that involve browsing and processing large multi-dimensional datasets. Other examples include satellite data processing systems [7] and water contamination studies that couple multiple simulators [20]. We also provide experimental performance results for a prototype implementation.

*This research was supported by the National Science Foundation under Grant #ACI-9619020 (UC Subcontract # 10152408), and the Office of Naval Research under Grant #N6600197C8534.

2 Motivation and Overview

Over the past several years we have been actively working on data intensive applications that employ large-scale scientific datasets, including applications that explore, compare, and visualize results generated by large scale simulations [20], visualize and generate data products from global coverage satellite data [7], and visualize and analyze digitized microscopy images [2]. Many scientific applications generate and use datasets consisting of data values associated with a multi-dimensional space. Scientific simulations typically generate datasets with at least three spatial dimensions and a temporal dimension. Satellite data and microscopy data generally have two (or more) spatial dimensions and a temporal dimension. Applications frequently need to access spatially defined data subsets via a *spatial range query*, which is a multi-dimensional box in the underlying dataset space. Spatial subsets can encompass contiguous regions of space, as for retrieving satellite data covering a particular geographical region. Spatial subsets can also be defined once features of interest are categorized using spatial indices. For instance, subsetting can be carried out to retrieve simulation data associated with shocks in fluid simulations, or tissue regions with particular cell types in microscopy datasets.

There are various situations in which application-specific non-spatial subsetting and data aggregation can be applied to targeted data subsets. Some data analysis require values for only some of variables at a data point. For example, a computational fluid dynamics simulation dataset can be organized so each data element contains velocity, momentum, and pressure values. An analysis code may only use the pressure value at a grid point, and may ignore values for velocity and momentum. In other cases, there may be a need to obtain an application-dependent low resolution view of a dataset. For example, a hydrodynamics simulation may generate and store flow data (e.g., velocity values) at fine time steps. The analysis may need to be performed using coarser time steps, which requires the stored velocity values to be averaged over several time steps. In these cases, aggregation and transformation operations could be applied to data elements at the data server where they are stored, before returning them to the client where the analysis program is run.

In some cases data analysis can be employed in a collaborative environment, where colocated clients access the same datasets and perform similar processing. For instance, a large group of students in medical training may need to simultaneously explore the same set of digitized microscopy slides, or visualize the same MRI and CT datasets. There may be a large number of overlapping regions of interest, and common processing requirements (e.g., same magnification level for microscopy images, or same transfer functions to convert scalar values into color values) among the users employing the analysis tools (*clients* of the data server). In these cases, caching reused dataset portions closer to the clients (i.e., on the same local area network) can provide significant performance benefits.

We have developed the Active Data Repository (ADR) [6] framework to use for developing parallel applications that make use of large centralized scientific datasets. ADR provides support for accessing subsets of multi-dimensional scientific datasets via range queries, and allows users to integrate user-defined processing of large centralized datasets with storage and retrieval on distributed memory parallel machines with multiple disks attached to each node. ADR is designed as a set of *customizable* and *internal* services. Through the use of customizable services, users can specify and implement application specific dataset

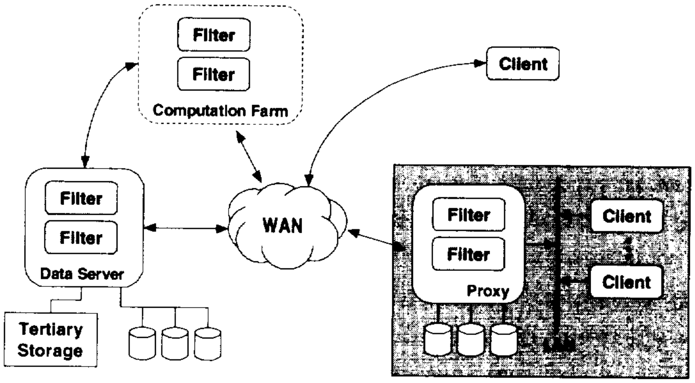


Figure 1: Architecture of the data management/manipulation framework.

indexing, and user-defined aggregation and transformation operations used in processing one or more datasets. The internal services provide support for common operations such as memory management, data retrieval, management of multiple datasets, and query planning and scheduling of processing on a parallel machine. A number of applications have been developed using ADR and good performance has been demonstrated [6, 20]. However, the continuing increase in the capabilities of high performance computers and sensor devices implies that datasets with sizes up to petabytes will be common in the near future. Such vast amounts of data require the use of archival storage systems distributed across a wide-area network. Data analysis, on the other hand, is usually performed on machines at an application scientist's local institution. Efficient storage, retrieval and processing of multiple large scientific datasets on remote archival storage systems is therefore one of the major challenges that needs to be addressed for efficient exploration and analysis of these datasets.

There is a large body of hardware and software research on archival storage systems, including distributed parallel storage systems [19], file systems [23], image servers [22], and data warehouses [18]. Several research projects have focused on digital libraries and geographic information systems [3, 14] that access collections of archival storage systems, high-performance I/O systems [9] and remote I/O [11, 21]. In addition to many end-point solutions, the *Grid* [8, 10, 13] has been emerging in recent years as infrastructure to link distributed computational, network and storage resources, and to provide services for unified, secure, efficient and reliable access. Several research projects have focused on providing services in a Grid environment, such as Globus [12], which provides services to access computational resources, and the Storage Resource Broker (SRB) [21], which provides uniform UNIX-like I/O interfaces and meta-data management services to access collections of distributed data resources. However, providing support for efficient exploration and processing of very large scientific datasets stored in archival storage systems in a Grid environment remains a challenging research issue, and the necessity of infrastructure to provide such support was recognized in recent Grid forums [16].

We are developing an infrastructure to make it possible to explore and analyze scientific datasets stored on archival storage across a wide-area network. Figure 1 illustrates the framework architecture. It consists of two major components: *DataCutters* and *Proxies*. A proxy provides support for caching and management of data near a set of clients. The goal is to reduce the response time seen by a client, decrease the amount of redundant data transferred across the wide-area network, and improve the scalability of data servers. Our prior work on proxies can be found in [5].

The new middleware infrastructure, called DataCutter, provides support for processing of scientific datasets stored in archival storage systems in a wide-area network. DataCutter provides a core set of services, on top of which application developers can implement more application-specific services or combine with existing Grid services such as meta-data management, resource management, and authentication services. Our main design objective in DataCutter is to extend and apply the salient features of ADR (i.e. support for accessing subsets of datasets via range queries and user-defined aggregations and transformations) for very large datasets in archival storage systems, in a shared distributed computing environment. In ADR, data processing is performed where the data is stored (i.e. at the data server). In a Grid environment, however, it may not always be feasible to perform data processing at the server, for several reasons. First, resources at a server (e.g., memory, disk space, processors) may be shared by many other competing users, thus it may not be efficient and cost-effective to perform all processing at the server. Second, datasets may be stored on distributed collections of storage systems, so that accessing data from a centralized server may be very expensive. Moreover, distributed collections of shared computational and storage systems can provide a more powerful and cost-effective environment than a centralized server, if they can be used effectively. Therefore, to make efficient use of distributed shared resources within the DataCutter framework, the application processing structure is decomposed into a set of processes, called *filters*. DataCutter uses these distributed processes to carry out a rich set of queries and application specific data transformations. Filters can execute anywhere (e.g., on computational farms), but are intended to run on a machine close (in terms of network connectivity) to the archival storage server or within a proxy (see Figure 1). Filter-based algorithms are designed with predictable resource requirements, which are ideal for carrying out data transformations on shared distributed computational resources.

Many filter-based algorithms were originally developed and analyzed by our group for Active Disks [1, 24]. These filter-based algorithms carry out a variety of data transformations that arise in earth science applications and applications of standard relational database sort, select and join operations. In the DataCutter framework we are extending these algorithms and investigating the application of filters and the stream-based programming model in a Grid environment.

Another goal of DataCutter is to provide common support for subsetting very large datasets through multi-dimensional range queries. Very large datasets may result in a large set of large data files, and thus a large space to index. A single index for such a dataset could be very large and expensive to query and manipulate. To ensure scalability, DataCutter uses a multi-level hierarchical indexing scheme. In the following sections we describe the DataCutter infrastructure, in particular the indexing and filtering services, and present an implementation of the Virtual Microscope [2] using DataCutter.

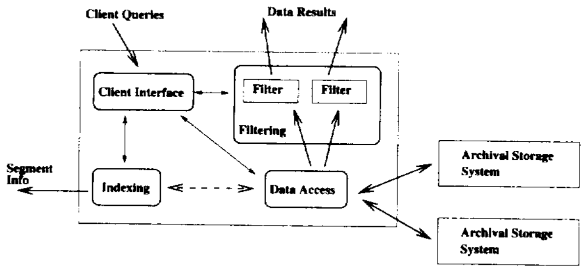


Figure 2: DataCutter system architecture.

3 DataCutter

The architecture of DataCutter (Figure 2) is being developed as a set of modular services. The client interface service interacts with clients and receives multi-dimensional range queries from them. The data access service provides low level I/O support for accessing the datasets stored on archival storage systems. Both the filtering and indexing services use the data access service to read data and index information from files stored on archival storage systems. The indexing service manages the indices and indexing methods registered with DataCutter. The filtering service manages the filters for application-specific aggregation operations. In the following sections we describe the indexing and filtering services in more detail.

3.1 Indexing

A DataCutter supported dataset consists of a set of data files and a set of index files. Data files contain the data elements of a dataset; data files can be distributed across multiple storage systems. Each data file is viewed as consisting of a set of *segments*. Each segment consists of one or more data items, and has some associated metadata. The metadata for each segment consists of a minimum bounding rectangle (MBR), and the offset and size of the segment in the file that contains it. Since each data element is associated with a point in an underlying multi-dimensional space, each segment is associated with an MBR in that space, namely a hyperbox that encompasses the points of all the data elements contained in the segment. Spatial indices are built from the MBRs for the segments in a dataset. A segment is the unit of retrieval from archival storage for spatial range queries made through DataCutter. When a spatial range query is submitted, entire segments are retrieved from archival storage, even if the MBR for a particular segment only partially intersects the range query (i.e. only some of the data elements in the segment are requested).

One of the goals of DataCutter is to provide support for subsetting very large datasets (sizes up to petabytes). Efficient spatial data structures have been developed for indexing and accessing multi-dimensional datasets, such as R-trees and their variants [4]. However, storing very large datasets may result in a large set of data files, each of which may itself be very large. Therefore a single index for an entire dataset could be very large. Thus, it may be expensive, both in terms of memory space and CPU cycles, to manage the index, and to perform a search to find intersecting segments using a single index file. Assigning an index file for each data file in a dataset could also be expensive because it is then necessary to access all the index files for a given search. To alleviate some of these problems, DataCutter uses a multi-level hierarchical indexing scheme implemented via *summary index* *files* and *detailed index files*. The elements of a summary index file associate metadata (i.e. an MBR) with one or more segments and/or detailed index files. Detailed index file entries themselves specify one or more segments. Each detailed index file is associated with some set of data files, and stores the index and metadata for all segments in those data files. There are no restrictions on which data files are associated with a particular detailed index file for a dataset. Data files can be organized in an application-specific way into logical groups, and each group can be associated with a detailed index file for better performance. For example, in satellite datasets, each data file may store data for one week. A detailed index file can be associated with data files grouped by month, and a summary index file can contain pointers to detailed index files for the entire range of data in the dataset. DataCutter uses R-trees as its default indexing method. However, the infrastructure allows users to add new indices and indexing methods (through the use of C++ class inheritance).

3.2 Filters

In DataCutter, *filters* are used to perform non-spatial subsetting and data aggregation. Filters are managed by the filtering service. A filter is a specialized user program that pre-processes data segments retrieved from archival storage before returning them to the requesting client. Filters can be used for a variety of purposes, including elimination of unnecessary data near the data source, pre-processing of segments in a pipelined fashion before sending them to the clients, and data aggregation. Filters are executed in a restricted environment to control and contain their resource consumption. Filters can execute anywhere[1], but are intended to run on a machine close (in terms of network connectivity) to the archival storage server or within a proxy (see Figure 1). When run close to the archival storage system, filters may reduce the amount of data injected into the network for delivery to the client. Filters can also be used to offload some of the required processing from clients to proxies or the data server, thus reducing client workload.

Filters are written in a *stream-based programming model*, originally developed for programming Active Disks [1]. A filter consists of an initialization function, a processing function, and a finalization function. The initialization function is run when the filter is first installed on the data server. The processing function is run repeatedly as new data arrives at the filter input ports (via streams). The finalization function is run when the filter terminates (either by consuming the data on all its input streams or by calling `exit`).

The programming model for filters is built around the notion of a stream abstraction. A stream denotes a supply of data to or from the storage media, or a flow of data between two application components, such as between two separate filters or between a filter and a client. Streams can be of two types – file streams and pipe streams. File streams are a sequence of ranges in files, and constitute the primary access method for data residing in secondary or archival storage. Pipe streams are a representation of a unidirectional flow of data between any two components of the application, and are used for both control interaction and data transfer. The stream-based programming model provides and enforces a standard interface for accessing streams [1]. Streams deliver data in fixed-size buffers whose size is fixed at

[1] Filters do not migrate state, and are not written in a platform independent language such as Java, but rather are compiled for the target platform and placed by the DataCutter filter service at runtime.

the time the stream is created. The size of the stream buffer cannot be changed after its creation. A filter may, optionally, contain scratch space, which is allocated on its behalf before it is initialized and is automatically reclaimed after it exits. Filters specifically cannot dynamically allocate and deallocate space, which allows the filtering service to better perform scheduling and buffer management and enable execution in environments with limited resources (e.g., memory).

Communication between a filter and its environment is restricted to its input and output streams. The sources and sinks for these streams are specified by the client program as a part of filter installation. A filter cannot determine (or change) where its input stream comes from or where its output stream goes to. This has two advantages. First, a filter does not need to handle buffering and scheduling for its own communication, thereby reducing the complexity of filters. Second, filters can be transparently executed in proxies or other convenient locations as resource constraints at the client and/or server change.

4 An Example: the Virtual Microscope using DataCutter

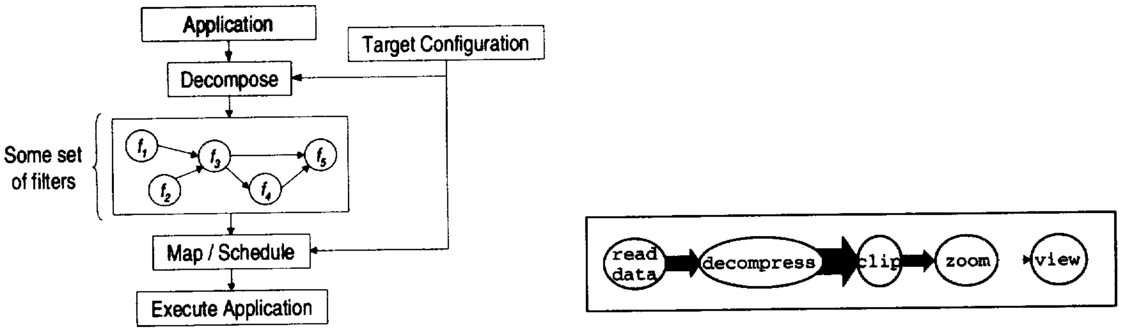
In this section we describe an implementation of the Virtual Microscope application [2] using the DataCutter infrastructure.

4.1 The Virtual Microscope

The Virtual Microscope is a client-server software system, which is designed to realistically emulate a high power light microscope. The data used by the Virtual Microscope are digitized images of full microscope slides at high power. Digitized images from a slide effectively form a three-dimensional (3D) dataset because each slide may contain multiple focal planes, each of which is a 2D image. Images are stored at the highest magnification level, and the size of a single slide typically varies from $100MB$ to $5GB$, compressed. At a basic level, the system is required to provide interactive response times similar to a physical microscope, including continuously moving the stage and changing magnification. A typical query allows a client to request a 2D rectangular region at a particular magnification from within the bounds of a single focal plane. The processing for the query requires projecting high resolution data onto a grid of suitable resolution (governed by the desired magnification) and appropriately compositing pixels that map to a single grid point, to avoid introducing spurious artifacts into the displayed image. The Virtual Microscope can support completely digital dynamic telepathology [2], as well as enabling new modes of operation that cannot be achieved with a physical microscope, such as simultaneous viewing and manipulation of a single slide by multiple users.

4.2 The Original Implementation

The original Virtual Microscope system is composed of two components; a client to generate queries and display the results (i.e. images), and a server, implemented with the Active Data Repository, to process the queries. A protocol has been defined between the client and the server for exchanging queries and results. The server is composed of a frontend and a backend. The frontend interacts with clients; it receives queries from clients and forwards



(a) Overall process

(b) Virtual Microscope filters

Figure 3: Process of applying filter and stream-based programming model.

them to the backend. The backend consists of one or more processes (when run on a parallel machine). The processing of a query is carried out entirely in the backend.

In order to achieve high I/O bandwidth, each focal plane in a slide is regularly partitioned into data chunks, each of which is a rectangular subregion of the 2D image. When the host machine is a parallel machine with multiple disks attached to each processor, data chunks are declustered across all the disks to achieve I/O parallelism. Each pixel in a chunk is associated with a coordinate (in x- and y-dimensions) in the entire image. As a result, each data chunk is associated with a minimum bounding rectangle (MBR), which encompasses coordinates of all the pixels in the chunk. An index is created using the MBR of each chunk. Since the image is regularly partitioned into rectangular regions, a simple lookup table, consisting of a 2-dimensional array, serves as an index.

During query processing, the backend process finds the chunks that intersect the query region, and reads them from the local disks. In the original server implementation, each data chunk is stored in compressed form (JPEG format). Hence, each retrieved chunk is first *decompressed*. Then, it is *clipped* to the query region. Afterwards, each clipped chunk is *subsampled* to achieve the magnification (*zoom*) level given in the query. The resulting image blocks are directly sent to the client. The client *viewer* assembles and displays the image blocks from each of the backend processes to form the query output.

4.3 An Implementation using DataCutter

Developing an application with the DataCutter infrastructure requires partitioning of a dataset used by the application into segments, and building spatial indices on the segments. DataCutter provides default interfaces to create and search R-trees. In this implementation of the Virtual Microscope, we employed the data chunks in the original implementation as the segments, and used the default R-tree indexing method of DataCutter. Each focal plane consists of a partitioned set of data files, each with a single detailed index file, and one summary index file is created to index all focal planes in a slide.

The next step in developing the application is to implement the application specific processing using filters and the stream-based programming model. Figure 3(a) illustrates the general steps for implementing an application using filters. First, the application processing structure is decomposed into a set of filters. An important issue is how to choose the number of filters for implementing the application processing. For instance, the original server im-

```
VM_zoom::init() {
  // Allocate from pre-allocated scratch space
  bufOut = AllocFromScratch(getOutputStreamBufferSize());
}
VM_zoom::process(stream_t &st) {
  DC_StreamBuffer *buf;
  VMQuery *query;
  VMChunk *chunk;

  // receive and extract the query
  buf = st.ins[0].read();
  query = VMUnpackQuery(buf);

  // while more data to read from input stream
  while ((buf = st.ins[1].read()) != NULL) {
    // extract chunk and perform zoom
    chunk = VMUnpackChunk(buf);
    zoom_chunk(chunk, query);

    // pack into buffer and write to output stream
    bufOut = VMPackChunk(chunk);
    st.outs[0].write(&bufOut);
    FreeToScratch(chunk->Data);
  }
}
VM_zoom::finalize() {
  FreeToScratch(bufOut);
}
```

```
void VM_zoom::zoom_chunk(VMChunk *chunk,
                         VMQuery *query) {
  int rel_zoom = query->Zoom/chunk->Zoom;
  int width = chunk->Width/rel_zoom;
  int height = chunk->Height/rel_zoom;
  int size = width*height*PIXELSIZE;

  char *pSrc = chunk->Data;
  char *pDst = chunk->Data = AllocFromScratch(size);
  // subsample the image block
  for (j = height; j>0; --j) {
    for (i = width; i>0; --i) {
      memcpy(pDst, pSrc, PIXELSIZE);
      pSrc += rel_zoom*PIXELSIZE;
      pDst += PIXELSIZE;
    }
    pSrc += rel_zoom*chunk.Width*PIXELSIZE;
  }
  // update chunk metadata
  chunk->Zoom = query->Zoom;
}
```

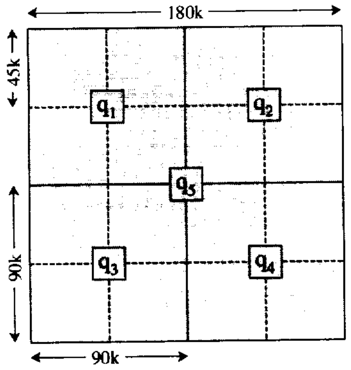
Figure 4: Zoom filter pseudo-code, which performs subsampling of an image chunk based on the query magnification (zoom).

plementation could be considered a single filter. In choosing the appropriate decomposition, we need to consider the complete data flow path from data generation to ultimate consumption as well as the target machine configuration, which can be a distributed collection of heterogeneous machines. The main goal is to achieve efficient use of limited resources in a distributed and heterogeneous environment. We are currently developing techniques and guidelines to assist in this important step.

The selected decomposition of the Virtual Microscope system into filters is shown in Figure 3(b). The figure only depicts the main dataflow path of image data through the system; other streams related to the client-server protocol are not shown for clarity. In this implementation each of the main processing steps in the server is a filter:

- **read-data:** Each full-resolution rectangular image block (i.e. data chunk) that intersects the query window is read from disk, and immediately written to the output stream before the next read operation.
- **decompress:** Image blocks are read individually from the input stream. The block is decompressed using JPEG decompression and converted into a 3 byte RGB format, and the block's metadata header is changed to indicate the image block's new format. The image block is then written to the output stream.
- **clip:** Uncompressed image blocks are read from the input stream. Portions of the block that lie outside the query region are removed, and the clipped image block is written to the output stream.
- **zoom:** Image blocks are read from the input stream, one block at a time. Using the requested magnification in the query, image blocks are subsampled to achieve the desired magnification. The resulting image block is written to the output stream.
- **view:** Image blocks are received for a given query, collected into a reply required for the Java client, and sent to the client.

Figure 4 illustrates the high-level code for the zoom filter. Implementation of the filters is done through C++ class inheritance and virtual functions. The DataCutter infrastructure



Filter	*Total Volume*	*Volume Per Chunk*
read data	3.60 MB	102.52 KB
decompress	83.42 MB	2373.04 KB
clip	57.83 MB	1645.02 KB
zoom	.90 MB	25.70 KB

Figure 5: The 2-dimensional dataset and queries used in the experiments. The solid and dashed lines show different partitionings of the dataset into files for the experiments. The table shows transmitted sizes for q_5.

provides base classes and virtual functions for *initialization*, *processing*, and *finalization* operations in filters, as well as functions to set scratch space size and stream buffer size (not shown in the figure). The zoom filter has two input streams and one output stream. It reads the query from stream 0 (st.ins[0]) and data from stream 1 (st.ins[1]), and subsamples the received data chunks using the zoom_chunk function. The zoom filter uses scratch space to store the results during subsampling and to pack the subsampled chunk into the output buffer. The result is written to the output stream (st.outs[0]), which connects the filters *zoom* and *view*.

As is discussed in Section 3, streams between filters deliver the data in individual fixed-size buffers. In the current implementation we send data chunks in stream buffers, and the size of the buffer is chosen to be the maximum size of a chunk in the dataset. This allows us to reuse code from the original Virtual Microscope implementation with little modification.

5 Experimental Results

We have developed a prototype implementation of the DataCutter services. Using this prototype, we have implemented a simple data server for digitized microscopy images stored on the IBM HPSS system [17] at the University of Maryland. The implementation is based on the functionality of the Virtual Microscope and uses the filters described in Section 4.

Our HPSS setup has 10TB of tape storage space and 500GB of disk cache, and is accessed through a 10-node IBM SP with 4 multiprocessor (1 4-processor and 3 2-processor) and 6 single processor nodes. In all of the experiments, we use a 4GB 2D image dataset, in JPEG compressed format (90GB uncompressed), created by stitching together small digitized microscopy images. This dataset is equivalent to a digitized slide with a single focal plane that has $180K \times 180K$ RGB pixels. The 2D image is regularly partitioned into 200×200 data segments and stored in the HPSS as a set of files. For all experiments we use 5 different queries, each of which covers 5×5 segments of the image (see Figure 5). Execution times presented in this section are the response time seen by the visualization client (including submitting a query and receiving the results) and are the average of processing each query 5 times. One node of the IBM SP is used to run the indexing service, and the client was run on a SUN workstation connected to the SP node through the depart-

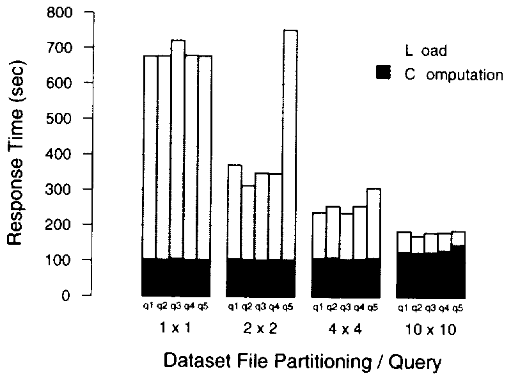
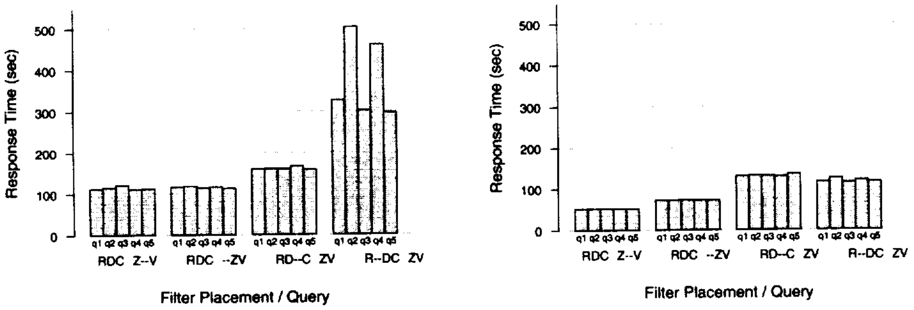


Figure 6: Query execution time with the dataset organized into 1×1, 2×2, 4×4 and 10×10 files. *Load* shows the time to open and access the files, which contain segments that intersect a query. *Computation* shows the sum of the execution time in the indexing service, for searching segments that intersect a query, and in the filtering service, for processing the retrieved data via filters. All filters and the indexing service were run on the same SP node.

ment Ethernet. We experimented with different placements of the filters by running some of the filters (and the filtering service) on the same SP node where the indexing service is executed, as well as on the SUN workstation where the client is run.

The first experiment isolates the impact of organizing the dataset into multiple files. Figure 6 shows the results when the 2D image is partitioned into 1×1, 2×2, 4×4 and 10×10 rectangular regions, and all data segments in each region are stored in a data file. Figure 5 illustrates the partitioning of the dataset into 1×1 (entire rectangle), 2×2 (solid lines), and 4×4 (dashed lines) files. Each data file is associated with a *detailed index* file, and there is one *summary index* file for all the detailed index files for each partitioning. As is seen in the figure, the *load* time decreases as the number of files is increased. This is because of the fact that HPSS loads the entire file onto disks used as the HPSS cache when a file is opened. When there is a single file, the entire 4GB file is accessed from HPSS for each of the queries– in these experiments, all data files are purged from disk cache after each query is processed. When the number of files increases, only a subset of the detailed index files and data files are accessed using the multi-level hierarchical indexing scheme, decreasing the time to access data segments. Note that the *load* time for query 5 for the 2×2 case is substantially larger than that of other queries, because query 5 intersects segments from each of the four files (Figure 5), hence the same volume of data is loaded into the disk cache as in the 1×1 case. The load time for that query is also larger than that in 1×1 case because of the overhead of seeking/loading/opening four files instead of a single file. The *computation* time, on the other hand, remains almost the same, except for the 10×10 case, where it slightly increases. Each query intersects more files as the dataset is partitioned more finely. As a result, the overhead from opening and accessing a large number of detailed index files can increase the computation time. These results, demonstrate that applications can take advantage of the multi-level hierarchical indexing scheme by organizing a dataset into an appropriate set of files. However, having too many files may increase computation time, potentially decreasing overall efficiency when multiple similar queries are executed on the same dataset.



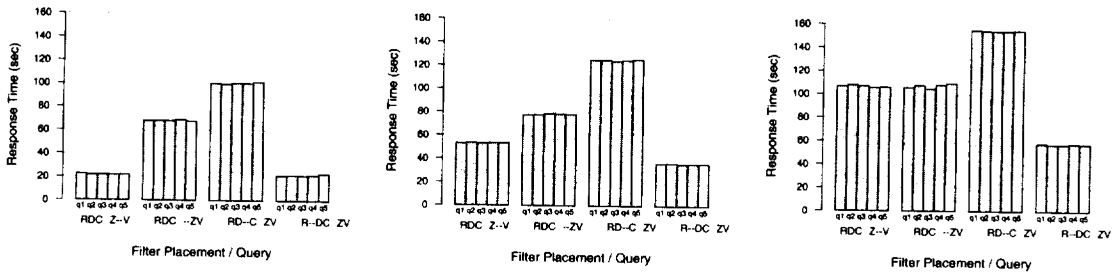
(a) no zooming (no subsampling) (b) subsampling by a factor of 8

Figure 7: Execution time of queries under varying zoom (subsampling) factor. R,D,C,Z,V denote the filters *read_data*, *decompress*, *clip*, *zoom*, and *view* respectively. {server}–{client} denotes the placement of the filters in each set.

Next, we consider varying the placement of the filters under different conditions. Figures 7 and 8 show query execution times for different filter placements under varying processing requirements (i.e. the subsampling factor) and varying server load, where the server is the machine where the *read_data* filter is run. The different server loads in Figure 8 were emulated by artificially slowing down the set of filters running on the server machine. Figure 7 shows the query execution times when the image is viewed at the highest magnification (no subsampling) and when the subsampling factor is 8 (i.e. only every 8th pixel in each dimension is displayed). As is seen from the figure, when there is no subsampling, query execution times remain almost the same whether the zoom filter is run at the server or at the client, because the volume of data transfer between server and the client is the same in both cases. When queries require subsampling, the placement of the zoom filter affects performance, since the volume of data sent from the server to the client decreases if the zoom filter is executed at the server. As is also seen from the figure, running the filters at the server (RDCZ-V) achieves better performance than running them at the client (R-DCZV) as would be expected since the client is a less powerful machine than the server.

Figure 8 shows query execution times when the server load changes. As is seen in the figure, as the server load increases (or the client becomes faster), running the filters on the client machine achieves better performance. The experimental results show that the decomposition of an application processing structure into filters and placement of the filters are important factors that affect overall performance. One of our long term goals in this work is to devise methodologies for a wide range of data-intensive applications for efficient restructuring of application processing structure into filters with the stream-based programming model, as well as developing cost models for filters to achieve efficient execution under changing processing requirements and system resource availability.

The query execution times for the original optimized Virtual Microscope server versus the prototype filter implementation using DataCutter are shown in Figure 9. In this experiment the entire dataset is loaded from HPSS and stored on a single local disk on a SUN Ultra 1 workstation since the original server is implemented to access datasets stored on disks. The loading of the dataset took 4750 seconds (1 hour 19 minutes). The original server is run as a single process, and all filters in the DataCutter implementation are executed on the same SUN workstation where the dataset is stored. In both cases the client is run on an-



(a) 1x server load (b) 2x server load (c) 4x server load

Figure 8: Execution time of queries under varying server load. 1× server denotes the case where the server is dedicated to running the filters, whereas 2× and 4× increased load implies server execution time doubles and quadruples that of the dedicated case. The subsampling factor is 8 in all cases.

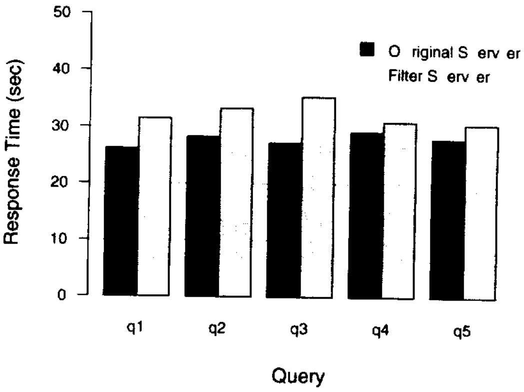


Figure 9: Query execution times for the original server and the server implemented using the DataCutter with filters. The subsampling factor is 8 in all queries.

other SUN Ultra 1 workstation connected to the local Ethernet segment. As is seen from the figure, the filter implementation does not introduce much overhead compared to the optimized original server. The percent increase in query execution time ranges from 6% to 30% across all queries. We should note that the timings do not include the time for loading the dataset, which can substantially increase for larger datasets and datasets stored in archival storage systems across a wide-area network. In addition, the use of filters in DataCutter takes advantage of pipelining and threaded execution, especially when the filters are run on multiprocessor architectures, resulting in overall higher performance.

6 Conclusions and Future Work

In this paper we have presented a middleware infrastructure, called DataCutter, to provide support for processing of large datasets stored in archival storage systems in a wide-area network environment. DataCutter provides support for subsetting of very large datasets through spatial range queries via hierarchical multi-level indexing, and user-defined aggregation and transformation on datasets via filters. We are in the process of developing standard interfaces and a client API for the DataCutter services. We also have several active projects that involve the use of DataCutter services and proxies. In a joint project with the data intensive computing environments group at the San Diego Supercomputing Center, we are interfacing DataCutter with the Storage Resource Broker (SRB) [21]. Our goal is to make it possible for SRB clients to perform spatial subsetting and data aggregation on distributed data collections accessible through the SRB. In a project with The University of Maryland Global Land Cover Facility [15], we are integrating DataCutter and the Active Data Repository (ADR) with the GLCF data servers to make it possible to visualize and generate data products from Landsat Thematic Mapper (TM) datasets stored in HPSS. We will extend our proxy infrastructure to cache data on disks as well as in memory, and integrate proxies with ADR so that clients can generate data products using ADR and data cached on the disks. DataCutter will provide support for accessing subsets of TM datasets from HPSS. We also plan to work on developing distributed stream-based algorithms via use of filters and carry out performance studies for a wider range of data intensive applications.

References

[1] A. Acharya, M. Uysal, and J. Saltz. Active disks: Programming model, algorithms and evaluation. In *Proceedings of the Eighth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS VIII)*, pages 81–91. ACM Press, Oct. 1998. ACM SIGPLAN Notices, Vol. 33, No. 11.

[2] A. Afework, M. D. Beynon, F. Bustamante, A. Demarzo, R. Ferreira, R. Miller, M. Silberman, J. Saltz, A. Sussman, and H. Tsang. Digital dynamic telepathology - the Virtual Microscope. In *Proceedings of the 1998 AMIA Annual Fall Symposium*. American Medical Informatics Association, Nov. 1998.

[3] Alexandria Digital Library. *http://alexandria.ucsb.edu/*.

[4] N. Beckmann, H.-P. Kriegel, R. Schneider, and B. Seeger. The R^*-tree: An efficient and robust access method for points and rectangles. In *Proceedings of the 1990 ACM-SIGMOD Conference*, pages 322–331, Atlantic City, NJ, May 1990.

[5] M. Beynon, A. Sussman, and J. Saltz. Performance impact of proxies in data intensive client-server applications. In *Proceedings of the 1999 International Conference on Supercomputing*. ACM Press, June 1999.

[6] C. Chang, R. Ferreira, A. Sussman, and J. Saltz. Infrastructure for building parallel database systems for multi-dimensional data. In *Proceedings of the Second Merged IPPS/SPDP Symposiums*. IEEE Computer Society Press, Apr. 1999.

[7] C. Chang, B. Moon, A. Acharya, C. Shock, A. Sussman, and J. Saltz. Titan: A high performance remote-sensing database. In *Proceedings of the 1997 International Conference on Data Engineering*, pages 375–384. IEEE Computer Society Press, Apr. 1997.

[8] A. Chervenak, I. Foster, C. Kesselman, C. Salisbury, and S. Tuecke. The Data Grid: Towards an architecture for the distributed management and analysis of large scientific datasets. *http://www.globus.org/*, 1999.

[9] A. Choudhary, R. Bordawekar, M. Harry, R. Krishnaiyer, R. Ponnusamy, T. Singh, and R. Thakur. PASSION: Parallel and scalable software for input-output. Technical Report SCCS-636, NPAC, Sept. 1994. Also available as CRPC Report CRPC-TR94483.

[10] I. Foster. The Beta Grid: A national infrastructure for computer systems research. In *Net-Store'99*, 1999.

[11] I. Foster, D. K. Jr., R. Krishnaiyer, and J. Mogill. Remote I/O: fast access to distant storage. In *Fifth Workshop on I/O in Parallel and Distributed Systems (IOPADS)*, pages 14–25. ACM Press, 1997.

[12] I. Foster and C. Kesselman. Globus: A metacomputing infrastructure toolkit. *International Journal of Supercomputer Applications and High Performance Computing*, 11(2):115–128, 1997.

[13] I. Foster and C. Kesselman. *The GRID: Blueprint for a New Computing Infrastructure.* Morgan-Kaufmann, 1999.

[14] Geographic Information Systems. *http://www.usgs.gov/research/gis/title.html.*

[15] The University of Maryland Global Land Cover Facility. *http://glcf.umiacs.umd.edu.*

[16] Grid Forum. Birds-of-a-Feather Session, SC99, Nov 1999.

[17] The High Performance Storage System (HPSS). *http://www.sdsc.edu/hpss/hpss1.html.*

[18] T. Johnson. An architecture for using tertiary storage in a data warehouse. In *the Sixth NASA Goddard Space Flight Center Conference on Mass Storage Systems and Technologies, Fifteenth IEEE Symposium on Mass Storage Systems*, 1998.

[19] W. E. Johnston and B. Tierney. A distributed parallel storage architecture and its potential application within EOSDIS. In *NASA Mass Storage Symposium*, Mar. 1995.

[20] T. M. Kurc, A. Sussman, and J. Saltz. Coupling multiple simulations via a high performance customizable database system. In *Proceedings of the Ninth SIAM Conference on Parallel Processing for Scientific Computing*. SIAM, Mar. 1999.

[21] SRB: The Storage Resource Broker. *http://www.npaci.edu/DICE/SRB/index.html.*

[22] N. Talagala, S. Asami, and D. Patterson. The Berkeley-San Francisco fine arts image database. In *the Sixth NASA Goddard Space Flight Center Conference on Mass Storage Systems and Technologies, Fifteenth IEEE Symposium on Mass Storage Systems*, 1998.

[23] M. Teller and P. Rutherford. Petabyte file systems based on tertiary storage. In *the Sixth NASA Goddard Space Flight Center Conference on Mass Storage Systems and Technologies, Fifteenth IEEE Symposium on Mass Storage Systems*, 1998.

[24] M. Uysal. *Programming Model, Algorithms, and Performance Evaluation of Active Disks.* PhD thesis, Department of Computer Science, University of Maryland, College Park, 1999.

Towards Mass Storage Systems with Object Granularity

Koen Holtman
CERN – EP division
CH - 1211 Geneva 23, Switzerland
Koen.Holtman@cern.ch

Peter van der Stok
Eindhoven University of Technology
Postbus 513, 5600 MB Eindhoven,
The Netherlands
wsstok@win.tue.nl

Ian Willers
CERN – EP division
CH - 1211 Geneva 23, Switzerland
Ian.Willers@cern.ch

Abstract

Many applications, that need mass storage, manipulate data sets with KB – MB size objects. In contrast, mass storage devices work most efficiently for the storage and transfer of large files in the MB – GB range. Reflecting these device characteristics, mass storage systems typically have a file level granularity. To overcome the impedance mismatch between small objects and large files, we propose a move towards mass storage systems with object granularity. With an object granularity system, the application programmer stores and retrieves objects rather than files. The system internally maps and re-maps these objects into files. The system can adapt to changing object access patterns by re-mapping objects. This allows the application to be more efficient than if it were built on top of a traditional file granularity mass storage system, employing a fixed mapping of objects to files.

In this paper we report on investigations on the potential benefits of object granularity systems. We present an architecture that incorporates solutions to the scalability and fragmentation problems associated with object granularity.

1 Introduction

For some applications, the application dataset is so large that data storage on tape is an economic necessity. Examples where datasets can be in the Terabyte scale are high energy physics data analysis and satellite image analysis. Such applications can be built on top of a mass storage system, which controls data movement between tape storage and a disk farm that serves both as a staging pool and as a cache, this disk farm is called the *disk cache* below.

Tape drives work efficiently if the data on them are accessed in terms of MB – GB size files. Reflecting these hardware characteristics, mass storage systems generally have a file granularity, with the expectation of managing large files. Conversely, in many mass storage applications, the application-level data consist of objects with sizes in the 1 KB – 1 MB range. The application designer must map the application-level objects to files on tape, with every file containing many objects. This mapping is usually done when the mass storage system is being filled, and no re-mapping is done over the lifetime of the dataset. Though such a fixed mapping to large files allows the mass storage system to function efficiently, it can cause application-level inefficiencies. The inefficiency will be especially high if the application often needs a small subset of the objects in a file.

To overcome the impedance mismatch between small application level objects and the large files desired on tape, we propose a move towards mass storage systems with object granularity, that hide the underlying files.

In a mass storage system with object granularity, the application programmer stores and retrieves objects rather than files. Caching and migration inside the system are also object-based. The system internally maps and re-maps objects to files. By re-mapping objects, the system can adapt to changing application-level object access patterns. This allows the application to be more efficient than if it were built on top of a mass storage system with file granularity, employing a fixed mapping of objects to files.

While the potential benefits of an object granularity system are clear, so are its potential problems. The size of the indexing and scheduling tasks associated with managing objects, rather than files, will be some orders of magnitude larger. Also, there is an obvious danger of data fragmentation on tape and in the disk cache.

In this paper we report on investigations on the potential benefits of object granularity systems. We present an architecture that incorporates solutions to the scalability and fragmentation problems associated with object granularity. This architecture incorporates a commercial object database, Objectivity/DB [1] and a traditional file granularity mass storage system (for example HPSS). By using these standard components, the implementation cost of our object granularity mass storage system is kept low.

We show that object granularity systems outperform file granularity systems for applications in which the following conditions are met.

1. **Sparse access condition**: The application data access patterns have to be so diverse or unpredictable that a fixed mapping of objects to files will lead to inefficiencies. We quantify this condition as follows. Take the initial, fixed mapping to files as created (and optimised) by the application designer. Any query will 'hit' a certain number of files in this initial set. Now consider the objects, in these hit files, that are actually needed by the query. These objects should make up 30% or less of all objects in the files, on average, for the sparse access condition to hold.
2. **Repetitive access condition**: The application data access patterns should also be such that object (sub)sets selected at the application level are read not once, but a few times over a period of time.

Our work was driven by the problem of Petabyte-scale data analysis in the next-generation high energy physics experiments at CERN (see for example [2]). This is one application area where the above two conditions hold.

2 Overview of the architecture

We developed an architecture for an object granularity mass storage system that contains solutions to the scalability and fragmentation problems mentioned above. This architecture should be seen as an existence proof for a system with object granularity. Systems which employ different solutions to the object granularity problems may also be feasible. We have investigated some alternatives, but do not claim to have surveyed all possible solutions.

2.1 Software components

Our work is part of a larger research project, aimed at exploring database technology options for the storage and analysis of massive high energy physics datasets [3]. Our architecture is based on software solutions being pursued in this project [4]. We use the Objectivity/DB object database product [1], which is interfaced [5] to a generic file granularity mass storage system, like HPSS. We develop new software components that 1) add an abstraction layer, which provides object granularity mass storage, on top of the object database, and 2) control file movement between disk and tape, and the management (re-mapping, deletion) of data in the disk cache. In line with the work in [5], HPSS only acts as a file stager, its disk pool management functions are not used. As such, the choice for HPSS as a software component is not critical, and it could be interchanged with another file granularity mass storage system.

2.2 Filling the system with objects

Our object granularity mass storage system provides an 'append only' storage model, in which new objects can be added at any time, but in which objects become read-only once added. The application programmer fills the mass storage system by supplying *chunks* to it. A chunk is a set of objects (typical size 10 MB - 10 GB), which is initially mapped, by the system, to a single file on tape. This chunk model gives the application programmer a degree of control over the initial mapping to files on tape that is similar to that found with a traditional file granularity mass storage system. We found that retaining such control is important. Object re-mapping can in principle compensate fully for a bad or random initial mapping of objects to files. But performing such a re-mapping will take significant system resources. It is better to save these resources in advance by allowing the application programmer to encode advance knowledge about access patterns into the chunks.

The mapping of an object *to a chunk* is retained throughout the lifetime of the object. During this lifetime, the object can be (re)mapped to many different files.

2.3 Object addressing

Once stored, an object is uniquely identified by its *chunk identifier* and its *sequence number* inside the chunk. Sequence numbers run from 1 to n for a chunk with n objects, and reflect the object storage order inside the original chunk file, which was determined by the application programmer. Figure 1 shows a visual example of object addressing.

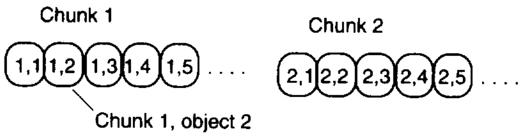


Figure 1: Example of logical object addressing

Our system maps the objects in chunks to physical files. Figure 2 shows an example of such a mapping. The system allows many files to be present for any chunk. Every file present for a chunk holds a subset of the objects in that chunk. In a running system, the number of files per chunk typically ranges from 1 to 20, depending on the access patterns to the objects in the chunk. The subsets of objects held by the different files may overlap, and generally do overlap, so that some objects in a chunk are physically present in multiple files.

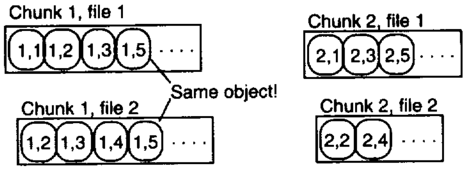


Figure 2: Example of physical mapping of objects to files

Files never mix objects from two or more chunks. This strict chunk-level separation makes indexing and scheduling problems much more manageable. The system does not maintain a single global index for looking up in what files an object is contained. Instead, there is a file-level index for each file, which can be used by sub-queries to read objects from the file, and by schedulers to quickly determine the exact set of objects in a file. All file-level indices are kept on secondary storage.

Our system does not require that the above 'files' are actual files managed by a filesystem. In our prototyping efforts using the software components described in section 2.1, the 'files' are actually (sets of) ODMG containers in the Objectivity/DB database.

2.4 Object access

The application programmer can access stored objects by executing a *query* against the store. A query specifies a set S of object identifiers, with the intention that all these objects must be visited, and a *query function*, which is a piece of executable C++ code (typically a loadable shared library). To execute the query, the system first computes the set C of all chunks that contain one or more objects identified in S. For every chunk, the system runs a *sub-query* over the objects in this chunk. This sub-query iterates through all objects in S which are in its chunk. For every object, the programmer-supplied query function is called. The query function is handed the object identifier and a reference to an in-memory copy of the object. The application programmer can optionally supply code that is to be executed at the start and the end of the query and of any sub-query.

Iteration by a sub-query always happens in the order of the sequence numbering of the objects in the chunk, this order was determined by the application programmer when the chunk was added to the system. The fixed iteration order allows the system to ensure fast data access and to prevent fragmentation.

The scheduling of sub-queries is outside the control of the application programmer: this is done by the system to ensure that the sub-query is synchronised with any necessary file staging operations preceding it. Many queries, and their sub-queries, can run in parallel. In applications with highly CPU-intensive application code, as found in high energy physics, tens to hundreds of sub-queries may be running in parallel on a CPU farm.

When a sub-query is scheduled to start its iteration, it first determines which files on disk should be accessed in order to read all objects that it needs to visit. To choose this set of files, the sub-query compares the set of objects it needs to the sets of objects present in the different files of its chunk. These set comparisons are implemented as comparisons between sets of object identifiers, the object identifiers of all objects in a file are obtained by accessing the file-level index of that file. Sometimes, because of an overlap in the object sets contained in the files, there are many options in choosing a set of files which together contain the needed objects. If there is a choice, the sub-query will always choose the set of files with the smallest sum of file sizes. This choice minimises any disk efficiency losses because of sparse reading when accessing the files, and, more importantly, it yields the best possible input for the cache replacement algorithm (section 4.3), in which file access statistics play an important role. When the sub-query comes to visiting a particular object that is contained in several of the chosen files, it will read that object from the smallest of these files. The choice for the smallest file is immaterial to the cache replacement algorithm, which works with file level access statistics and ignores object-level details. The smallest file is chosen based on the assumption that this usually minimises the overall sparseness of reading, so optimising the I/O performance.

2.5 Re-mapping of objects

Many mechanisms are possible for the re-mapping of objects to files. We chose a mechanism based on an object *copy*, rather than an object *move* operation. Using a copy operation has some advantages: in particular, a copy operation does not affect concurrently running sub-queries accessing some of the objects being copied. The use of move operations would require a strong synchronisation between these sub-queries: this makes the implementation more complex and might be a source of performance loss, caused by lower concurrency and more locking traffic. A disadvantage of copying is that the resulting duplication implies less efficient use of scarce storage space, in particular in the disk cache.

Object re-mapping is done during sub-query execution, while the sub-query iterates through its objects. Re-mapping is always from a file (staged) on disk to another file on disk. In the simplest case, shown in figure 3, some objects from a single existing file are copied into a new smaller file. The original file can then be deleted. The end result of such a re-mapping and deletion is that disk space previously occupied by the cold (non-queried, white) objects is freed, while all hot (queried, grey) objects are still present. Thus, we can effectively cache more hot objects on disk.

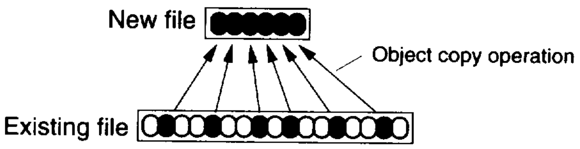


Figure 3: Simplest case of object re-mapping: the (grey) objects read by a sub-query are copied into a new file

Re-mapping decisions are based on the 'densities' of the hot objects in the existing files. For example, if an existing file contains 95% hot and 5% cold objects for a sub-query, then the hot objects in this file will never be copied to a new file, as the storage efficiency in the existing file is already near-optimal. The densities are determined at the start of the sub-query, using the file indices. From this information, a re-mapping function is computed, which can be used to quickly decide, for each object accessed by the sub-query, whether this object should be re-mapped.

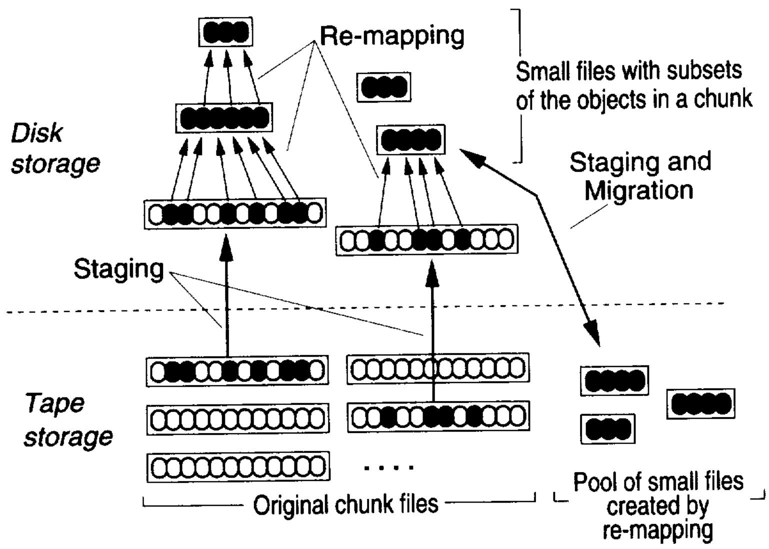

Re-mapping
Small files with subsets of the objects in a chunk
Disk storage
Staging and Migration
Staging
Tape storage
Original chunk files
Pool of small files created by re-mapping

Figure 4: Overview of the complete storage architecture

Figure 3 shows the simplest case of re-mapping. In more complicated cases, the copying of some objects into the new file can be suppressed because they are already present in another small file. More details on the optimisation problems involved can be found in [6].

2.6 Complete storage architecture

Figure 4 shows the complete storage architecture of our system, with both the tape and disk storage layers. The tape contains two different pools of files: 1) the original chunk files and 2) smaller files created by re-mapping. The original chunk files are created on tape when the application programmer adds chunks to the system (section 2.2). These files are never updated or deleted. As the original chunk files are always retained, all other files on disk and tape can be deleted, whenever more space is needed, without running the risk of loosing objects. The smaller files on tape are all first created on disk, by re-mapping operations, and later migrated down to tape.

As shown in figure 4, re-mapping operations can be done recursively: if a new query yields a smaller set of hot objects, a still-smaller file can be made, and the larger file can be deleted.

3 Bursty sequential reading

As noted in section 2.4, the high energy physics requirement of using highly CPU-intensive query function code implies that the system should allow for tens to hundreds of sub-queries running concurrently on a CPU farm, all accessing files containing objects. Re-mapping operations may scatter the objects needed by a sub-query over many files. In tests with our system, a sub-query generally reads objects only from a single file, but sometimes from a few files, and in extreme cases from up to 10 files.

Taking everything together, in a running system there could be concurrent access to a few hundreds of files on disk. We use an optimisation we call *bursty sequential reading* to guarantee a good parallel I/O performance. This optimisation works at the file level, and has two parts. First, the objects in a file are always read sequentially, possibly sparsely, in the order in which they are stored in the file. This sequential reading is easy to achieve: in the original chunk files the object ordering is by definition the same as the sub-query iteration order, and all re-mapping operations preserve the object ordering. The second part of the optimisation is that, in every file, the sub-query reads the needed objects in bursts of about 1 MB. The objects read in these bursts are buffered in memory until they are delivered to the query function. The bursty sequential reading optimisation, which we implemented on top of the Objectivity/DB database C++ binding, is sufficient to achieve good parallel I/O performance. Disk throughputs are near the maximum possible throughputs achievable with pure sequential reading, as specified by the disk manufacturers. It should be noted that the Objectivity/DB architecture, which employs no central database engine, also contributes to the good I/O scalability we found.

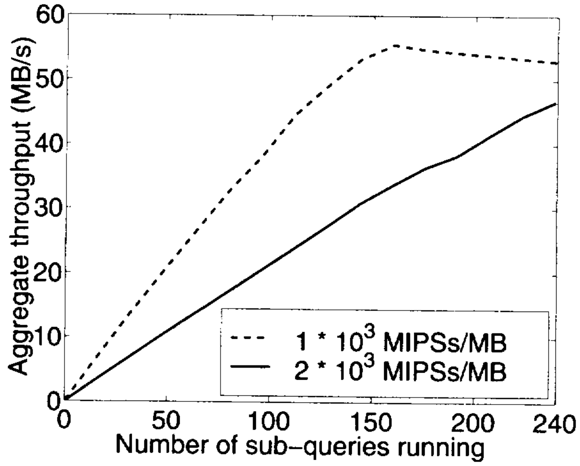
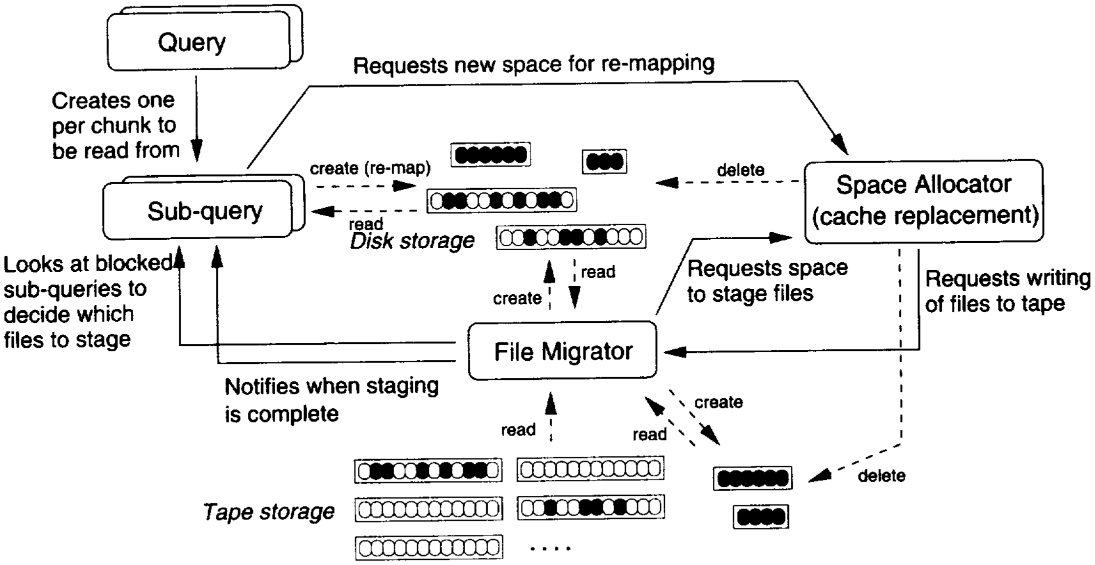


Figure 5: I/O scalability with bursty sequential reading

Figure 5 shows the results of some parallel I/O tests of our sub-queries with bursty sequential reading. The tests were performed on a 256-CPU HP Exemplar supercomputer. We ran up to 240 sub-queries concurrently. Each sub-query uses almost all of the power of a single CPU to execute application code. The two curves are for application code spending $1 * 10^3$ MIPSs per MB read, and $2 * 10^3$ MIPSs per MB read. Each sub-query reads its objects interleaved from 3 files, while also re-mapping (copying) every tenth object into a new file. Every sub-query is executed on its own in a private UNIX process. Multi-threading inside UNIX processes was not used, though it is in principle supported by the database and OS software used.

Both curves in figure 5 show very smooth I/O scaling, indicating efficient use of the available disk resources. In the lower curve, the system remains CPU bound. In the upper curve, with less computation per MB read, the system becomes I/O bound above 160 concurrently running sub-queries: at that point, the available I/O resources (16 disks in 4 striped file-systems) are saturated.

Note that these good scalability results were achieved with only a single optimisation layer on top of a commercial object database product. A special purpose parallel I/O library was not needed.



Query
Creates one per chunk to be read from
Requests new space for re-mapping
Sub-query
create (re-map)
read
Disk storage
delete
Space Allocator (cache replacement)
Looks at blocked sub-queries to decide which files to stage
read
create
Requests space to stage files
Requests writing of files to tape
File Migrator
Notifies when staging is complete
read
create
read
delete
Tape storage
....

Figure 6: System components and their interactions

4 Components and policies of the architecture

Figure 6 shows the different active components (rounded rectangles) in our architecture, and their responsibilities and interactions with each other. The components invoke scheduling algorithms (not shown) to make optimisation and scheduling decisions. The query component, and its production of sub-queries, have already been discussed in section 2.4. Below, the remaining components are discussed, along with some of the scheduling algorithms.

4.1 Sub-query

As already discussed in section 2.4, a sub-query performs the reading of selected objects in a single chunk, and executes the query function against them. The sub-query can also perform a re-mapping operation when it is running. When started by its parent query, a sub-query will immediately examine the (indices of) the files on disk to determine if all objects it needs are present on disk. If not, the sub-query 'blocks', it will suspend its execution to wait until all objects are present. It is the responsibility of the file migrator (see below) to ensure that blocked sub-queries eventually become un-blocked.

When a sub-query, possibly after having been blocked, finds all needed objects present on disk, it computes from which files to read these objects, and whether to do any re-mapping. After locking these files against deletion by cache replacement, the sub-query requests permission from a central scheduler (not shown in figure 6) to start reading objects. The central scheduler ensures that not too many sub-queries will do intensive disk I/O at the same time. For example, on the system configuration in section 3, a good limit would be to bound concurrency to some 400 sub-queries. When permission to read is obtained, the sub-query will first request some free disk space if re-mapping is to be done. Then, the sub-query iterates over the needed objects in its chunk. Objects are read from existing files, fed to the query function, and possibly written to a new file in re-mapping.

4.2 File migrator

The file migrator manages the tape drive(s) in the system, and migrates files disk and tape.

The file migrator examines all blocked sub-queries to decide which file to stage next. Many hundreds of sub-queries may be blocked at the same time. Sometimes, many sub-queries (of different queries) are all blocked, waiting for the staging of objects from the same chunk. The file migrator partitions the blocked sub-queries into clusters. Every cluster is a group of blocked sub-queries waiting for objects in the same chunk. For every cluster, the file migrator identifies a single file on tape, whose staging would allow all sub-queries in the cluster to un-block. This pooling of tape requests from different queries is known as *query batching*, and it can lead to dramatic savings [7], especially for workloads with many concurrent large queries.

In any tape-based system, it is important to minimise the randomness of tape I/O as much as possible, because tape mounts and seeks are expensive operations. After an investigation of alternatives, we chose the following policy that aggressively minimises mounts and seeks. The policy cycles over all tapes in the system in a fixed order. When the next tape is reached, and a tape drive becomes available for reading, the file migrator looks if any of the files needed by the current clusters are on this tape. If so, the tape is mounted. Then, any needed files are staged in the order of their position on tape. This results in a sequential pattern of seeking and reading on the tape. When the last file has been staged, the tape is rewound and unmounted. The fixed cycling order ensures that sub-queries are never blocked indefinitely.

4.3 Space allocator

The space allocator manages two pools of files: the files on disk and the small pool of files created by re-mapping on tape. Both these pools can be seen as caches, and so are managed by cache replacement policies.

For the pool of files on disk, the cache replacement policy has to achieve some conflicting aims. Firstly, a recently used file of course has to be retained as long as possible. But secondly, a file from which objects were recently re-mapped should be deleted as quickly as possible, so that the goal of the re-mapping operation, creating a tighter packing of hot objects in the cache, is actually achieved. Thirdly, if a query with a size many times that of the disk cache size is executed, no attempt at caching these files on disk should be made, but they should be deleted as quickly as possible, to maximise the available cache space for smaller queries. A special policy called 'usage based cooling' was developed to reconcile these conflicting aims. Because of space limitations, we refer the reader to [6] for a detailed discussion of this policy.

For the pool of smaller files on tape, the following management policy is used. A set of tapes is reserved exclusively to hold these small files. One tape at a time is filled, files are written on the tape sequentially. When all tapes in the pool are full, the oldest tape is recycled: all files on it are deleted and writing starts again from the front of the tape. The above scheme amounts to a 'least recently created' cache replacement policy. Of course, a policy closer to 'least recently used' would potentially be more effective at maintaining a set of useful files on tape, if a way could be found to keep the associated free space fragmentation on tape in check. To investigate the potential benefits of other replacement policies on tape, we used simulation to determine the performance of 'least recently used' replacement under the (unrealistic) assumption that there is no performance loss due to fragmentation. We found that a 'least recently used' policy simulated under this assumption outperformed 'least recently created' with factors of 1.0 (no improvement) up to 1.2 over our range of test workload parameters. From this small improvement factor we conclude that our simple 'least recently created' tape replacement policy is an appropriate choice. A better policy may be possible, but it is unlikely to be better by more than a factor 1.2.

4.4 Writing of small files to tape

Chunk reclustering is done by copying some of the small files on disk, which were produced by re-mapping, to tape. When the space allocator determines that a file is soon to be replaced (deleted) from the disk cache, it invokes an algorithm to decide whether the file is to be copied to tape. If the file is to be copied, the space allocator includes this file in a batch of write requests to the file migrator, and will then hold off deleting the file from disk until it has been copied to tape.

There is no obvious method of deciding whether or not a file should be copied to tape before deletion on disk. Obviously, only the 'best' files should be copied to tape, but when the space allocator offers a file up for consideration, it has already decided itself that this is one of the 'worst' files it has on disk! We have tried to find a good selection method as follows: we simulated a system in which (unrealistically) all files would be copied, at zero cost, to a very large tape pool before deletion from disk. Then, we examined the files that were actually staged back onto disk in the simulation, to find some identifying characteristics. However, we failed to find a good predictive identifying characteristic that could be used in a selection heuristic. In the end, we used a simple heuristic based on the observation that very large files should obviously not be written back to tape, because the initial chunk files already present would allow for the staging of large object sets at similar costs. We introduced a size cut-off: all files smaller than 20% of the chunk size are selected for copying to tape. This value of 20% was determined in a tuning exercise. We found that 40%, for example, works almost as well. We tried some more refined heuristics but found no heuristic that was noticeably better.

5 Benefits of object granularity

To assess the benefits of object granularity, we used simulation over a large workload parameter space. In these simulations we compared the performance of our object granularity mass storage system with that of a file granularity system, over a range of application workloads and hardware parameters. The chunks of the object granularity system appeared as initial files on tape in the file granularity system. The workloads satisfied the sparse and repetitive access conditions formulated at the end of section 1. The workloads are multiuser workloads, with query sizes ranging from 0.03% to 6% of the complete dataset size, with an average of 0.34%. In our simulations, the disk cache size ranged from 2% to 20% of the

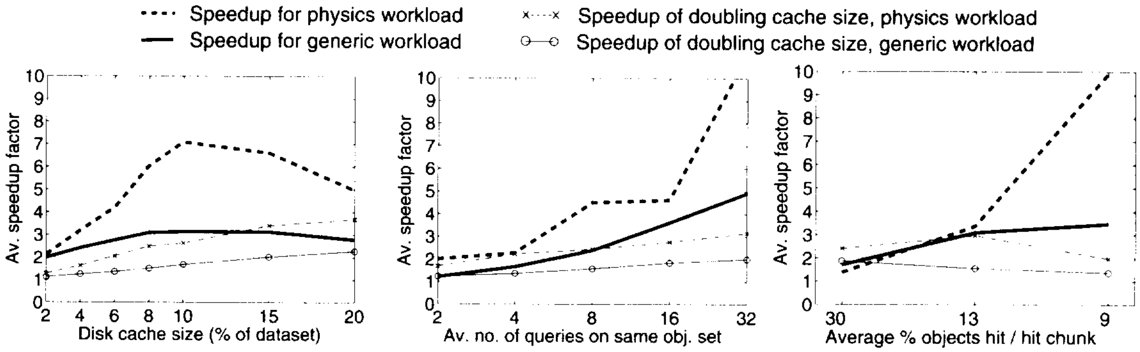


Figure 7: Dependence of different speedup factors on several parameters. Every speedup factor shown in a graph is the average over all combinations of the parameter values on the x-axes of the other two graphs. The rightmost graph shows the dependency on the sparseness of access, on the x-axis is the measure defined in section 1, the average percentage of objects which are needed by a query in every chunk that the query hits.

dataset size. For details on the simulation workloads used, we refer to [6].[1]

To assess the benefits, we determined the speedup factor of our object granularity system over the normal file granularity system. We found that the speedup factor is dependent on many workload and hardware parameters. We found speedup factors from 1 (no speedup) to 52. As expected, speedups are higher when workloads more often access the same sets of objects. For workloads with a high repetition factor, i.e. if on average at least 4 queries visit the same object set, speedups are typically a factor 2 – 4. Other forms of repetitiveness, for example if new queries access subsets of the object sets visited by older queries, also improve the speedup.

Again as expected, following the reasoning behind the sparse access condition in section 1, the speedup over file granularity systems is higher if access to the chunks is more sparse. Speedups in excess of 10 are found if, on average over all queries in the workload and all chunks in the system, a query 'hits' less than 10% of the objects in every chunk that it hits.

5.1 Dependence on workload parameters

Figure 7 illustrates the dependence of the speedup factors on various system and workload parameters. All these graphs plot averages of speedup factors over parts of the parameter space, and so de-emphasise some of the more extreme cases. Curves for *physics* and *generic* workloads [6] are shown: in a physics workload, new queries access subsets of the object sets visited by older queries, this corresponds to what happens in high energy physics data analysis. In a generic workload the object sets selected by (sequences of) queries are completely independent. For comparison, figure 7 also shows speedup curves for doubling

[1] The workloads used for the tests discussed below differ from those in [6] only in that a new value, 9%, was added to the range of the parameter 'average percentage of objects needed in a chunk that is hit by a query'.

the disk cache size in the baseline system.

5.2 Benefits of keeping a pool of small files on tape

We found that the re-mapping of files on disk, followed by the deletion of original files on disk, contributed most to the speedup, by improving the storage efficiency in the disk cache.

The speedup contribution of having a pool of small files on tape was lower. The small files on tape do save tape resources, because staging in a smaller file can often substitute for staging in the much larger original chunk file. However, the resources needed to write the small files to tape in the first place are considerable. Typically half to all of the time saved by reading small files in stead of larger ones is spent in writing the small files. Figure 8 shows simulation results for our system with and without the optimisation of keeping a pool of small files on tape. The speedup contribution of having small files on tape was often only a factor 1.2 or lower. We found that this speedup depended strongly on the size of the disk cache: the smaller the disk cache, the larger the gains of maintaining a pool of small files on tape. For disk cache sizes of 4% or less than the application data set size, we sometimes found worthwhile speedups factors, from 1.5 to 2.1, depending on workload parameters.

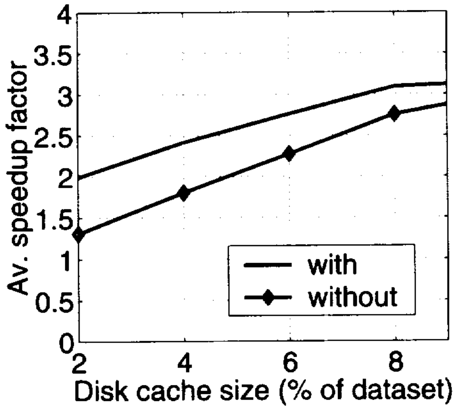


Figure 8: Average speedups with and without the optimisation of keeping a pool of small files on tape, for generic workloads

6 Towards object granularity

The architecture presented in this paper is a viable one, but not the only possible solution or even provably the best solution for an object granularity mass storage system. It therefore makes sense to review the design process that led towards the architecture in this paper, so as to distinguish between largely arbitrary design choices on the one hand and some forced moves on the other hand.

The challenge in moving towards a system with object granularity is to make the grain size smaller, so as to increase chances for optimisation, without making the grain size so small that the system design collapses under the increased complexity, or the I/O performance breaks down due to increased fragmentation. Our solution centres around introducing an intermediate level of granularity. Our system has objects, files, and chunks: three levels compared to the two, objects and files, discernible in applications that use file granularity systems. We perform tasks like migration and cache replacement at the intermediate file level, rather than the object level. This way, many of the drawbacks of true object granularity, like the risks of fragmentation and a too heavy load on the scheduling algorithms, can be avoided. Of course, it was not clear from the start of the design exercise whether the supposed benefits of object granularity, opportunities for successful optimisations, could be preserved when taking this route!

Working from the decision to have an intermediate file level, it is relatively easy to draw a first picture of data movement and layout, like the one in figure 4. Based on this picture one can make a list of all processes that have to be steered by the system implementation. Most these processes interact with each other, and this makes the creation of an optimised design very difficult. To make any progress at all, we decided to take the risky approach of completely disregarding these interactions at first, and decomposing the system into a few weakly interacting active components and scheduling algorithms. The creation of the individual components would then be followed by a 'big bang' integration step. Of course, this 'big bang' design method has a deservedly bad reputation. Before embarking on it, we spent considerable time in searching for more stepwise methods (without much success), analysing the associated risks, and developing techniques to mitigate these risks. We mitigated risks by making robustness of individual system elements an important design goal. We designed the active components and scheduling algorithms so that they would keep working, and ensure some degree of progress, no matter how bad the decisions of other scheduling algorithms would be.

For the 'big bang' integration phase, we used a simulation-driven approach. First, the tuning parameters in all scheduling algorithms were set to some plausible initial values, this way we obtained a first integrated, and more or less working, version of the whole system. Then, using simulations with likely workloads, we tuned the individual scheduling algorithms to globally deliver good performance. Beyond the initial 'big bang' point the integration phase was therefore an iterative one, with many test-adjust cycles. The integration phase was supported by a simulation framework that allowed parameters and algorithm details to be altered quickly. In exploring the workload and tuning parameter space to investigate design options, we used about 600 CPU hours running some 25000 simulations.

An analysis of the above design process shows some obvious opportunities for designing an object granularity system that outperforms our current one. The exact implications of many small decisions in the system design are unknown, so reversing these decisions may lead to a better system. Also, the grain size at the file level is fairly coarse, typically there are at most some 20 files per chunk, and ways could be sought to obtain a finer grain size, with possibly higher payoffs. Most importantly, given the knowledge gained in the creation and evaluation of the current architecture, the creation of a new design that more closely integrates the different scheduling tasks may now be a tractable problem.

7 Related work

To our knowledge, there is no other work which takes true object granularity in caching and staging as a goal, and develops and evaluates an architecture to deal with the associated scalability and fragmentation problems. Our previous work [8] uses techniques similar to re-mapping in a disk-only reclustering system. In fact, the system developed in [8] served as a proof of concept, which gave us confidence that re-mapping could feasibly be introduced in a mass storage system with both disk and tape. Our work [6] explores re-mapping in the disk cache, but not keeping a set of smaller files on tape to achieve some kind of object granularity in staging operations. Our architecture builds on experience from existing mass storage systems [7] [9] [10], especially with respect to cache replacement and staging policies.

Many systems cache data at a finer granularity when it moves upwards in the storage hierarchy, see for example [9]. At least one existing mass storage product [11] structures data into small units (atoms, like our objects), and allows the application programmer to request (sets of) such data units, rather than complete files. To our knowledge, this product uses a caching granularity below the staging granularity, but it does not go down to the 'atomic' level in its caching mechanisms.

Many tape based data analysis systems in use in science allow users or administrators to optimise performance through the creation of new, smaller datasets which contain some selected objects from the complete dataset. Queries can then be redirected to these new datasets, or are redirected automatically. Such strategies are known as 'view materialisation', or, in high energy physics, the creation of 'data summary tapes'. View materialisation strategies are similar in intent and effect to our two new optimisations. This can lead to similarities at the architectural level, see for example [12] for a view materialisation system that, though not targeted at tape based data, has some patterns in common with our architecture. We know of no existing tape based data analysis systems in which creation of such smaller datasets, the picking of objects for them, and their eventual deletion have been automated to a large extent.

8 Conclusions

For the foreseeable future, the use of tape storage remains the only cost-effective option for the massive datasets used in a number of scientific endeavours. CERN is actively pursuing research into data management options to address the needs of its future physics experiments.

We propose a move towards mass storage systems with object granularity, to overcome the impedance mismatch between small application level objects and the large files desired on tape. Such systems hide the mapping of objects to files from the application programmer, and dynamically re-map objects to files in order to improve application performance.

We have identified two conditions, the sparse access condition and the repetitive access condition, which an application must fulfil to make the use of an object granularity mass storage system underneath the application attractive.

We have investigated the potential benefits of object granularity mass storage systems by developing a viable architecture for such a system. The architecture resolves scalability and fragmentation problems by managing files containing (sub)sets of objects, rather than individual objects. The architecture incorporates a commercial object database and a normal file granularity mass storage system. We have evaluated the architecture through implementation and simulation studies. We found speedup factors from 1 to 52. The speedup gains of our object granularity system are mostly due to the increased cache efficiency on disk, which is achieved through object re-mapping. The storage of files with re-mapped objects on tape seems less attractive as an optimisation, except when disk space is very small compared to tape space or average query size.

The architecture is shown to be a viable one, but probably not an optimal one. We have identified some research opportunities that could lead to improvements over the current architecture. The results obtained here will serve as a basis for future R&D at CERN.

References

[1] Objectivity/DB. Vendor homepage: http://www.objy.com/

[2] CMS Computing Technical Proposal. CERN/LHCC 96-45, CMS collaboration, 19 December 1996.

[3] The RD45 project (A Persistent Storage Manager for HEP). http://wwwinfo.cern.ch/asd/rd45/

[4] J. Shiers. Massive-Scale Data Management using Standards-Based Solutions. 16th IEEE Symposium on Mass Storage Systems, San Diego, California, USA, 1999.

[5] A. Hanushevsky, M. Nowak. Pursuit of a Scalable High Performance Multi-Petabyte Database. 16th IEEE Symposium on Mass Storage Systems, San Diego, California, USA, 1999.

[6] K. Holtman, P. van der Stok, I. Willers. A Cache Filtering Optimisation for Queries to Massive Datasets on Tertiary Storage. Proc. of DOLAP'99, Kansas City, USA, November 6, 1999.

[7] J. Yu, D. DeWitt, Query pre-execution and batching in Paradise: A two-pronged approach to the efficient processing of queries in tape-resident data sets. Proc. of 9th Int. Conf. on Scientific and Statistical Database Management, Olympia, Washington (1997).

[8] K. Holtman, P. van der Stok, I. Willers. Automatic Reclustering of Objects in Very Large Databases for High Energy Physics, Proc. of IDEAS '98, Cardiff, UK, p. 132-140, IEEE 1998.

[9] R. Grossman, D. Hanley, X. Qin, Caching and Migration for Multilevel Persistent Object Stores. Proc. of 14th IEEE Symposium on Mass Storage Systems 127-135 (1995).

[10] S. Sarawagi, Query Processing in Tertiary Memory Databases, Proc. of 21st VLDB Conference, Zurich, Switzerland, 1995, p. 585–596.

[11] StorHouse, the Atomic Data Store. Vendor homepage: http://www.filetek.com/

[12] Y. Kotidis, N. Roussopoulos, DynaMat: A Dynamic View Management System for Data Warehouses. Proc. of ACM SIGMOD, May 31 - June 3, 1999, Philadephia, USA.

The InTENsity PowerWall:
A Case Study for a Shared File System Testing Framework

Alex Elder, Thomas M. Ruwart, Benjamin D. Allen,
Angela Bartow, Sarah E. Anderson, David H. Porter
Laboratory for Computational Science and Engineering
University of Minnesota
Minneapolis, MN 55455
{elder,tmr,benjamin,bartow,sea,dhp}@lcse.umn.edu
tel +1-612-626-0059
fax +1-612-626-0030

Abstract

The InTENsity PowerWall is a display system used for high-resolution visualization of very large volumetric data sets. The display is linked to two separate computing environments consisting of more than a dozen computer systems. Linking these systems is a common shared storage subsystem that allows a great deal of flexibility in the way visualization data can be generated and displayed. These visualization applications demand very high bandwidth performance from the storage subsystem and associated file system.

The InTENsity PowerWall system presents a real-world application environment in which to apply a distributed performance testing framework under development at the Laboratory for Computational Science and Engineering at the University of Minnesota. This testing framework allows us to perform focused, coordinated performance testing of the hardware and software components of storage area networks and shared file systems.[2] We use this framework to evaluate various performance characteristics of the PowerWall system's storage area network. We describe our testing approach and some of the results of our testing, and conclude by describing the direction of our future work in this area.

1 Introduction

The InTENsity PowerWall is a very high-resolution display system built in the summer of 1999 at the Laboratory for Computational Science and Engineering (LCSE) at the University of Minnesota. It represents the second generation of PowerWall technology, pioneered at the LCSE in the mid-1990's. The new PowerWall is comprised of five large flat display screens oriented radially around a central viewing area, with each screen displaying two rear-projected XVGA (1280x1024 pixel) panels. The high resolution of the InTENsity PowerWall allows for detailed visualization of very large data sets. It is also designed to allow for display of full, wall-sized images at rates in excess of 20 frames per second. Figure 1 depicts the InTENsity PowerWall screen and projector layout.

Currently the two major applications for the PowerWall system are generation and presentation of wall content. In "movie generation" mode, the power of either computing environment can be harnessed to render movies for display on the wall. The rendering software is also able to distribute its work across machines. Locating the data sets from

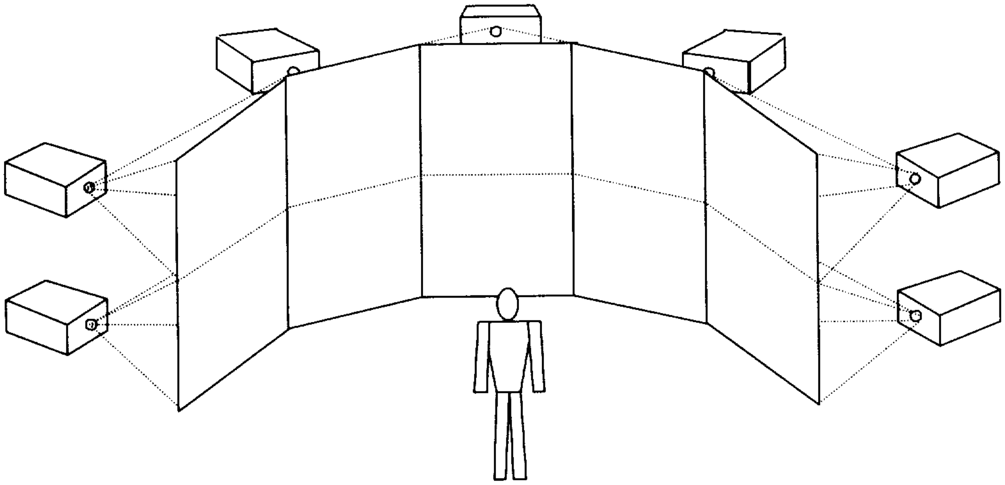


Figure 1. The InTENsity PowerWall at the LCSE

which these movies are derived (as well as the movie files themselves) on shared storage is desirable in order to avoid unnecessary data movement.

The "movie playback" mode requires a great deal of bandwidth from the storage subsystem into the memory of the system driving the displays. For movies to appear reasonably fluid they must be played at a rate of at least 15 frames per second, and preferably greater. These frame rates are made possible by distributing the task of movie display across ten machines. Given the PowerWall's 6400x2048 resolution, a single frame using 4-byte pixels is over 50 MB of data. Thus, over one gigabyte per second aggregate bandwidth is needed to sustain a full-resolution InTENsity PowerWall movie at 20 frames per second.

The InTENsity PowerWall can be driven by either of two distinct computing environments. The first consists of a pair of high-end SGI computers: an Onyx and an Onyx 2, each with two Infinite Reality™ graphics engines. These systems are used primarily for continued support of our existing PowerWall software and hardware base thereby easing the transition to the new InTENsity PowerWall format. Each of these systems has multiple Fibre Channel interfaces providing high bandwidth access to the storage subsystem. The second computing environment consists of a cluster of ten 4-processor SGI Visual PC 540 workstations. Each workstation sends video output to one of the ten panels on the wall. Two Fibre Channel interfaces on each workstation provide access to the storage subsystem.

A fairly complex storage area network was required to meet the performance and connectivity requirements of the InTENsity PowerWall. The result of our design is a storage system that is both capable and flexible. It is also an excellent environment in which to test performance characteristics of emerging storage area network hardware and software technologies. Our existing applications naturally stress storage systems in a number of ways. Yet we believe that application level testing like this is not sufficient to understand the complexity that comes into play to yield a given level of storage system performance. Instead, a more focused and closely controlled test environment is needed.

We will describe in this paper the framework we have developed for performing just this kind of controlled testing in a storage area network environment.

2 The LCSE Storage Area Network

We have constructed a storage area network that connects the systems involved with driving the PowerWall with a common set of disks via a Fibre Channel fabric. We designed this storage area network (SAN) with two main goals:

- Maximizing bandwidth available between hosts and disks
- Maximizing connectivity between hosts and disks

The hosts on the SAN consist of the two "large" Silicon Graphics ONYX systems and 12 "small" Intel-based Windows NT machines. All the hosts are connected to over 5 terabytes of disk storage through a fabric of four 16-port Fibre Channel switches (see Figure 2).

One of the large hosts is an 8-processor Silicon Graphics Onyx 2 which has four 2-port Prisa Fibre Channel adapters connected to the fabric. The second large host is a 4-processor Silicon Graphics Onyx with a total of four Fibre Channel ports connected to the fabric. Each of these machines has two Infinite Reality™ graphics engines for rendering images and/or for displaying movies on the InTENsity PowerWall.

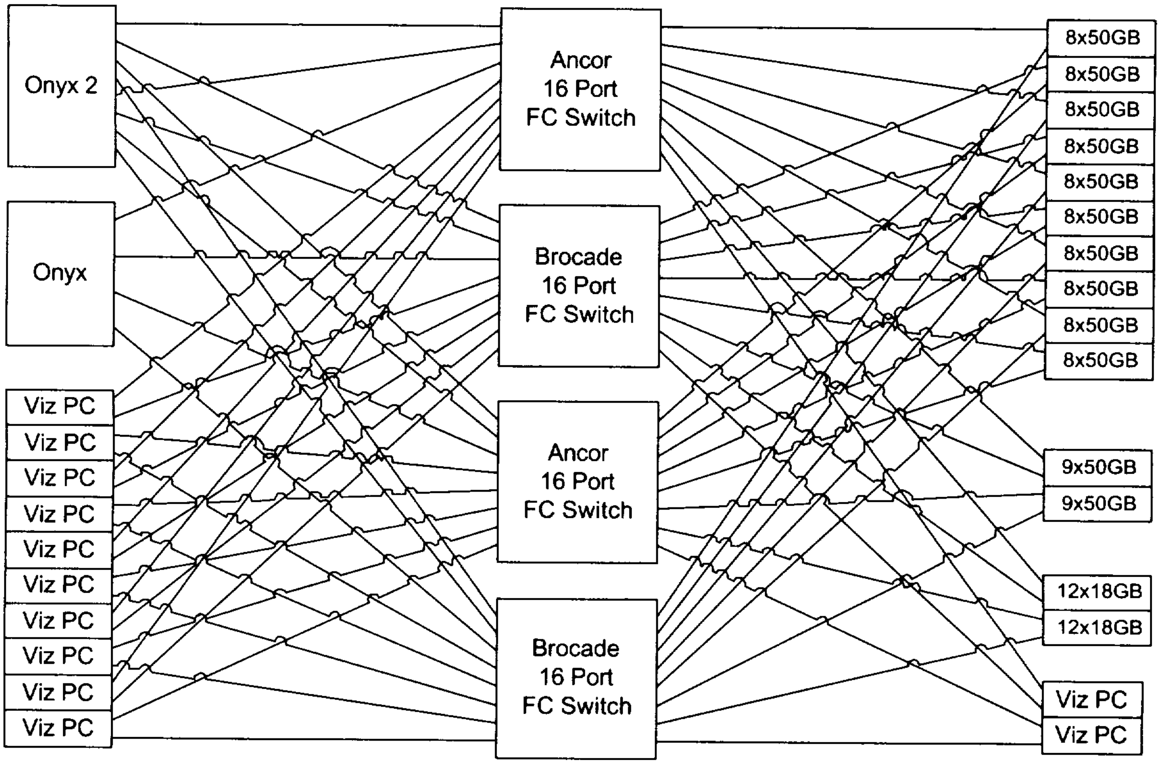


Figure 2. The LCSE storage area network supporting the InTENsity PowerWall

The small hosts are all 4-processor SGI Visual PC model 540 workstations (VizPC's). The graphics outputs of ten VizPC's are used to drive the ten panels of the InTENsity PowerWall. The remaining two VizPC's are used for development and for control of the wall. These machines also expand the computing capability of the cluster, and serve as redundant spares in the event of a failure of one of the wall-driving machines. Each of the VizPC's has a single 2-port Qlogic Fibre Channel host-bus adapter; all of these Fibre Channel ports are connected to the fabric.

Four 16-port Fibre Channel switches implement the fabric. Two of the switches are Ancor MK-II's, and the other two are Brocade Silkworm's. Wherever a Fibre Channel host bus adapter or a drive enclosure has two ports to the fabric, one is connected to an Ancor switch and the other connects to a Brocade switch. Our configuration allows all hosts access to all disks through at least one path through the fabric.

The storage portion of our storage area network is made up of two generations of Seagate Fibre Channel disks enclosed in three types of drive enclosure. Our newest drives, Barracuda 50's, make up the majority of our storage. Ten JBOD enclosures made by JMR hold 80 of these disks. An additional 18 of these drives are enclosed in two Ciprico JBOD boxes. Finally we have two 12-drive MTI JBOD's filled with 18GB Barracuda drives.

3 PowerWall Applications

As mentioned earlier, two current PowerWall applications drive the high performance requirements of its shared storage subsystem. The first application is the generation of movies for display on the InTENsity PowerWall. The second application is the display of the generated movies. Each of these has fairly well-behaved I/O characteristics.

3.1 Movie Generation

Movie generation is a process of rendering movie frames from a very large, time-varying 3-Dimensional data set. Each data set contains many instances of a single volume of data, each instance representing the state of the volume as it evolves over time. The view of the volume is determined through an interactive process, whereby low-resolution image frames are generated and reviewed until a desired viewing location and angle are found. The resulting view information is then saved as a "key frame." A sequence of these key frames define a "flight path" though the data set in space and time. View information for the remaining frames of a movie are defined by interpolation between key frames.

Once the frames along the movie path have been defined the final rendering process is initiated. This process involves rendering full resolution images for display on all ten panels of the wall for each frame in the movie path. Figure 3 depicts a path that rotates around the volume three times before repeating itself.

The data being rendered tends to be on the order of several gigabytes for an instance of a single volume and terabytes for an entire data set. As such, it is currently not possible to render the entire data set using in-core rendering techniques. The data set is therefore organized as a fixed-size hierarchy of sub-volumes. The size of these sub-volumes is a tunable parameter that can be matched to the characteristics of the system doing the rendering; typically they're in the range of 1 MB to 16 MB apiece. This organization allows for efficient data access by the rendering process, which has been designed to run in parallel across many processors and computers in a clustered-computing environment.

The output of each rendering step is a single panel-sized image; at four bytes per pixel, this comes out to 5 MB of data output per step. By using a shared storage resource for the original data set and the resulting movie frame storage, separate host systems can perform any rendering step without concern for excess data movement.

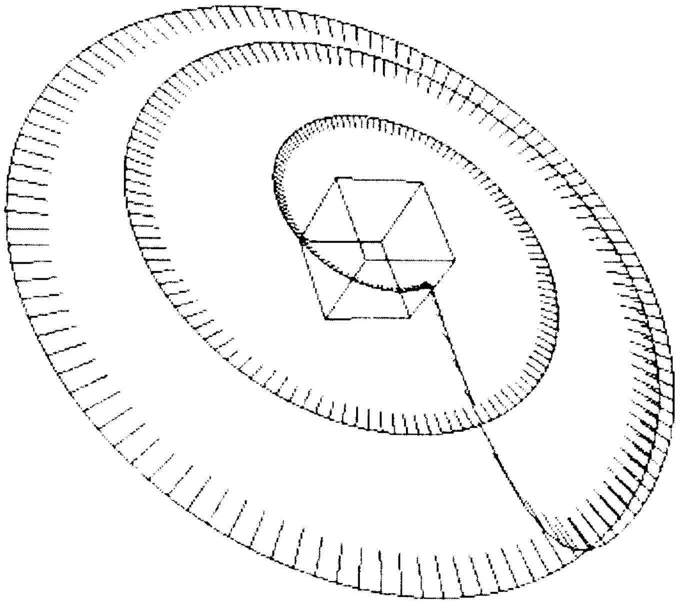


Figure 3. A movie path; each tick represents a frame's view of the volume

3.2 Movie Playback

This application consists of transferring up to ten independent streams of movie frames from the disk subsystem to the display. The streams are synchronized so that corresponding frames on separate screens are shown together. The movie player allows the display frame rate to be adjusted, subject to the limits imposed by the host systems' ability to keep pace with the data transfer rates required. The movie player makes use of read-ahead and asynchronous I/O techniques to maximize the effective data transfer rate and hence, the frame rate.

4 Testing Approach

There are different aspects of performance (such as bandwidth, requests per second, request completion time) that may be of interest for a given situation. But it is inadequate to express any performance metric as a single number, such as 1000 transactions or 20MB per second. Rather, the performance of a single disk (for example) should be expressed as a function of some other variable, such as request size or media position. This is because these other variables can have significant impact on the value of the metric being measured. Furthermore, being able to review a series of measurements in a time-correlated manner is useful in understanding where and when various performance anomalies occur in a storage subsystem. This is especially important in a shared-access environment where a single computer system does not have the ability to exclusively storage subsystem access.

To perform the evaluation of the various system components we have leveraged our existing tools and experience in testing raw storage system performance. The xdd program is a utility developed at the LCSE to assist in determining performance characteristics of the storage devices, both individually and in groups (i.e. logical volumes). The xdd program offers a very fine level of control and produces highly reproducible results, making it more suitable than some other available benchmarking programs for our purposes.

Our approach to performance testing attempts to take into account the all components of the system under test. Where possible, we perform tests that specifically exercise one component to understand its contribution to the overall system performance. This reflects our philosophy that the performance behavior of a complex system can only be understood after first understanding the performance of its components. Our testing attempts to evaluate simple cases, gradually making them more complex. As anomalies in behavior are noted, we consider all components in attempting to determine the cause. Based on this, our approach to understanding the performance of the storage area network was to do a series of single host tests, then move on to more complex tests involving multiple hosts accessing shared storage area network resources concurrently.

For our storage area network, the kinds of components that can impact performance include:

- Components internal to a computer system. These include architectural features which place limits on performance. They also include limits imposed by operating system software.
- Components at the system/storage interface. This covers host bus adapters (HBA's) and their drivers, which are typically developed somewhat independently of any particular type of hardware or operating system.
- Components making up the fabric. This includes the hardware (switches, hubs, media) that make up the communication channel as well as the way in which those components are interconnected.
- Storage devices. Each storage device type has characteristics such as speed and as on-device cache size that can have sometimes surprising effects on performance.

When multiple hosts join to share access to common storage on a storage area network, the interactions become much more complex than the single host case. One host's activities involving the fabric or one or more storage devices can have large and unpredictable impact on the performance. So in addition to the above, we are interested in:

- The shared file system software. We consider this separate from the operating system because in the storage area network case this component is distributed among a number of host systems.

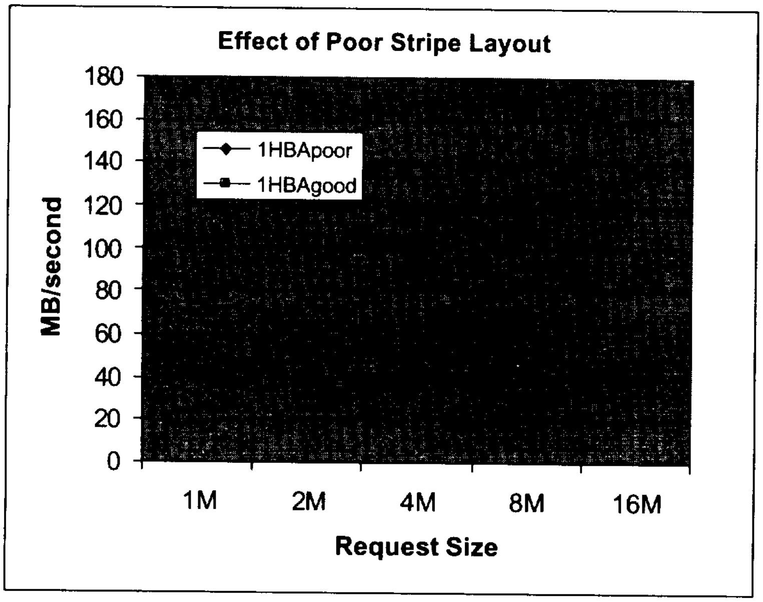
With a firmer understanding of the behavior of the components of our storage area network, we can get a better grasp on interactions that can complicate the performance picture. We have a much better basis for explaining storage system performance.

5 Single Host Testing

The first step in this performance evaluation process is to characterize the performance of a single host system connected to a logical volume through the fabric in isolation (i.e. through the fabric without any other traffic).

This establishes a baseline for further tests. These and all other tests described herein were performed using a single CVFS volume comprised of sixteen 50 GB drives, eight drives in each of two Ciprico JBOD's. Each of these JBOD's has two channel connectors, for the A and B ports of the drives within the enclosure. We connect both channels to the fabric. In addition, both of the Fibre Channel ports on each SAN host are connected to the fabric.

We observed immediately that the performance we were getting from the file system was not close to what we had expected. After some analysis we determined that the way the striping of the volume had been automatically laid out was less than ideal. Laying out the volume the way we had intended yielded a considerable improvement in some of the performance numbers. (Note: All performance results listed here are for read operations.)



0	1	8	9	10	11	12	13
14	15	2	3	4	5	6	7

"Bad" stripe layout

0	2	4	6	8	10	12	14
1	3	5	7	9	11	13	15

'Good" stripe layout

Figure 4. Effect of stripe layout on performance

Next we did some experimenting with the host bus adapter. Our development system had two Fibre Channel HBA's installed, and this gave us an additional option on testing. It allowed testing for the one host case to be performed using either two ports from a single adapter, or one port from each of two adapters. We found that this too made a difference in the rate at which data could be read from disk (see figure 5a).

We did a some testing to examine the effect of varying file system striping parameters on overall file system performance. CVFS allows the file system logical block size to be changed at file system build time. It is also possible to define what they call the "stripe breadth," meaning the number of file system blocks that are read/written to a given disk in a stripe before moving on to the next drive in the stripe. We tried a number of configurations of this: 32x32K, 16x32K, 8x32K, and 16x16K. The performance curves for these combinations are show n in figure 5b.

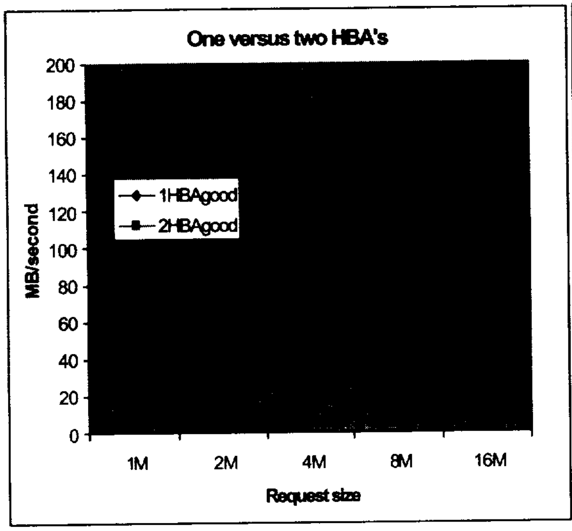


Figure 5a. Improved performance due to use of multiple HBA's

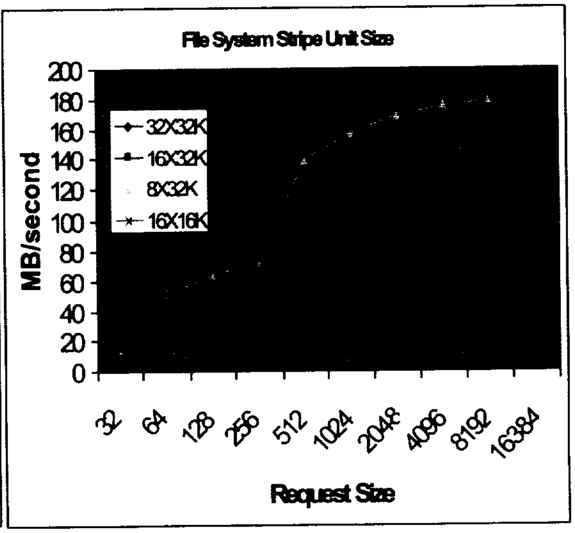


Figure 5b. Affect of different stripe breadths on read performance

This deserves a little further explanation. By reducing the effective stripe breadth the number of bytes transferred to and from a disk drive for a given operation was reduced. The effect of this was to increase performance for the mid-range request sizes (between 128KB and 2MB). The reason for this has to do with the size of the cache on the individual disk drives. Smaller transfers more readily fit in the cache, allowing it to be utilized more efficiently. Larger transfers, meanwhile, tend to overrun the disk cache, thereby losing performance benefit the cache might have offered.

6 Multiple Host Testing

Testing the shared file system software was more complex than testing the underlying storage subsystem and required the creation of a framework to coordinate testing on multiple systems concurrently. The two basic functions of this framework are:

- Accounting for the existence of multiple clocks
- Coordinating the initiation of tests to run concurrently on multiple hosts

Our performance testing utilities make use of precise time stamps to quantify and report storage performance characteristics. Each host accessing the shared storage has its own internal sense of time, and without a common reference clock it is impossible to interpret the relationship between tests run on separate hosts. Thus a consistent time base is needed in order to correlate test results generated by separate systems. We are also able to make use of a common clock to coordinate initiating tests on multiple hosts simultaneously.

6.1 Reference Clock

Each of the systems used for testing has a clock register that updates at a high frequency, allowing for very precise measurement of elapsed time. The resolution of this clock varies for different systems (ranging from 2 to 80 nanoseconds per tick or so), so clock values are converted to a common time unit (picoseconds) for the purpose of synchronization.[1]

We use a very simple clock model to establish a common time base. We assume that all clocks run at the same, constant rate. We therefore assume that conversion from a given machine's "local time" to the common "global time" involves only the addition of a constant to the local clock's value. With this simplified model we must only determine the value of the constant difference between pairs of clocks.

One machine is designated to keep the global sense of time. That machine provides a service with which others communicate to determine the offsets of their own clock from the global time. Each client initiates a request to the server to get the current global time. The difference between the time value returned and the client's local time is recorded as the basis for the offset. This offset is further adjusted to compensate for the propagation delay required to carry the time request and its response over the communication medium. This propagation delay bounds the error in the difference between our estimated and the actual offset between the two clocks. We perform this request/response protocol a number of times, and use the offset associated the minimum propagation delay as the final offset value.

6.2 Coordination of Concurrent Tests

With a common time base defined, it is possible to coordinate the initiation of tests on different host systems. We extended our existing testing software to determine the time offset for the machine under test. The program is provided an indication of a (global) time at which all tests are to begin. This global time is converted to a localized start time using the offset value. The program then polls the high-resolution clock repeatedly until the start time has arrived. At that point test execution begins. Test results generated by individual hosts are buffered during test execution, and saved to disk for later analysis.

6.3 Concurrent Host Test Results

We performed a series of tests using one, two, and four hosts concurrently reading from the same file on the shared file system. The graph in figure 6 shows the performance curves for the aggregate bandwidth achieved across all hosts for each of these tests.. Each host combination uses either two or four Fibre Channel ports connected to the disk subsystem. The results are given for one host using two ports, two hosts using two ports, two hosts using four ports, and four hosts using four ports. We observe that the performance curve for the single host case is the highest of the four up to a request size of 1 MB. This is because all the read operations are purely sequential, which makes ideal use of the caches on the disk drives themselves.

The graph of the two host, two port configuration shows the lowest performance. This is due the performance degradation caused by random I/O effects. Random I/O request patterns reduce performance because of the expense of positioning drive heads; it also does not allow effective use of the disk caches for reads. The remaining two curves did better than this case because they had four channels to the disks, and were able to make use of the additional bandwidth to improve overall performance.

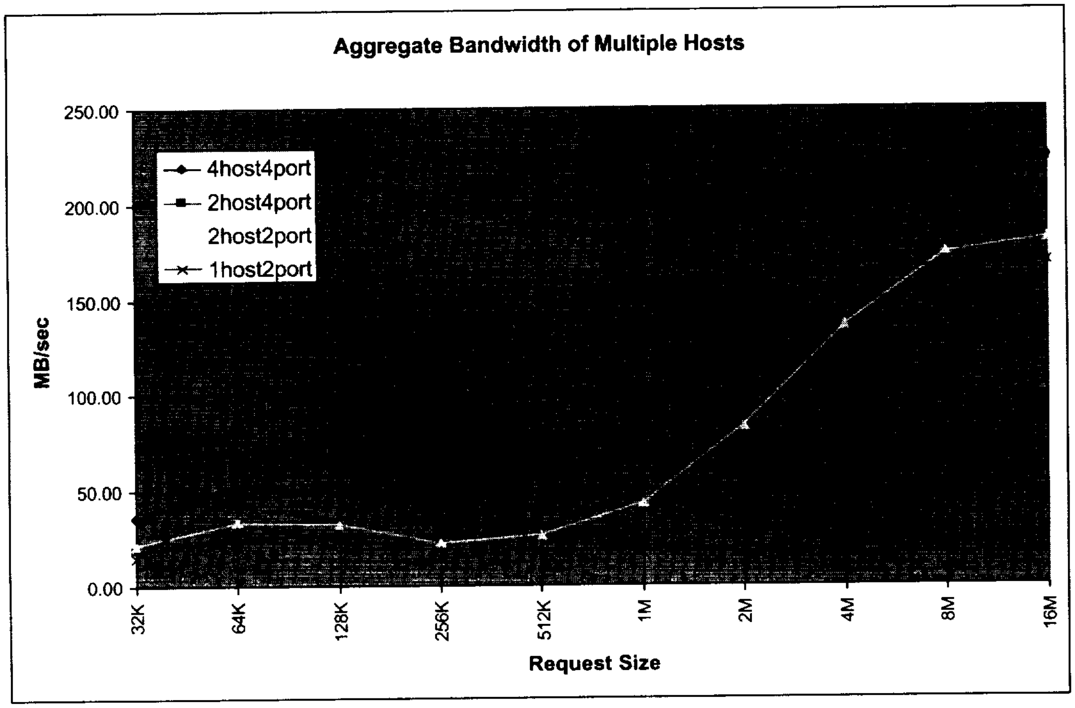


Figure 6. Aggregate bandwidth for multiple host tests.

7 Summary

The InTENsity PowerWall is a high resolution display system backed by a very high performance distributed computing system. The systems and storage area network that drive the PowerWall are a good test bed for evaluating and understanding the performance of shared storage technologies. We built a distributed testing framework that allows concurrent testing by multiple hosts of common hardware to help in evaluating such technologies. Our initial testing has demonstrated that achieving high performance, even in single host systems, is not as straightforward as might initially appear. Furthermore, achieving good, predictable performance in the face of the complexity of a shared storage environment will surely be a challenge. We believe there is much work to be done in this area.

8 Future Work

We have only scratched the surface on the topic of performance testing of shared file systems, and there are many obvious and relevant questions that spring to mind when considering this work. Generally speaking our future work will involve extending the testing framework to evaluate a much larger set of file system functionality. We also intend to continue beyond the limited testing whose results are presented here, and apply the testing framework to include other file system environments. We will be evaluating other network technologies, such as VIA or Myrinet, to assess their effectiveness in improving inter-host communications as well as establishing a more accurate sense of a common time reference.

9 Conclusions

Generation and display of movies on the InTENsity PowerWall at the LCSE are I/O intensive applications that can take great advantage of the benefits offered by shared storage systems. They demand data rates from storage that are both maximal and consistent. The InTENsity system also provides an opportunity for experimentation with and evaluation of emerging shared storage technologies. We have extended our existing disk testing software to accommodate testing performance in a distributed environment attached to common storage. These extensions addressed issues of establishing a common time base and synchronizing the execution of tests on multiple systems. We found that this distributed testing framework served our needs well. It opens up a whole new range of possibility for further study.

References

[1] This work was supported in part by the National Science Foundation, under the NSF Cooperative Agreement No. CI-9619019, and by the Department of Energy through the ASCI Data Visualization Corridor Program under contract #W-7405-ENG-48.

[2] Thomas M. Ruwart, "Disk Subsystem Performance Evaluation: From Disk Drives to Storage Area Networks." NASA/IEEE Joint Storage Conference, March 2000.

[3] Sue B. Moon, Paul Skelly, and Down Towsley. Estimation and Removal of Clock Skew from Network Delay Measurements. Technical Report 98-43, Department of Computer Science, University of Massachusetts at Amherst, October 1998.

[1] We are currently using 100 base T Ethernet as the communication medium through which synchronization information is passed. Because of the latencies involved with this medium, picosecond clock granularity is probably overkill. Nevertheless, we designed the model to ensure it can accommodate better low-latency communication channels as well as the ever-increasing clock rates of computing equipment as they become available.

Alternatives of Implementing a Cluster File Systems

Yoshitake Shinkai, Yoshihiro Tsuchiya, Takeo Murakami
Fujitsu Laboratories LTD.
1-1, Kamiodanaka 4-Chome, Nakahara-ku, Kawasaki 211-8588, Japan
shinkai,tsuchiya,takeo@flab.fujitsu.co.jp
tel +82-44-777-1111
fax +81-44-754-2793
Jim Williams
Amdahl Corporation
1250 East Arques Avenue Sunnyvale, California 94088-3470
jaw10@amdal.com
tel +1-408-737-5005
fax +1-408-737-6595

Abstract

With the emergence of Storage Networking, distributed file systems that allow data sharing through shared disks will become vital. We refer to Cluster File Systems as a distributed file systems optimized for environments of clustered servers. The requirements such file systems is that they guarantee file systems consistency while allowing shared access from multiple nodes in a shared-disk environment. In this paper we evaluate three approaches for designing a cluster file system – conventional client/server distributed file systems, symmetric shared file systems and asymmetric shared file systems. These alternatives are considered by using our prototype cluster file system, HAMFS (Highly Available Multi-server File System). HAMFS is classified as an asymmetric shared file system. Its technologies are incorporated into our commercial cluster file system product named SafeFILE. SafeFILE offers a disk pooling facility that supports off-the-shelf disks, and balances file load across these disks automatically and dynamically. From our measurements, we identify the required disk capabilities, such as multi-node tag queuing. We also identify the advantages of an asymmetric shared file system over other alternatives.

1 Introduction

Historically, large corporations have deployed and distributed UNIX systems in a manner leading to isolated islands of processing. The management cost of these systems is exceedingly high because of the resulting overall complexity and a lack of a global management capability. Consequently, many of these same companies are now re-centralizing their critical Unix systems to contain management costs. One of the benefits credited to SANs (Storage Area Networks) is that they facilitate re-centralization of storage by aggregating it onto a common interconnect. We use the term SAN to describe a dedicated storage network utilizing a storage protocol such as SCSI over Fibre Channel, apart from LAN, which allow servers to communicate using a networking protocol such as TCP/IP. SANs are typically composed of Fibre Channel switches and hubs.

To extract the full potential of a SAN, data sharing, where multiple servers share a common file system on a directly-connected disk, and disk-pooling, where users can place data on any disk (disk-pool) without suffering management overhead, is required. Cluster file systems are the means for achieving these requirements. In this paper we describe our experiences from developing HAMFS. HAMFS [1] is a prototype cluster file system, supporting disk pooling through a shared-disk capability. What was learned from HAMFS has been incorporated into our commercial file system product called SafeFILE. The rest of this paper is organized as follows: Section two describes alternatives to implementing a cluster file system and their characteristics. Section three presents overall HAMFS architecture. Section four shows measurement results from evaluating alternatives and necessary hardware capabilities using HAMFS. Section five describes related works. Finally, section six offers some brief conclusions.

2 Alternatives

As O'Keefe has shown [2], there are three alternatives to achieving data sharing between nodes in a shared-disk environment, client/server distributed file systems, symmetric shared file systems, and asymmetric shared file systems (Figure 1). In a conventional client/server distributed file system, a server node manages disk storage. Other nodes access data through the dedicated server across a communications network. While traditional client/server distributed file systems, such as NFS, cannot distribute user data across multiple nodes, some recent client/server distributed file systems, such as xFS [8], Zebra [9] and Frangipani [7], use multiple nodes for improved scalability. Alternatively, a second approach is symmetric shared file systems, such as GFS [3, 4, 20]. GFS allows every node equal access to all disks directly. The third alternative, asymmetric shared file systems, supports partial disk sharing. This approach was used in HAMFS. In this approach, a dedicated node manages disk blocks containing metadata, but all other disk blocks, containing user data, are accessed directly from all nodes.

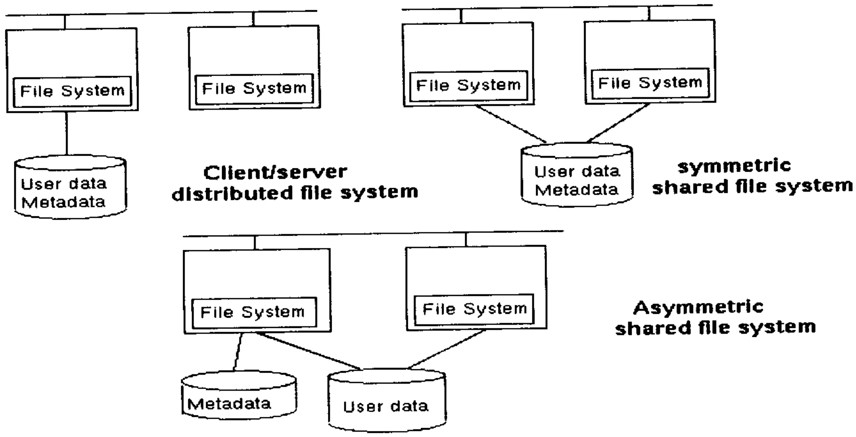


Figure 1: Alternatives for implementing a cluster file system

2.1 Characteristics of each Distributed File System Architecture

2.1.1 Complexity

The most challenging task for a designer of a cluster file system is maintaining consistency while guaranteeing good performance in a multi-node environment. Client caches, heavily used in UNIX file systems for improving performance, pose a major burden for designers. Although the symmetric shared file system has the simplest apparent organization, client caches, required for good performance, requires a complicated distributed lock manager. This leads to a rather complicated file system organization. An example of this complexity is the difficulty of implementing a distributed logging mechanism. Other examples include support for atomic transactions with deadlock detection and recovery.

For symmetric shared file system organizations there are two approaches to achieving file system consistency. The first one is a distributed lock manager, as implemented in VAXcluster [5]. For high performance, VAXcluster offers a sophisticated distributed lock function. To maintain cache consistency, the VAXcluster's distributed lock manager uses both lock versioning for passive cache invalidation, and a callback mechanism for active invalidation. Another approach for serialization, as proposed in GFS, uses special device locks.

For guaranteeing file system consistency, the GFS[1] implementation basically uses a read-modify-write schema and disables the client cache. While this approach is relatively simple, its performance, particularly for short file access environments, is degraded from not having a client cache.

In general, symmetric shared file systems require that all client nodes share a common semantic view of data on disks, including location, record format, and meaning of each field and update sequence. This creates maintenance complexity and makes data sharing more difficult for heterogeneous environments. Any operating system vendor wishing to implement a particular symmetric shared file system must incur this complexity.

Client/server distributed file systems and asymmetric shared file systems avoid much of this complexity by localizing metadata access to a common node. Asymmetric shared file systems make use of a well-known fact that user data is rarely accessed by multiple nodes concurrently from multiple nodes. However, metadata is frequently accessed from multiple nodes concurrently [6]. Consequently, asymmetric shared file systems utilize a dedicated node for metadata management. This approach alleviates the disk contention resulting from concurrent metadata access. Additionally, shared direct access to disks, containing user data, reduces network overhead associated with conventional client/server distributed file systems.

2.1.2 Performance

Conventional client/server distributed file systems typically consume considerable network bandwidth and processor resources for both the client and server. Furthermore, these file systems cannot derive the maximum performance of the underlying disks. And because of their client/server organization, these file systems are not easily scaled through adding additional disks. Balancing performance after adding new servers typically require a manual file system reconfiguration. Both symmetric and asymmetric file systems are easier to scale than conventional client/server distributed file systems like NFS. The reason is because they support common disk pools accessible from multiple nodes that make adding disks far simpler.

A variation of a client/server distributed file system, Frangipani [7], xFS [8], and Zebra [9] solve the scaling problem by distributing data across multiple nodes. However, they still inherit the drawback related to transferring data across a network. While asymmetric

[1] Although a new GFS implements lock versioning for permitting cached data, it does not solve the inherent performance problem. [20]

shared file systems require a means for transmitting control messages across a communications network, the amount of data transmitted is small compared with client/server distributed file systems. Comparatively speaking, asymmetric shared file systems require greater message exchange across a networking than symmetric shared file systems. This is because of the communication between the metadata manager and the other nodes. However, by improving protocols within the asymmetric shared file system, such as a space reserve function, fine grain tokens and token escalation, byte range logging, and support for a secondary buffer cache, much of the performance degradation is minimized. Our experience with HAMFS indicates that these optimizations are possible without significant additional complexity.

2.1.3 Scalability

Both client/server distributed file systems and asymmetric shared file systems have scalability problems related to localized loading of management function on a dedicated node. Distributing the management function across multiple nodes eliminates this drawback. This distribution is akin to the distributed lock manager used in symmetric shared file systems. For instance, both Frangipani and xFS partition file system space into separate segments. A separate node manages each space segment.

2.1.4 Reliability

Both client/server distributed file systems and asymmetric shared file system have a single point of failure because they rely on a single dedicated node. This drawback can be eliminated for asymmetric shared file systems by using a replication schema. Moreover, by deploying an improved logging mechanism, as implemented in HAMFS, and replicating only metadata, asymmetric file systems outperform local file systems. This is a significant advantage over client/server distributed file systems since they would otherwise require mirroring all data.

Disk contention and distributed lock management overhead leads to performance problems for symmetric shared file systems. However, with asymmetric shared file systems, these issues can be addressed through various compromises. The following section describes HAMFS' organization and describes some of these compromises.

3 HAMFS file system

HAMFS is classified as an asymmetric shared file system. Describing the detailed operation of HAMFS is beyond the scope of this paper, however we do briefly review its high-level organization.

3.1 Configuration

HAMFS divides the contents of the file system into three parts, metadata, user data, and log data, which are stored on Meta volumes, Data volumes, and log volumes, respectively.

HAMFS software consists of two components, HAMFS client, and the Name Server. To prevent contention from frequent Meta volume writing, the Name Server, which manages Meta volumes and log data, runs on a dedicated node in a cluster system. Conversely, HAMFS client (client) code embedded in the kernel of each node as a Virtual File System (VFS), resides on every node in a cluster and directly accesses Data volumes. Data volumes access is managed with the file extent information provided by the Name Server. This information reflects data location on each Data volumes.

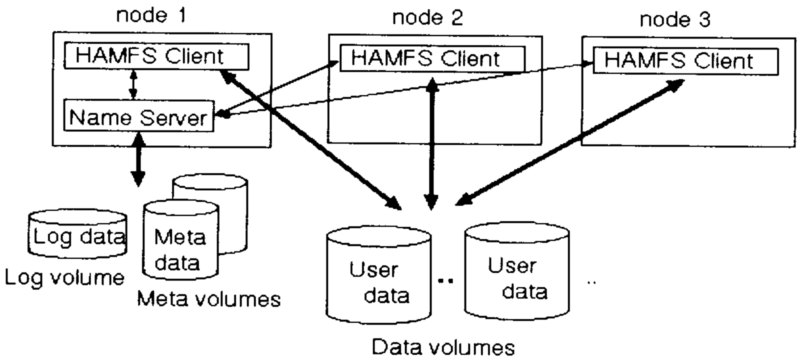


Figure 2: Configuration of HAMFS file system

The Name Server processes requests from the client as atomic transactions using a log schema. The interface between the client and the Name Server is a high-level protocol and is independent of metadata format. This protocol is similar to the NFS protocol. For improved availability, the user may replicate the Name Server on a secondary node. When a secondary Name Server is deployed, Meta volumes are replicated on the secondary Name Server by transmitting the log data. HAMFS file organization permits users to select the disk organization used for each data type as well as whether metadata is replication.

3.2 Disk pooling

HAMFS offers the following disk pooling capabilities.

3.2.1 Disk Pool

A disk-pool is defined as a set of data volumes comprising a HAMFS file system. HAMFS manages each member disk. Data placement across the disk pool is managed automatically according to a placement policy. Current placement policy equalizes the amount of free space across volumes.

3.2.2 File RAID

Because HAMFS manages the underlying disk devices, it's possible to allow files to have different RAID properties in a common name space. File RAID is the function for permitting this file placement policy. A user can specify RAID type of None (default), RAID 0 (striping), RAID 1, and RAID 5 through a new CHATTR command. All directories and files inherit a parent directory's RAID type. With RAID five the client allocates a parity block for each stripe in a file. Since this works like a RAID type one for a small files having only one data block, common in UNIX environments, parity recalculation overhead is greatly reduced.

3.2.3 Dynamic Expansion

An installation may dynamically expand a file system with new disks even though user applications are running. Because HAMFS automatically balances load across the old and new disks, file name tree reconstruction is unnecessary. A newly added Data volume remains offline until all client nodes send ADDVOL requests for the new volume. The Data volume is then brought online only after the Name Server receives an ADDVOL requests from every client.

3.3 Tokens

For improved file system performance, each client caches user data and file information (file data) in their local memory. The consistency of cached file data is guaranteed with tokens that are managed by the Name Server.

There are five token types, NAME, TIME, SIZE, ATTR and DATA. NAME and ATTR tokens correspond to directory entry and file attributes (or directories) that are not represented by other tokens.

Every file has a set of associated DATA tokens. There is a corresponding DATA token for each file data block. Owning a DATA token guarantees accuracy of file extent information reflecting the location of data on the Data volumes. Using this file extent information, clients access Data volumes directly and independently. The Name Server provides file extent information along with the token when a client requests a DATA token. When new data is written, the client must first obtain the corresponding DATA write token, which allows the client to cache the user data locally. Afterwards, if another client wants to read this cached data, the Name Server callbacks the write DATA token from the client owning it. On receipt of a callback request, the client allocates new disk space (file extent) and writes any cached user data. The client honoring the callback request also updates any newly allocated file extent information.

The TIME token allows multiple clients to concurrently access common files while guaranteeing the accuracy of file access times. The SIZE token permits independent programs running on different nodes to both write and read from different parts of the same file.

On a data token request, the Name Server provides all the required non-data related tokens plus, if available, all DATA tokens for the entire file (Token Escalation). Through token escalation, most clients obtain the needed tokens, including file extent information, at file open time.

3.4 Space Allocation

When a client must allocate additional disk space, it selects a pre-allocated extent based on the amount of cached data to be written. In most cases, only one extent with consecutive disk blocks is allocated at file close time. The Name Server is notified of this through the CLOSE request. On notification of their usage, pre-allocated (reserved) extents become file extents when the DATA write token is released or the file is closed. When the number of pre-allocated extents drops below a threshold, clients replenish their free pool with a RESERVE request made to the Name Server. On receipt of a RESERVE request, the Name Server provides free extents on a Data volume with the greatest free space.

To reduce wasted space related to allocation, every extent, including free extents, is represented with a B-tree data structure. The Btree structure is stored in the inode for files consisting of only a few contiguous extents. With this space allocation function, combined with deferred file writes, reduces message overhead and permits allocating contiguous disk blocks for files.

3.5 Transaction processing

For increase performance and availability, the Name Server processes file operations requested by clients as a single atomic transaction using logging and a two-phase lock with deadlock detection mechanism.

3.5.1 Logging

These transactions complete quickly because they require writing only a small amount of data to the log volume. Actual updates to Meta volumes are deferred as long as possible and are performed completely and asynchronously.

Log data generated by HAMFS is an after image log containing only the modified range of data instead of the entire contents of the modified blocks. This byte-range log significantly reduces the amount of log data and improves metadata update performance. The log data is written on a log volume in a cyclic fashion, synchronously before a command response is returned to the client. Actual metadata updating of Meta volumes is deferred as long as possible by using a secondary buffer cache. The secondary buffer cache caches any modified and committed metadata that has not been written back to a Meta volume. Dirty metadata, cached in the secondary buffer cache, is asynchronously written when the amount of available space in the log volume or the number of non-dirty blocks in the secondary cache reaches a threshold. Furthermore, since the metadata writes are aggregated, the total I/O load is reduced.

3.5.2 Deadlock detection

With the HAMFS token schema, deadlock avoidance, as used in conventional file systems, would be difficult to implement. For example, assume the following scenario. The Name Server processes a file remove request for one client while another client is writing to the file. To complete the remove request, the Name Server must callback any DATA write tokens related to the file from other clients. On receipt of this callback request, the client requests the Name Server to pre-allocate disk space with a RESERVE request for writing back cached data. After the Name Server replies to the RESERVE request, the client writes its cached data to the pre-allocated extent returned from the Name Server. Afterwards, it notifies the Name Server of any allocated extents. In such circumstances, determining the access order to resources required for preventing deadlocks, while probably possible, would be difficult.

In HAMFS, we developed a deadlock detection mechanism for easing development and maintenance. The Name Server uses several threads for processing client requests. When requests arrive, idle threads process the requests. A two-phase lock technique maintains consistency among these requests. Deadlocks result when two or more threads obtain multiple vnode locks, buffer cache locks, or tokens. Deadlocks are detected by maintaining the following information in a linked list by the Name Server.

- Identifier of the thread owning a resource for vnode or buffer cache lock.
- Resource identifier for threads waiting.
- Vnode address for tokens a thread is waiting on.

When a deadlock situation occurs, the Name Server cancels one of the conflicting transactions that led to the deadlock and retries it from the beginning. Memory resident control blocks, such as in-memory inodes, are automatically restored. With HAMFS deadlock detection and recovery, the order for updating metadata can be selected from a performance viewpoint as opposed to a consistency and deadlock avoidance viewpoint. Therefore, complicated error recovery and deadlock avoidance logic, scattered throughout the file system, is avoided.

3.6 Replication

For fast recovery and improved performance in cluster environments, HAMFS replicates the Name Server. There can be separate primary and a secondary Name Servers for each file system. However, HAMFS clients only communicate with the primary Name Server.

The secondary Name Server possesses a replicated copy of the metadata. If the primary Name Server crashes, the secondary Name Server takes over the primary role using the replicated metadata. When a secondary Name Server is deployed, instead of writing log data to a log volume, before signaling an operation complete, the primary Name Server simply transfers the log to the secondary Name Server. The secondary Name Server acknowledges its receipt. After receiving the acknowledgment, the primary Name Server is free to signal a completion to the client. We call this technique Early Commit. Actual metadata updating on Meta volumes is deferred and done asynchronously using a secondary buffer cache on both name servers. If a power failure occurs, modified metadata, in the secondary buffer cache, is written back to the Meta volumes with the aid of an UPS.

3.7 Crash Recovery

When a client recognizes that the primary Name Server has crashed, it begins communicating with the secondary Name Server (new primary Name Server). The new primary Name Server reconstructs tokens and file lock status using information sent by the clients. Afterwards, the Name Server resumes processing. If the new primary Name Server detects requests already committed by the old primary Name Server it simply replies with the saved reply status transmitted by the old primary Name Server.

When the Name Server detects a client crashed, the Name Server releases any tokens and file locks held by the failed client. After this step, the Name Server releases pre-allocated extents also owned by the crashed client.

3.8 Status

The current HAMFS prototype is operating in our laboratory with the following configuration. The Name Server running as a user mode daemon with multiple threads on Solaris. Clients are running as a VFS file system in the FreeBSD kernel. All functions, except for file RAID described in this paper, have been implemented.

4 Measurement Results

We evaluated various file system alternatives by measuring performance of an asymmetric file system (HAMFS) and a distributed file system (NFS). We did not measure performance of a symmetric shared file system. We can only comment on the performance of a symmetric file system where it relates to our other measurements.

We used three PCs with 64MB memory for these measurements. For the NFS measurements, two PCs were running FreeBSD with one operating as a NFS client and the other as a NFS server, respectively. For the HAMFS measurements the three PCs were configured with the first PC running FreeBSD and acting in a client role and the other two PCs operating as primary and secondary Name Servers running the Solaris operating system. Consequently, the measured data represents HAMFS performance for a cluster environment.

These three PCs were connected together with a 100Mbps Ethernet. In every case, a common disk was connected to the first two PCs. This disk contained all HAMFS volumes, except for replicated Meta volume. This was done to create a fair comparison with NFS. The replicated Meta volume resided in a dedicated disk on the third PC running the secondary Name Server. This replication configuration is justified because separate disk should be used in a replicated environment.

4.1 Distributed file system vs. Asymmetric shared file system

We measured HAMFS and NFS performance for small and large files. This was done to compare access performance of a distributed file system to an asymmetric shared file system. We chose NFS version 3 to represent the distributed file system. For NFS measurements, two PCs running FreeBSD were used. Since NFS invalidates cached data on a predetermined time interval, it generates more control traffic over the network than other distributed file systems. But we believe the measured results apply to other client/server distributed file systems.

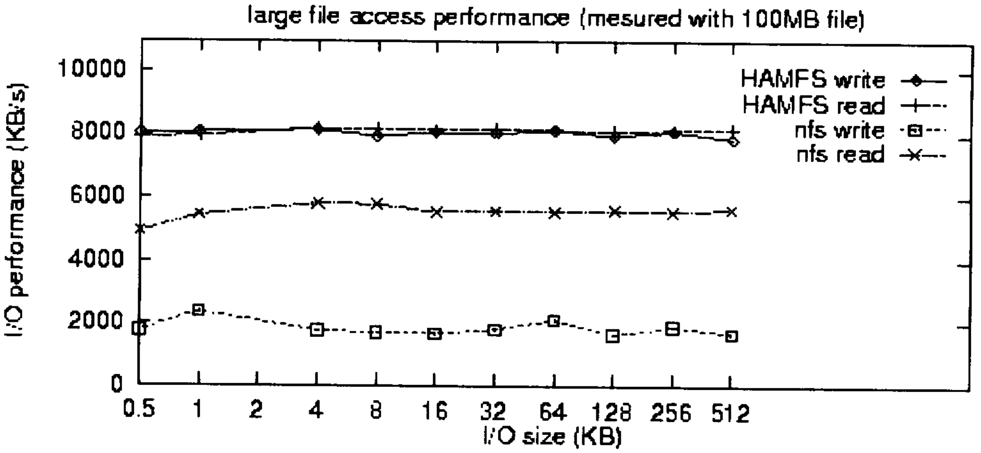


Figure 3: Large file write performance

Figure 3 shows the performance for large file access environments. In these measurements, a 100MB file is created and read sequentially from the beginning. HAMFS shows far greater performance, compared with NFS, because it is accessing data directly from disk through a high-performance disk path. An interesting point is that NFS write performance is poor although writing data on server side is done asynchronously with the NFS v3 commit feature. The reason for this is that the network cannot drive the disk with enough data necessary to derive its maximum performance, resulting in additional rotational delay. On the other hand, as the track buffer in the disk unit offsets the network overhead, NFS reads reflect relatively good performance. As disk transfer rates continue to improve, delays due to networks will further impact performance. However, the client cache provides a speed-matching buffer, which alleviates this problem, somewhat by aggregating data into larger chunks and writing them as contiguous blocks on disk. We believe asymmetric shared file systems have an advantage in this regard over symmetric shared file systems because they can make better use of the client cache.

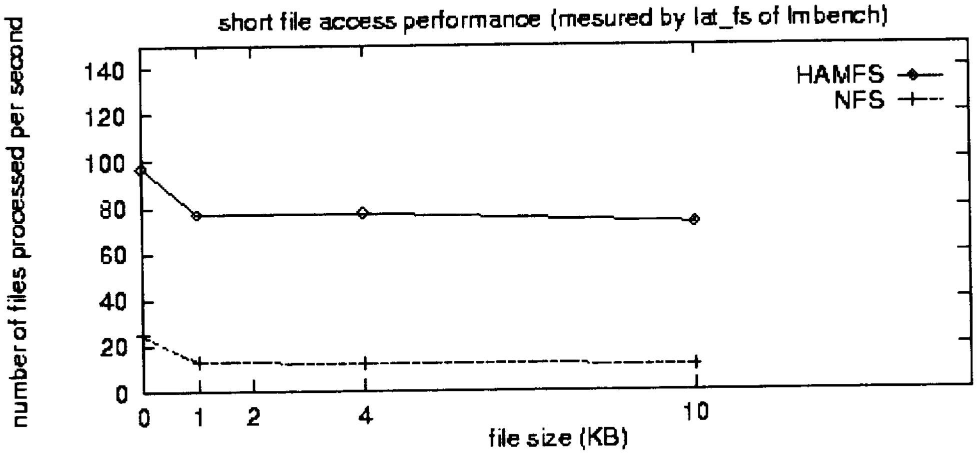


Figure 4: Short file access performance

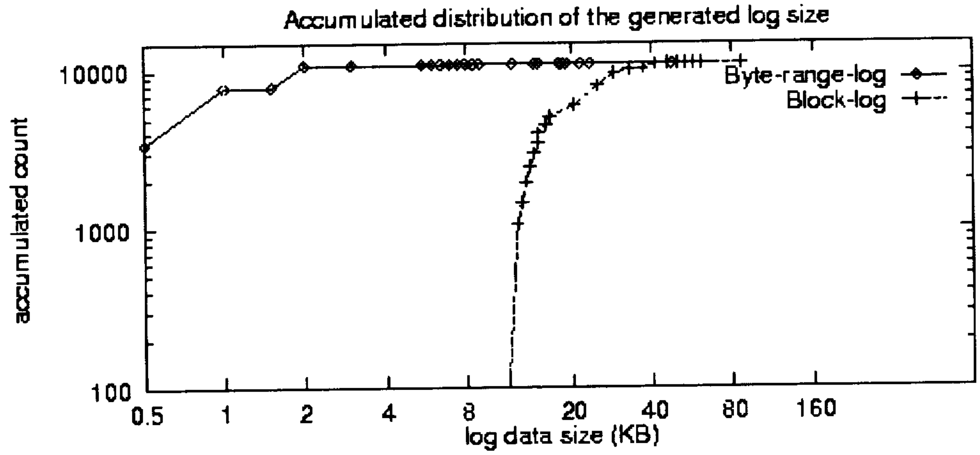


Figure 5: The amount of log generated

Figure 4 presents the performances for short file access environments. These measurements were conducted using the lat_fs program in lmbench [10] micro benchmarks. This micro bechmark program creates 1000 files with various sizes and then removes them. The vertical line reflects how fast this program runs. As shown in Figure 4, HAMFS, with a secondary Name server, outperformed NFS by a factor of five. The reason is that metadata updating through Early Commit is faster than doing actual disk I/O.

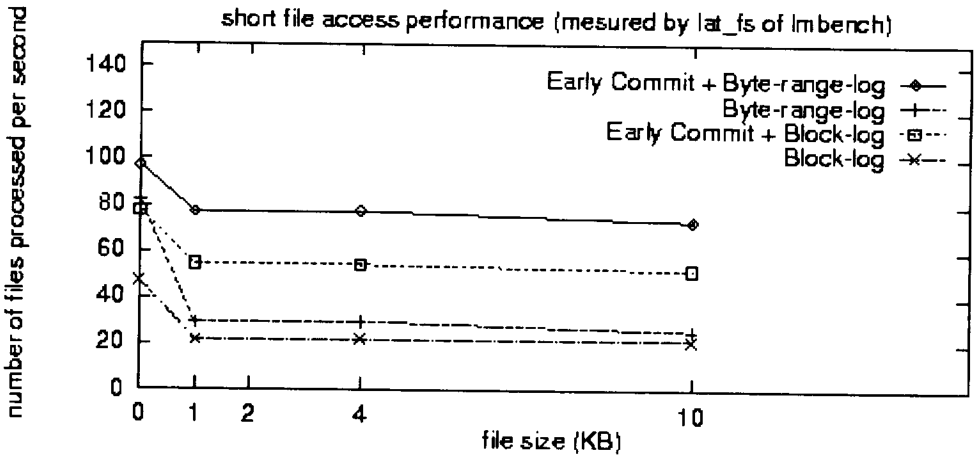


Figure 6: Effect of a Byte-range log

Figure 5 and 6 shows how size of log affects total performance.

Since HAMFS uses a Btree data structure for representing space information on Meta volumes, more disk blocks are updated on behalf of a file operation when compared with UNIX file systems. UNIX uses an array of block address lists and bit maps for representing available disk blocks. Figure 5 illustrates the distribution of generated log data size per file operation in this measurement. Without a byte-range log, most log data writes would be 30K data. Less than 2K bytes of log data is generated with a byte-range log (that is when the log is produced with byte granularity). This difference is apparent even in a single name server environment (the second line vs. the last line in Fig. 5)[2]. It has a more drastic effect on the performance in a duplicated name server environment (the first line vs. the third line in Fig. 5). Additionally, this figure indicates the superiority of asymmetric shared file systems over traditional client/server distributed file systems for improving availability through replication. Traditional distributed file systems must transfer all user data, as well as metadata, making it difficult to reduce data replicated over the network. Finally, we believe this measurement also reflects the superiority of asymmetric file systems over symmetric file systems since logging puts additional implementation burden for symmetric shared file systems.

4.2 Limitations of asymmetric shared file systems

Figure 7 shows aggregate write performance for large files in a shared environment. In this measurement, two PCs are used as clients with a third PC running as a name server.

[2] In actual single node environments this difference may have more impact on performance. Because HAMFS measurements were conducted in a cluster environment with the Name Server and the client running on separate nodes, the logging overhead has less effect on total performance.

Performance measurements are not as good when compared with Figure 3. The last line, annotated UFS, shows how performance degradation is independent of file system type. The reason is that the disk devices used for these measurements cannot efficiently process requests from multiple nodes. To validate our hypothesis, we measured the case when two processes running on the same client write large, but separate files on separate UFS file systems (disk partitions). The second line tagged with UFS single shows this performance impact.

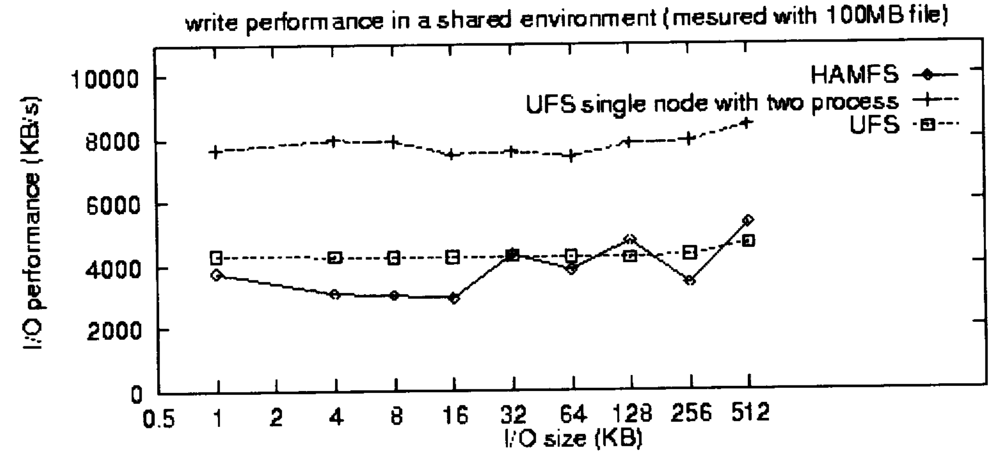


Figure 7: Large file access in a shared environment

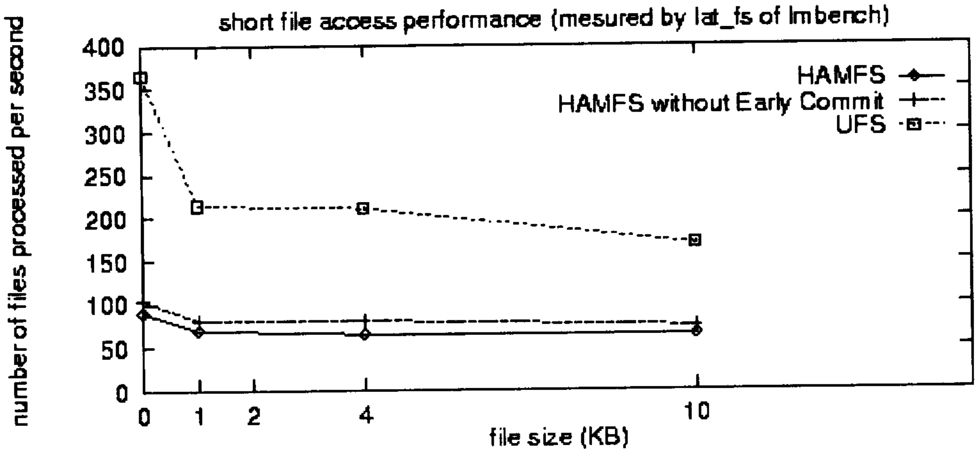


Figure 8: Short file access with high-speed disk array

As the figure 7 shows, disk tag queuing performs well when two processes are running on the same node and writing concurrently. On the other hand, when two processes on separate nodes write large files, the loss of tag queuing is evident. Consequently, for extracting maximum performance from disks in a cluster environment, tag queuing support across multiple nodes is a critical feature.

Figure 8 shows performance for short file operations on a high speed RAID disk array with 32MB nonvolatile memory.

Compared with Figure 6, HAMFS performance is not as good as indicated in the previous measurements. In previous measurements, HAMFS outperformed the local file system, however, for this measurement, UFS outperformed HAMFS. The reason for this is that writes for short blocks are faster with a nonvolatile cache in a RAID array than having to transfer control data over network. This result also suggests that writing small log data to disk (HAMS without Early Commit) is faster than transferring log data across a network (first line and the second line in Figure 8). Consequently, eliminating any unnecessary communication between the nodes and reducing the amount of data transferred over the network is essential to an efficient cluster file system. An important message is that a cluster file system must adapt to underlying disk topology for best performance. These measurements indicate that the preferred mode of operation is highly dependent on system configuration.

5 Related Works

Devarakonda [11] evaluated alternatives for implementing a cluster file systems. The author compares a symmetric shared file system with a token-based client/server distributed file system. They conclude that their client/server distributed file system (Calypso) provides much better performance than a symmetric shared file system. In our paper we have shown how an asymmetric shared file system can outperform a distributed file system organization.

GFS [3,4, 20] is an example of a symmetric shared file system. It proposes a special hardware feature in the disk providing multiple logical locks. However, an asymmetric shared file system can accommodate off-the-shelf disk devices. Additionally, we expect GFS suffers from low performance as a result of heavy disk contention except in specialized environments such as broadcasting.

NASD [12] and HPSS [13] are similar to asymmetric shared file systems. They isolate metadata and user data, and permit shared access to user data directly from clients through a high-speed communications network. However, since they both transfer user data across a communication network they have performance limits that HAMFS does not. In addition, they require special hardware features as GFS does; HPSS utilizes a complicated two-phase commit mechanism suitable only in scientific environments where large files are dominant. NASD depends on intelligence in the disk devices for space allocation.

Zebra [9] and xFS [8], are examples of client/server distributed file systems. They scale performance by distributing data across servers in a RAID schema through a LFS technique. However, user data is transferred over a communications network and their RAID schema is fixed. HAMFS supports file RAID for increased performance and allows the user to specify data striping policy on a file basis. Frangipani [7] also offers a similar capability by distributing data across servers using a network virtual disk function. This approach has many of the same drawbacks as xFS.

HAMFS deploys many of the similar techniques developed for increased performance and availability used in many of the documented file systems, such as Cedar [14], Echo [15], XFS [16], Locus [17], HARP[18], and Spritely NFS [19]. Additionally, HAMFS offers features that others do not, including Token escalation, Space reserve, and automatic deadlock detection.

7 Conclusions

The asymmetric shared file system organization is a superior approach for implementing a commercial cluster file system. They outperform client/server distributed file systems and symmetric shared file systems for many common access environments. Because disk bandwidth improvements have outpaced network bandwidth improvements, asymmetric shared file systems' performance is superior to that of distributed file systems. Additionally, processor overhead associated with distributed file systems is not evenly distributed across clients, but highly localized to a server. In a large cluster environment, this limitation quickly becomes a bottleneck. However, to extract the full performance of an asymmetric shared file system, tag queuing across multiple initiators is required. Also, having an efficient protocol for reducing communication between clients and Name Server is important as well. Finally the current HAMFS prototype may have some scalability limitations because of a single Name Server per file system restriction. We do not expect this limitation to be a real problem in the short term. If required, we could remove this restriction using similar techniques as used in the Frangpani distributed file system.

References

[1] Yoshitake Shinkai, Yoshihiro Tsuchiya, Takeo Murakami, and Jim Williams. HAMFS File System. In proceedings of 18th IEEE Symposium on Reliable Distributed Systems, pages 49-70, Lausanne, Switzerland, October 1999.

[2] Matthew T.O'Keefe. Shared file systems and fibre channel. pages 1-16, In proceedings of Fifteenth IEEE Symposium on Mass Storage Systems, College Park, Maryland, March, 1998

[3] Steven R. Soltis, Grant M. Erickson, Kenneth W. Preslan, Matthew, T. O'Keefe, and Thomas M. Ruwart. The Design and Performance of a Shared Disk File System for IRIX. In proceedings of Sixth NASA Goddard Space Flight Center Conference on Mass Storage Systems and Technologies, College Park, Maryland, March, 1998

[4] S. R. Soltis, T. M. Ruwart, and M. T. O'Keefe. The global file system. In proceedings of the 5th NASA Goddard Mass Storage Systems and Technologies Conference, College Park, MD., September 1996.

[5] Nancy P. Kronenberg, Henry M. Levy, and William D. Strecker. VAXclusters: A Clos ely-Coupled Distributed System. ACM Transactions on Computer Systems, 4(2): 130-146, May 1986.

[6] Mary Baker, John H. Hartman, Michael D. Kupfer, Ken Shirriff, and John K. Ouster hout, Measurements of a Distributed File System. pages 198-212, In proceedings of th e Thirteenth ACM Symposium on Operating System, Pacific Grove, California,Octobe r1991.

[7] Chandramohan A. Thekkath, Timothy Mann, and Edward K. Lee. Frangipani: A Scalable Distributed File System. In proceedings of sixteenth ACM Symposium on Operating System Principles, pages 224-237. Saint Malo, France, October 1997

[8] Thomas E. Anderson, Michael Dahlin, Jeanna lvi. Neefe, and David A. Patterson. Serverless network file systems. ACM Transactions on Computer Systems, 14(1):41-79, February 1996.

[9] John H. Hartman and John K. Ousterhout. The zebra striped network file system. .ACM Transactions on Computer Systems, 13(3):274-310, Aug. 1995.

[10] Larry McVoy and Carl Staelin. lmbench: Portable tools for performance analysis. In pr oceedings of USENIX 1996 Annual Technical Conference, San Diego, CA., January, 1 996.

[11] Murthy V. Devarakonda, Ajay Mohindra, Jill Simoneaux, and William H. Tetzlaff. Evaluation of design alternatives for a cluster file system. In USENIX 1995 Technical Conference Proceedings, pages 35-46, Berkeley, CA, January 1995.

[12] Garth Gibson, David Nagle, Khalil Amiri, Fay Chang, Eugene Feinberg, Howard Gobioff, Chen Lee, Berend Ozceri, Erik Riedel, David Rochberg, and Jim Zelenka. File server scaling with network-attached secure disks. In Proceedings of the ACM International Conference on Measurement and Modeling of Computer Systems, Seattle, WA., June 1997.

[13] David Fisher, John Sobolewski, and Terrill Tyler. Distributed metadata management in the high performance storage system. In Proceedings of the FIRST IEEE METADATA CONFERENCE, Silver Spring, Maryland, April 1996.

[14] Robert B. Hagmann. Reimplementing the cedar file system using logging and group commit. In Proceedings of Eleventh ACM Symposium on Operating System Principles, pages 155-162, Austin, Texas, November 1987.

[15] Andy Hisgen, Andrew Birrell, Charles Jerian, Timothy Mann, and Garret Swart. New-value logging in the echo replicated file system. Technical Report SRC Research Report 104, Digital Systems Research Center, June 1993.

[16] A. Sweeney, D.Doucette, W. Hu, C. Anderson, M. Nishimoto, and G. Peck. Scalability in the xfs file system. In proceedings of the USENIX 1996 Annual Technical Conference, pages 1-14, San Diego, CA, January 1996.

[17] B. J. Walker, G. J. Popek, R. English, C. S. Kline, and G. Thiel. The locus distributed operating system. In proceedings of Ninth ACM Symposium on Operating System Principles, pages 49-70, Bretton Woods, New Hampshire, October 1983.

[18] Barbara Liskov, Sanjay Ghemawat, Robert Gruber, Paul Johnson, Liuba Shrira, and Michael Williams. Replication in the harp file system. In Proceedings of the Thirteenth ACM Symposium on Operating System Principles, pages 226-238, Pacific Grove, CA, October 1991.

[19] Jeffrey C. Mogul. Recovery in spritely nfs. ACM Transactions on Computer Systems, 7(2):201- 262, 1994.

[20] Kenneth W. Preslan, Andrew P. Barry, Jonathan E. Brassow, Grant M. Erickson, Erling Nygaard, Christopher J. Sabol, Steven R. Soltis, David C. Teigland, Mattew T. O'Keefe. A 64-bit, Shared Disk File System for Linux. In proceedings of Seventh NASA Goddard Space Flight Center Conference on Mass Storage Systems and Technologies, pages 22-41,San Diego, California, March 1999.

StorHouse/Relational Manager (RM) – *Active* Storage Hierarchy Database System and Applications

Felipe Cariño Jr.
FileTek. Inc.
360 N. Sepulveda Blvd. Suite 1080
El Segundo CA 90245
FCARINO@filetek.com

John Burgess
FileTek, Inc.
9400 Key West Avenue
Rockville MD 20850
JGB@filetek.com

Abstract

This paper describes how database systems can use and exploit a cost-effective ***active storage hierarchy***. By active storage hierarchy we mean a database system that uses *all* storage media (i.e. optical, tape, and disk) to store and retrieve data and not just disk. We describe and emphasize the *active* part, whereby all storage types are used to store raw data that is converted to strategic business information. We describe an evolution to the Data Warehouse concept, called ***Atomic Data Store***, whereby atomic data is stored in the database system. Atomic data is defined as storing all the historic data values and executing queries against the historic queries. We also describe a Data Warehouse information collection, flow and central data store ***Hub-and-Spoke architecture,*** used to feed data into Data Marts. We also describe a commercial product; ***StorHouse/Relational Manager (RM)***. RM is a commercial relational database system that executes SQL queries directly against data stored on the storage hierarchy (i.e. tape, optical, disk). We conclude with a brief overview of a real world AT&T Call Detail Warehouse (CDW) case study.

1.0 Introduction

Commercial Database Management Systems (DBMS) have evolved and been developed to diverse and ubiquitous range of applications. DBMS have been based on hierarchical, network, relational, object-oriented and the new emerging object/relational database model. With few exceptions these database systems and applications primarily use disk media as their storage. Hierarchical Storage Management (HSM) is used by some of these applications to exploit some of the benefits of cost-effective optical storage systems.

We propose and analyze that database systems use and exploit a complete active storage hierarchy (i.e. tape, optical, and disk). The key proposal is that active data be *also* stored, queried and analyzed on tape farm libraries and optical jukeboxes. Figure 1 shows the cost, performance, size and reliability considerations. In this paper, we use the term active storage hierarchy when (say SQL) queries execute against data stored on diverse media.

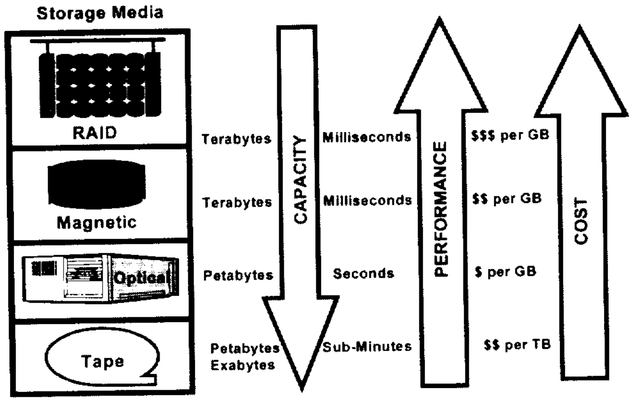


Figure 1: Cost, Performance Size Tradeoffs

We provide an overview analysis of different commercially available storage systems. We provide an update to tape and optical technology, performance and systems described and analyzed in [HS 96] [JM 98]. During VLDB 1998 10-year best paper award [GB 88] talk Jim Gray described emerging disk technology trends. Key disk technology trends are more storage and CPU-like processing capabilities [RGF 98]. We claim and discussed at a [VLDB 99] panel why the complete storage hierarchy media is needed to address certain application needs (i.e. atomic data, objects, etc.)

The *current* economics of the storage media are that storing data on tape is approximately 7% the cost of storing the data on disk. The cost for storing data on optical is about 42% of storing the data on disk. Some interesting byproducts of cost-effective use of active storage hierarchy described in the paper are:

- *Atomic Data:* where all the historical data can be stored and directly queried. Other users must make decisions what data to keep "on-line" on disk for querying. We could almost summarize this paper by the following: "Users should keep all data on-line for querying all the time". Data can be moved to the most cost-effective storage hierarchy media. Instead, what customers do today is "age out" data to archive and rarely restore it for querying.
- *Atomic Data Store (ADS):* a Data Warehouse (DW) concept evolution where historical atomic data is stored and used for information mining or decision support. Bill Inmon spawned an information revolution with what is now know as a Data Warehouse [Imn 92]. There are several potential DW and Data Mart architectures [Gar 98] and philosophies [BZ 98]. One of the DW architectures is an enterprise-wide DW where detail data is stored and used for strategic reasons [Arm 97]. Implicit in [Arm 97] value of detail data argument is that data is aged or migrated out of the repository. The case for ADS DW concept is the same as DW vendors that sell the notion of storing detail data, except that in ADS the detail data is actually stored and used (i.e. not migrated out of the DW).
- *Hub-and-Spoke Architecture (Figure 2):* where operational data stores load (all) their atomic data values into a Data Warehouse with active storage, and then feed Data Marts. This provides a cost-effective way to manage and load data in Data Marts.

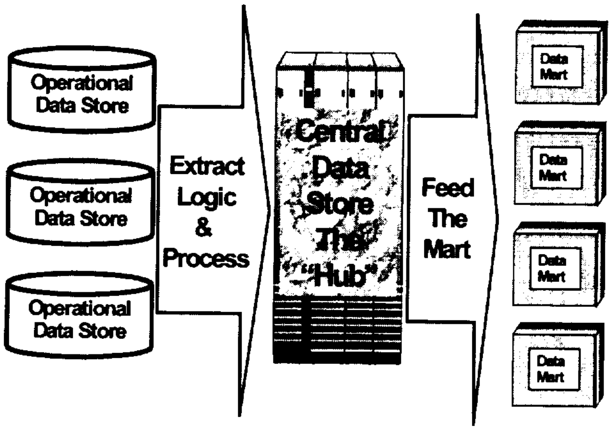


Figure 2 Hub-and-Spoke Architecture

StorHouse/RM is a commercial relational database system that support SQL-92 queries for data stored in the storage hierarchy. StorHouse/RM handles the data placement issues described in [CTZ 97]. We describe and analyze StorHouse/SM (Storage Manager) which is the storage manager used by StorHouse/RM. This paper concentrates on the relational database uses of storage. Other non-database centric uses of StorHouse/SM can be found at www.filetek.com. [CB 99] provides an analysis of (a) storage trends, (b) database trends, (c) commercial products and (d) AT&T's use of active storage hierarchy for their Call Detail Warehouse application.

This paper is organized as follows: Section 2.0 analyzes StorHouse/Relational Manager (RM), a database system that supports the diverse storage systems. Section 3.0 describes StorHouse/Storage Manager (SM) which manages diverse storage devices and storage media. We conclude describing future work.

2.0 StorHouse/Relational Manager (RM)

There are three major ways database systems use diverse storage hierarchies:

(1) Hierarchical Storage Management (HSM) is used to migrate data to optical storage. An HSM implementation strategy to place a marker in the database and move the value to optical. The query engine must understand where and how to access data (on disk or optical).

(2) Data Backup and Archival copies or moves data to tape or optical storage. Data backup is used to make a complete or partial copy of the database in order to restore databases in case of disasters or data corruption problems. Data archival writes the data to optical or tape (tape is most likely) and then removes the data from the database. A customer may decide to keep "N periods" of data and archive (i.e. migrate) data at the "N+1 period". In order to use the archive information, users must restore the data and then later delete it. This is a cumbersome process that for many practical reasons is rarely done.

(3) Atomic Data Store, as described earlier, is where all the historical data is stored on some or all of the storage hierarchy media.

StorHouse/RM

StorHouse/RM is a database system that supports SQL-92 queries against the storage hierarchy. This database system was designed and optimized to store (atomic) data on diverse media. StorHouse/RM

(RM for short) works in conjunction with StorHouse/SM (SM for short) to specifically administer the storage, access, and movement of relational data. SQL access is available from different platforms through a variety of industry-standard protocols. RM runs on Sun Microsystems Ultra Enterprise platforms.

Figure 3 shows that RM architecture and components are similar to all major commercial relational database systems. An RM database consists of the following:

- User table data that you store and access.
- Optional indexes—value, hash, and range—that locate the table data.
- Metadata that describes database components.

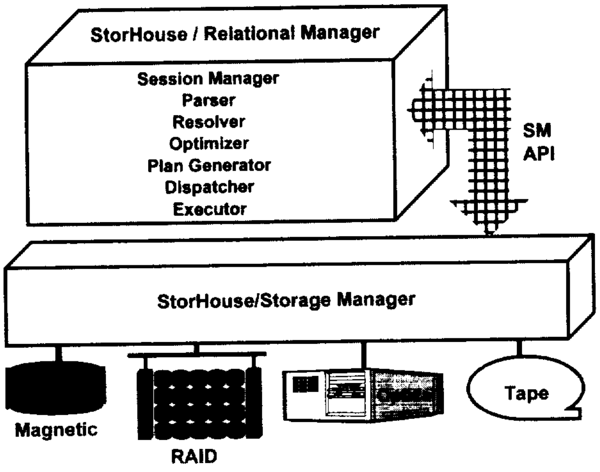


Figure 3: Relational Manager Architecture

RM databases (Figure 4) have both a logical and a physical structure. Logically, user tables and associated indexes reside in user tablespaces, and metadata resides in a system tablespace. Physically, user tables, indexes, and metadata are stored in SM files. RM user tables consist of one or more table segments. Each table segment is a separate SM file. Whether a user table is composed of one or multiple segments depends on the size of the user table and whether you later load more data into the table.

Extents

Table and index segments are RM files that can reside on any storage device in the RM storage hierarchy. These files are composed of different (Data, Definition and Map) extents. (1) Data Extent – holds user data and/or control data. (2) Definition (DF) extent - contains information necessary to retrieve the data. (3) Map Extent - is the high-level index that RM always reads first when doing index lookups. You can retain some or all extents in the magnetic disk performance buffer to enhance performance. For instance, you can hold the value index and hash index DF and Map extents on the performance buffer longer than the table Data extent to speed access to the data. To add more data into a table, RM would create a new table segment and corresponding value and hash index segments. Range indexes do not consist of index segments; they are stored in the Meta Data.

Value and hash indexes also consist of index segments. Each index segment is a separate RM file. For each value index on a table, there is always one value index segment associated with each table segment. For each hash index on a table, there is always one hash index segment associated with each table segment.

UNIX Database Files

RM database metadata and range indexes reside on and are managed by RM on the UNIX file system. Each system table, system table index, system table log, system table index log, and range index is a separate UNIX file.

User Tables

A user table is the basic unit of data storage in a RM database. User tables hold user-accessible data. Logically, RM user tables are like most RDBMS user tables; they consist of columns and rows of data. Each row contains data values conforming to the constraints of the columns that make up the row.

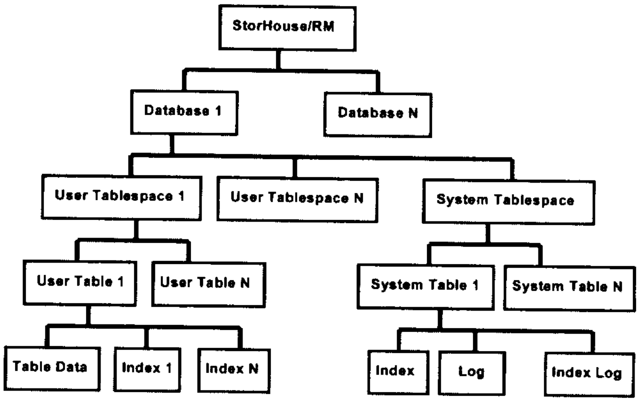


Figure 4: StorHouse/RM Database Layout

Physically, an RM user table is one or more files on the RM storage hierarchy. User tables can reside on any storage device in the hierarchy. The user tablespace defines the target storage device and the migration path through the hierarchy. For instance, you can store time critical data on RAID and then migrate that data to tape or optical as the data ages. The RM software automatically manages storage and migration based on your user tablespace parameters.

Indexes

Indexes provide efficient access to table data. You can create an index on a column or combination of columns in a user table. An index based on one column is a simple index. An index based on more than one column is a compound index. RM supports three index types—value, hash, and range.

Value index

Value indexes work best with queries that return multiple rows based on a range of values. A value index contains an ascending list of all the values in a column (or group of columns for a compound value index). For each column value, the index contains an index map to the table row containing that value. By searching the index rather than the table, then matching column values to row IDs, RM can more efficiently find requested table rows.

Hash index

Hash indexes work best with queries that return a specific record based on a specific value. A hash index is a two-part index based on an index map extent and a hit list that uses a proprietary RM algorithm to effectively locate individual table rows based on individual index values.

Range index

Range indexes are useful for user tables with multiple segments. A range index contains the lowest and highest column data values for each table segment for a user table. Instead of searching through multiple table segments, RM first looks at the range index to find the specific table segment with the requested data values. Then, RM might use any hash or value indexes to find a specific data value or range of values in the table segment.

User tablespaces

A user tablespace defines where table segments, hash index segments, and value index segments are stored on the RM storage hierarchy. It also sets attributes that influence storage management like backup and migration. When you create a user table, you assign it to a user tablespace. The table's segments and corresponding index segments then are stored according to the specifications of the user tablespace.

Allocating Storage

You allocate storage by identifying the volume sets and file sets for all table, hash index, and value index segments stored in the user tablespace. Whether you use multiple volume sets and file sets in a user tablespace depends on your data and your access and performance requirements.

Volume

A volume is a unit of media on which data can be recorded and read. RAID, magnetic disk, erasable and WORM optical disk and DLT cartridges are all examples of RM volumes, or media.

A **volume set (VSET)** is one or more physical volumes that are treated as a logical unit of storage. You use volume sets to control the physical grouping of files. A user tablespace identifies the volume sets that will contain the table, hash index, and value index segments for all tables and indexes in that tablespace. You can store table segments and associated hash and value index segments on the same volume set or on different volume sets.

A **file set (FSET)** is an area of storage within a volume set. Files are stored in file sets. Table segments and index segments are files. You can store table and associated hash and value index segments in the same file set or in different file sets.

For example, if you want to minimize volume mounts and don't need to manage the storage of table data and indexes in different ways, then you could allocate one VSET with one FSET for all components.

Or if you want to manage the storage of table data and indexes in different ways but remove them from RM at the same time, you could allocate separate FSETs in one VSET. Or if you want to manage the storage of table data and indexes in different ways and don't need to remove them from RM at the same time, you could allocate separate VSETs.

Performance and Backup Copies

A user tablespace contains a **Vulnerability Time Factor (VTF)** attribute that determines whether and when you'll create performance copies and/or backup copies of user table and index data. Performance copies reside on the RM magnetic disk performance buffer. Backup copies reside in primary file sets on the designated RM media.

A user tablespace contains an **Access Time Factor (ATF)** that works with RM migrate function to keep data that is most likely to be accessed in the RM performance buffer while maintaining a supply of free space. In a user tablespace, you can assign the same or different ATF values for table, hash index, and value index segments. This means you can retain index segments on the performance buffer longer than the data for faster query processing.

Metadata

Metadata are system components that RM creates and uses to manage a database. These components include system tables, system table indexes, system table logs, and system table index logs. Metadata is stored on the RM UNIX file system within a system tablespace.

System Tablespaces

For each database, RM creates a separate database directory on the RM server. This database directory, also called the system tablespace, is a structure that contains all of the system components for a specific database. Each system component is a separate UNIX file. All system tables have corresponding table logs. Some, not all, system tables have indexes.

System Tables

RM automatically creates a set of system tables for each database and stores them in the database's system tablespace. System tables contain information about a database. RM uses the system tables to record, verify, and conduct work. For example, RM updates various system tables when you create database components, and it reads system tables to verify that database components exist and that accounts are authorized to access them. Each system table is a separate UNIX file. Just like user tables, authorized users can query system tables by submitting a SELECT statement.

System Table Indexes

RM automatically creates system table indexes for specific system tables when a database is created. System table indexes are stored as UNIX files in the same directory as the system table files. Their operation is transparent.

System Table Logs

Each system table has a corresponding system table log that is used to recover changes to system tables. Before RM updates a system table, it first copies a "before image" of any record being updated to the system table log and then makes the change in the system table. If the transaction fails or is rolled back, RM copies the before image back to the system table, removing the change. If the transaction completes or is committed, RM empties the system table log.

System Table Index Logs

Each system table index has a corresponding log that is used to recover changes to system table indexes. The operation of the index log is the same as the table log.

3.0 StorHouse/Storage Manager (SM)

StorHouse/SM, FileTek's comprehensive HSM software, controls a hierarchy of storage devices comprised of cache, redundant array of independent disk (RAID), erasable and write-once-read-many (WORM) optical disk jukeboxes, and automated tape libraries. StorHouse/SM is also responsible for critical system management tasks, like data migration, backup, and recovery. StorHouse/SM provides system-managed storage that optimizes media usage, response time, and storage costs for each application. StorHouse/SM runs on Sun® Microsystems Ultra Enterprise Servers and comes standard with all StorHouse systems.

Some of the important functions of this storage management software include:

- Provide common view of all storage including magnetic disk
- Automatically deal with all storage hardware failures to avoid system down-time
- Provide record/RDBMS-page access methods
- Maintain meta data on magnetic disk
- Maintain indexes on magnetic disk or optical
- Order accessing so all accesses for a volume are performed with one volume mount
- Order tape accessing so it is serial

- Duplex data on separate libraries so there is no single point of failure
- Create copies of data for off-site storage
- Maintain volume directories on volumes for recovery and transport to other systems
- Where data is on both optical and tape use optical for direct and tape for serial accessing
- Cache most active data on magnetic disk
- Migrate data from optical to tape.

Future Work

This paper is based on commercially available products and real customers. Future on-going work includes federated databases and development of StorHouse/ORM (Object/Relational Manager) to support (multimedia) SQL-3 types and functions.

Conclusion

In this paper we described the need for active storage hierarchy. We described the tradeoffs, uses and potential uses that using tape, optical and disk storage can provide. We defined a new Data Warehouse concept, called Atomic Data Store, whereby applications exploit atomic (historic) data using a central data store Hub-and-Spoke architecture, used to feed data into Data Marts. We analyzed StorHouse/RM, which is an SQL RDBMS that stores and retrieves data from the storage hierarchy (using StorHouse/Storage Manager).

Acknowledgements

We thank FileTek's technical publications Sandy Cornfeld from whose award-winning documentation we freely borrowed the StorHouse/RM description.

References

[Arm 97] Armstrong, B., "Data Warehousing: Dealing with Growing Pains", Data Engineering, March 1997. Pp. 199 – 205.

[BZ 98] Charles Bontempo and George Zagelow, "The IBM Data Warehouse Architecture", CACM September 1998, Volume 41, No. 9, pp. 38 –48.

[CB 99] Cariño, F. and Burgess, J., "Lost in Active Storage Space - New Frontier for IT Managers", To appear in Intelligent Enterprise Magazine 1999

[CTZ 97] Christodoulakis, S., Triantafillou, P. & Zioga, F. "Principles of Optimally Placing Data in Tertiary Storage Libraries", VLDB '97, pp. 236 – 245.

[Gar 98] Gardner, S. R., "BUILDING the Data Warehouse", CACM September 1998, Volume 41, No. 9, pp. 52 – 60.

[GB 88] Gray, J. and Bitton, D., "Disk Shadowing", VLDB 88, pages 331 – 338.

[HS 96] Hillyer, B. and Silberschatz, A., "Random I/O Scheduling in Online Tertiary Storage Systems", SIGMOD 96, pp. 195 – 204.

[Imn 92] Inmon, W., "Building the Data Warehouse", QED Technical Publishing Group, Wellesley, Massachusetts, 1992.

[JM 98] Johnson, T. and Miller, E., "Performance Measurements of Tertiary Storage Devices", VLDB 98, pp. 50 – 61.

[RGF 98] Riedel, E., Gibson, G. and Faloutsos, C., "Active Storage For Large Scale Data Mining and Multimedia", VLDB '98, pages 62 – 73.

[VLDB99] Panel @ VLDB 99,"Active Storage Hierarchy, Database Systems and Applications – Socratic Exegesis, http://www.filetek.com/vldb.htm

Fault Tolerant Design in the Earth Observing System Archive

Alla Lake, Jonathan Crawford, Raymond Simanowith, Bradley Koenig
Lockheed Martin
1616 McCormick Drive, Upper Marlboro, MD 20774
alake@eos.hitc.com
jcrawfor@eos.hitc.com
rsimanow@eos.hitc.com
bkoenig@eos.hitc.com
tel: +1-301-925-0626
fax: +1-301-925-0651

Abstract

The Archive for the Earth Observing System (EOS) is one of the largest and highest data rate archives in the world. The EOS Archive is referred to as EOS Core System (ECS) and is a multi-site distributed data warehouse of Earth-oriented satellite images and science algorithms/reports. Its data holdings are projected to approach five petabytes by 2002. Each distributed site is referred to as a "Distributed Active Archive Center" or DAAC. The DAAC sites are being incrementally delivered with final deployment by the end of 2000. One of the sites, the EROS Data Center (EDC) in South Dakota, is receiving and archiving Landsat data in addition to the data generated by the instruments on the Terra satellite launched in December of 1999. Four of the DAACs will begin receiving Terra data in early 2000 [1].

The ECS archive architecture is based on a multi-site, distributed, client-server model. Its components are interdependent. As in any large and reasonably complex system robustness and ability of functional components to recover from faults is of great importance. In particular, ECS places heavy emphasis on data integrity and data capture robustness. This paper briefly describes the design of the hardware and software to insure the EOS data is captured and distributed in spite of faults. The description of hardware failover is confined to the Ingest Component design. The paper is intended as an introduction to the Poster Presentation material, and other components are discussed in the Poster Presentation itself.

1 Introduction

The overall fault recovery scheme in the ECS archive is designed to be a combination of the hardware server failover and software server recovery. The hardware server failover is operator initiated. It takes place in the event of a catastrophic failure of the hardware server itself or its associated network interface. Hardware failover to a secondary server can also be initiated as a planned maintenance or upgrade step. Failover is followed by the software server recovery for the ECS archive to continue operation. A software server recovery can also take place when the software fails, independently of any hardware faults. The hardware and software recovery designs are functionally independendent and are treated separately in this paper.

The hardware portion of a fault recovery design differs on a hardware subsystem by hardware subsystem basis using several different configurations, as appropriate, to meet the EOS mission objectives. In most cases fault handling requirements range from 2 hours to 24 hours depending on the subsystem. In practice, the down time must be minimized because of the impact on both the user community and processing of the data. The goal of the fault tolerant design is to reduce the down time to at most 15 –30 minutes per incident.

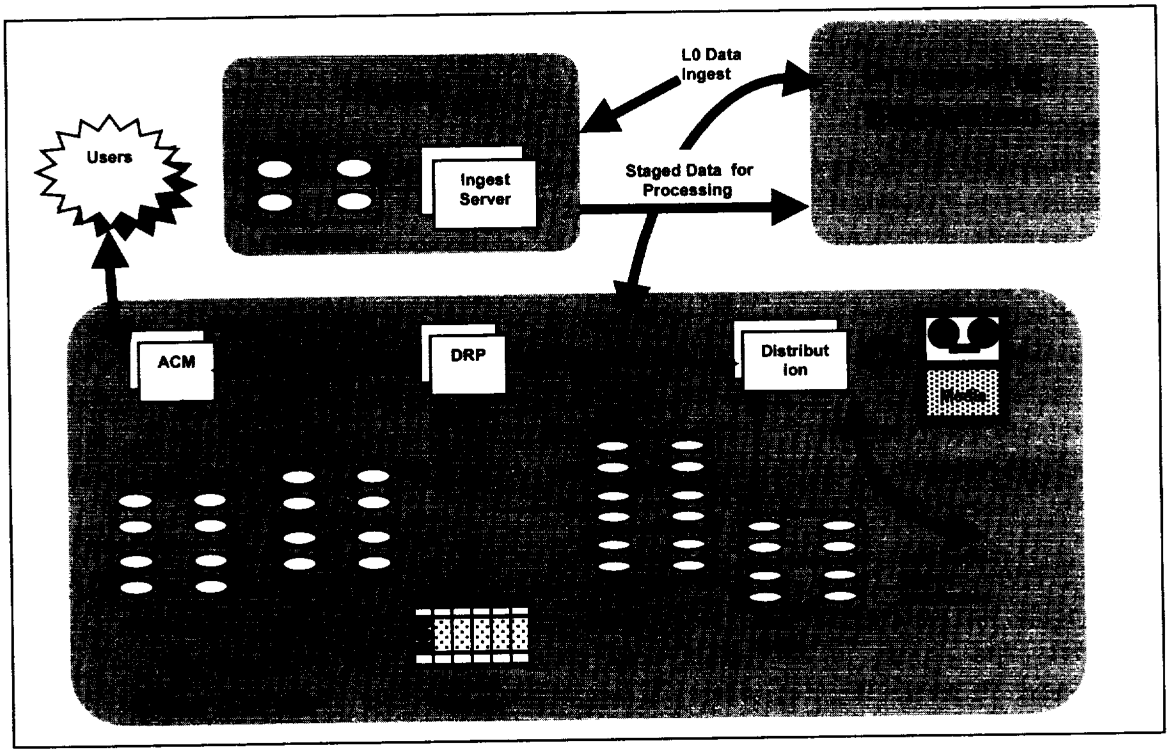


Figure 1. Hardware Configuration of the ECS Archive

Functionally, all external user electronic access to the system takes place via the Access Control Management (ACM) platforms, as illustrated in Figure 1, Hardware Configuration of the ECS Archive. The ACM is also where the ECS Data Server Metadata catalogue resides. In the ECS system a metadata catalogue indexes the total collection and points to files stored in silo-based archives.

The INGEST subsystem is responsible for data capture. Both the ACM and INGEST subsystems have the most stringent fault recovery requirements of 2 hours and a design goal of 15 minutes. Ingest hosts are used for Level 0 Instrument Data capture from the Front-End data capture facilities into the archive. A warm-standby pair configuration is used for the ACM and Ingest hosts. In this scheme failover to the secondary server is an operator-initiated event.

The Archive (DRP) hosts function as file servers connecting the rest of the system to the Nearline data holdings in the robotic silos. The hardware configuration of this component and aspects of its performance have been discussed at the March 1998 Sixth NASA Mass Storage Systems and Technologies Conference [2]. The DRP subsystem recovery requirement is 3 hours. A cluster "many-to-one" failover configuration is used for the DRP hosts. Once again, failover to the standby host is initiated manually. Once initiated, a portion of the process takes place automatically via execution of a series of scripts. Several steps within the failover procedure are manual, primarily the network router switchover. The ACM Data Base servers, the Ingest hosts and the DRP hosts platforms are at this time of Silicon Graphics Incorporated (SGI) Challenge[1] class servers. SGI Origin class servers will replace these as part of technology evolution during the life of the archive. The custom software for the Science Data Server in the ACM Hardware component resides on SUN platforms.

The Distribution component of the archive is responsible for the distribution of hard media to the users of ECS. The Distribution component has a recovery requirement of 2 hours. A load sharing configuration, allowing graceful throughput degradation in the event of failure, is designed for the Distribution hosts. The Distribution hosts are SUN Ultra servers.

More detailed descriptions of the software functions of the above system components can be found in other papers presented at this conference [1], [3].

2. Hardware Server Failover

Ingest is the only component which has been configured for hardware failover and tested in that configuration. All other components in the design have not been tested for the SGI Challenge class servers. The Ingest component consists of a pair of SGI Challenge servers. One of the servers is normally playing a primary role and the second one a secondary. Figure 1, Ingest Failover Pair, illustrates the hardware configuration.

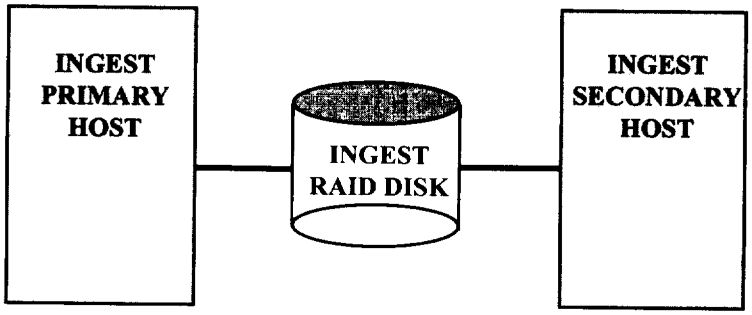


Figure 1. Ingest Failover Pair

Both hosts are physically connected to the Redundant Array of Independent Disk (RAID) in a "Dual-Bus/Dual-Initiator Configuration". Only one of the hosts is actively addressing the RAID at any one time. For simplifying the operations, the same host is always considered to be a "normally primary host". That is, in operation, that host is in the primary configuration at all times and is performing ECS ingest functions, except for the brief time periods for repair or upgrades. The secondary host may be used at the same time for other tasks or testing with two significant restrictions: 1) ECS functional configuration, both custom and Commercial Off The Shelf (COTS) must remain intact and in sync with the primary host, and 2) dual connected RAID is not available for use by the secondary host. Any attempt to address (read or write) dual connected RAID from the secondary machine may result in disk corruption. Switching control of the RAID from the normally primary to the normally secondary host is done through a failover procedure. Failback procedure is exercised when the RAID control is switched back from the normally secondary to the normally primary host. Both failover and failback involve 1) switching of ownership of RAID, 2) manual switching of all external network mounts and interfaces. Network switchover uses an *alias ip* mechanism. At the EDC DAAC, the only DAAC that currently uses HiPPI connection with Ingest, the HiPPI connection is also switched.

For the implementation of the Ingest Failover scheme, aside from the dual physical connection of the RAID, a number of specific changes must be made to the host system, network, and peripheral device configuration. Both the primary and the secondary hosts have an identical hardware complement, identically prepared RAID configuration, and their internal disks are loaded identically with the same complement of ECS COTS code.

3. Software Server Fault Recovery

The software for ECS is a C++ implementation using Distributed Computing Environment (DCE) [4] for process communications. The fault recovery software design relies on a combination of custom code supported with a relational database for request persistence and checkpointing, as well as COTS product features to allow network rebinding. Since all client-server interfaces are implemented using DCE RPC calls, it is crucial that lengthy processing operations not be repeated needlessly. At the first layer of software fault recovery, DCE rebinding is incorporated into the client interface classes. Rebinding permits automated detection and recovery of errors in DCE communications, including those introduced by network disruption. The second layer of fault recovery insulates against both client and server process failure ("crashes"). Long-running requests are checkpointed to the database, with all parameters and temporary data needed during processing. This checkpointing also provides a built-in queuing mechanism.

In the event of a client crash, the server, depending on the client type, takes one of the two possible actions. It abandons processing of the client's outstanding requests in favor of processing of requests from other client processes. Or, alternatively, it continues processing the client request and waits for the client to resynchronize to complete the transaction. In the event of a server crash, the client will attempt to rebind until the server is restarted, or until the client determines that an unacceptable period of time has elapsed. Resubmitted requests, whether through automatic rebinding or operator resubmission, are resumed from the last checkpointed state, thus eliminating redundant re-processing.

Upon restart, processes send a notification to the servers to which they are clients so that orphaned resources may be reclaimed.

Additional software fault recovery features provide for multiple server start "temperatures." Normal "warm start" processing permits resumption of request processing from the last checkpointed state upon client resubmission. "Cold start" mode terminates any in-progress requests and resets the persistence table to reflect an empty request queue. Resubmitted requests appear to the server as new requests. "Cold restart" mode provides a mechanism for "back-flushing" requests. Requests in progress are set to a failed state, and resubmission returns failure to the calling client.

4. Conclusion

Design of the failure recovery mechanisms in the ECS archive is an ongoing technical process as the system evolves following the computing technology trends. As an example, at the time of this writing, a replacement design for hardware failure recovery is being considered for implementation with the SGI's current generation Origin servers.

5. Acknowledgments

The authors thank Robert Howard of Raytheon Systems Corporation for the review of the text.

6. References

[1] Behnke, J., Lake, A., "EOSDIS: Archive and Distribution Systems in the Year 2000", Eighth NASA Goddard Space Flight Center Conference on Mass Storage Systems and Technologies, College Park, Maryland, March 2000

[2] Lake, A., "Performance Tuning of a High Capacity/High Performance Archive for the Earth Observing Systems Project", Sixth Goddard Conference on Mass Storage Systems and Technologies, College Park, Maryland, March 1998

[3] Crawford, J.M., "A Scalable Architecture for Maximizing Concurrency", Eighth NASA Goddard Space Flight Center Conference on Mass Storage Systems and Technologies, College Park, Maryland, March 2000

[4] OSF DCE User's Guide and Reference

[1] Here and subsequently, Silicon Graphics, Challenge and Origin are registered trademarks of Silicon Graphics, Inc., ULTRA is a trademark of SUN Microsystems, OSF is the trademark of the Open Software Foundation, Inc.

Connection of a Climate Model Database and Mass Storage Archive(s)

Michael Lautenschlager, Hannes Thiemann
Deutsches Klimarechenzentrum GmbH
Bundesstrasse 55
D–20146 Hamburg, Germany
data@dkrz.de
tel +49–40–41173–334
fax +49–40–41173–400

Abstract

An overview of an existing climate database which allows for storage of terabyte data volume is presented. Some features like the general architecture and the integration with an HSM are highlighted in more detail.

1 Introduction

The DKRZ (Deutsches KlimaRechenZentrum) is the central climate computing center for Germany. Numerical models for the coupled climate system were developed and integrated on the computer environment at DKRZ. The results are archived and disseminated for the climate research community in Germany and Europe as well as world wide. The mass storage archive contains currently 80 TByte (Aug. 99) of climate model data (90%) and of observational data (10%).

The TByte archive size is correlated with user access problems to climate model data. Data are basically accessible via a UNIX file system on the file server. No related catalogue information is available. Data are archived as time series of 4 dimensional data blocks (model raw data), whereas users access data as time series of 2 dimensional records (processed data, e.g. 2m temperature). Archived model data sets are stored on sequential storage devices, requested data are reloaded as complete files into the file server's disk cache. Then users have to transfer the files to their local client machines by FTP.

A database system, CERA (Climate and Environmental data Retrieval and Archiving system), has been developed in order to organize the data archive and to improve the users access to climate model data. Processed data and raw data are presently stored together with their description (meta data) in the CERA database [1]. Although not the entire archive is part of the database system, the currently existing CERA database with a size of 2 TByte contains more data than magnetic disks are available. Consequently parts of the database have to be stored on tapes yielding interaction problems between the database management system and the mass storage system. A hierarchical storage management including disks and tapes is presently not supported by commercial database management systems.

2 General architecture

The CERA data system itself is separated into three parts: data model [2], data hierarchy, and data processing. The CERA data hierarchy reflects three different levels of storage and access performance: metadata, processed climate data and climate model raw data. All parts of the data hierarchy are stored stored in tables of the CERA database.

The metadata contain the description of the climate data archive (data catalogue). The access should be as quick as possible.

The processed climate data contain frequently requested data structures. They are extracted and processed from the climate model raw data and are stored directly in the CERA database according to user requirements. The processed climate data can be accessed as BLOB entries (Binary Large Objects) in database tables. The processed climate data should be preferably available on hard disks near the database system in order to realize a performant access. The CERA data hierachy reflects the granularity in data warehouse architectures [3].

The third level in the data hierarchy contains the climate model raw data. Monthly accumulated model results are directly written into database tables as BLOBs. The data access is less performant than for processed climate data, because these data are stored on magnetic tapes under robot access. Only for a specific user request the raw data will be transferred from the file server into the database cache.

The basic problem with respect to storage of database files is to establish a flexible connection between database and mass storage system which allows for data migration and de–migration in dependence of user requests to the CERA database.

3 Database and Mass Storage Archive

Even the amount of data in the CERA database is too large to store the data exclusively on magnetic disks. It is only possible to store the actually used data sets on disk, the others have to be migrated to tapes of the mass storage system (MSS). The climate data in CERA are stored as BLOB's in database tables. Therefore the database management system (DBMS) has to interact with the mass storage system and the related archive system. At the DKRZ ORACLE is used for database management and the mass storage archive is administered by UniTree. The basic design assumption for DBMS is that all database files are randomly available on magnetic disks. A direct integration of a hierarchical storage management (HSM) is not available. The integration has to be developed individually in dependence of the used DBMS and the used archiving software.

Within ORACLE data are stored in tables and tables are summarized in tablespaces. The physical storage level is connected to the tablespaces (TS). When ORACLE is implemented on a (Unix) file system tablespaces are stored in one or more files, the database files (DBF). Tablespaces can be in different states; online, offline and read only are of interest with climate model data archiving [4].

- Online is the normal status. Data within tablespaces of this status are immediately accessible.

- Temporarily deactivated tablespaces are offline. DBFs belonging to such tablespaces are offline too. Objects residing in these tablespaces are not accessible. These tablespaces will be not opened at startup of the database.
- Read Only TS are tablespaces in which all objects can not be changed any more. The database system do not accesses these database files in write modus. This status is especially important in connection with backup and recovery. Read only tablespaces can be either online or offline.

When data are currently not needed it is possible to migrate DBFs if their affiliated tablespace is set offline. If data are requested by an application (e.g. user request) the DBF's have to be demigrated and the tablespace has to be set online again.

This mechanism has been automated at DKRZ by developing and installing a storage broker which acts as an interconnect between (database)–applications, the database system and the mass storage system. The storage broker is divided into interacting processes with some of them running inside the database, some of them outside using ORACLE's External Procedure Calls [4]. The main processes are:

- The main–storage broker accepts requests, checks space within database disk cache, allocates space and sends requests to other brokers.
- The make–space broker clears disk space based on dataset priorities in order to allow for de–migration.
- The de–migration broker de–migrates database files from mass storage system back to database disk cache.

The storage broker controls the migration and de–migration according to database requests and to that disk space which is available to the CERA database. Database request query optimization is realized in a disk cache area which is controlled by the broker and which is strictly separated from the archiving system. The migration/de–migration strategy is highly flexible. Priorities of requests are calculated online based on dataset, user and system load characteristics as well as on recorded database access statistics. These statistics may allow also for a pre–caching algorithm. A persistent interconnection between these processes allows for re–launch of requests even after database crashes.

Data extraction from and the data delivery into the mass storage system is realized by standard FTP in order to maintain independence from the archiving system.

4 Conclusions

The separation of the database migration disk cache from the standard HSM disk cache allows for an implementation of a database driven migration strategy and for ndependence from the connected HSM system. This two level cache approach is robust and provide a large degree of independence between RDBMS and HSM.

As the complete system is implemented on a Unix file system and not on raw devices standard file transfer mechanisms like 'ftp' and 'copy' can be used to connect the two disk cache areas. Therefore practically all HSM systems can be used.

References

[1] M. Lautenschlager and M. Reinke (Ed.), "Climate and Environmental Database Systems", pp. 197, Kluwer Academic Publishers, Boston 1997

[2] M. Lautenschlager, F. Toussaint, H. Thiemann and M. Reinke, "The CERA–2 Data Model", pp. 53, Technical Report No. 15, DKRZ, Hamburg 1998

[3] W. H. Inmon, "Building the Data Warehouse, Second Edition", pp. 401, John Wiley & Sons, Inc. 1996

[4] George Koch and Kevin Loney, "ORACLE8 – The Complete Reference", Orcale Press, pp. 1300, Osborne McGraw–Hill 1997

New Prospects for Electrostatic Data Storage Systems

Alexander N. Korotkov and Konstantin K. Likharev
State University of New York at Stony Brook
Stony Brook, NY 11794-3800
klikharev@notes.cc.sunysb.edu
tel +1-631-632-8159
fax +1-631-632-8774

1 Introduction

Magnetic data storage is a unique technology which has maintained an impressive bit density growth during the past few decades. It seems, however, that this growth will saturate somewhere near 100 Gbits/in^2, due to several fundamental factors – see, e.g., Ref. [2]. Recognition of this situation has triggered a wave of research in search of alternative technologies which would enable the current data storage scaling trend to continue after the magnetic systems run out of steam.

The natural candidate for an alternative technology is electrostatic data storage, where a digital bit is retained in the form of a minute (few-electron) charge of a small metallic grain or a group of grains. However, earlier attempts to implement such storage systems ran into several problems, including:

- the lack of sensitive and fast solid state electrometers for reading heads, and
- low speed of write/erase process.

During the past decade, the situation with the first problem has been radically improved due to the invention [3], demonstration [4], and gradual improvement (see, e.g., review [5]) of single-electron transistors (SET). These devices may serve as very good electrometers, with experimentally demonstrated charge sensitivity (at low temperatures) better than 10^{-5} $e/\sqrt{\mathrm{Hz}}$ [6]. Recently, the first operational room-temperature SET was demonstrated [7]. Though the charge sensitivity of room-temperature SETs has not yet been measured, theory [8] predicts that beyond the $1/f$ noise range it may be of the order of 10^{-7} $e/\sqrt{\mathrm{Hz}}$. This would be sufficient for readout of a few-electron signal at a very high (GHz-scale) speed.

Therefore the write/erase speed seems to have become the key problem for the practical introduction of electrostatic storage systems. We have carried out a theoretical analysis of various possible solutions to this problem, and run into what we believe is a very promising opportunity. Its brief description is the objective of this report.

2. Crested Tunnel Barriers

The most evident candidate for the write/erase process is Fowler-Nordheim tunneling, similar to that used in floating-gate memories [9]. The tunnel barrier should have negligible tunneling (corresponding to a charge retention time of a few years) for relatively low voltage V applied to the barrier by the stored charge. On the other hand, in the write mode the applied voltage should suppress the barrier to such an extent that tunneling current recharges the charge storing grains quickly. Simultaneously, this voltage should be small enough to avoid tunnel barrier degradation.

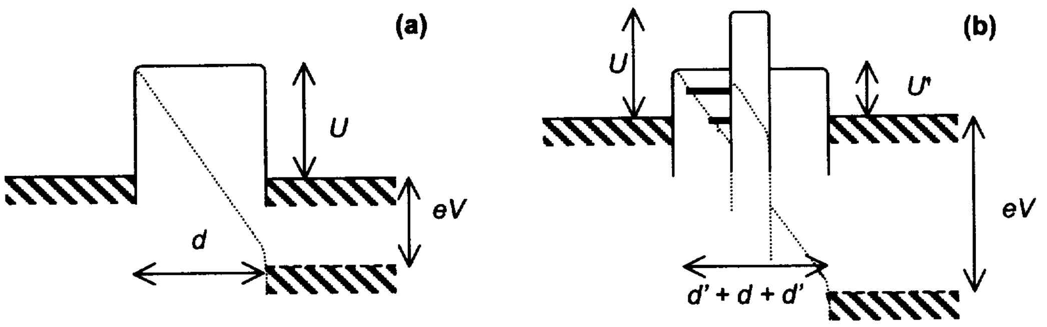


Figure 1. Conduction band edge diagrams of (a) - typical uniform barrier; and (b) - crested layered barrier. Dashed lines show the barrier tilting caused by applied voltage V. Thick horizontal lines in (b) show (schematically) the position of resonant electron subbands enabling resonant tunneling

The usual uniform tunnel barriers (Fig. 1a), made typically of silicon dioxide, cannot satisfy these two conditions simultaneously. Thin curves in Fig. 2 show the current density j and the grain recharging time scale $\tau(V) \equiv C_0V/j(V)$ as functions of voltage V for two typical values of SiO_2 barrier thickness. (The current has been calculated using the standard quasiclassical approximation, in the assumption of the isotropic and parabolic dispersion law for electrons both in the source conduction band and under the barrier. C_0 is the capacitance per unit area of the tunnel barrier.)

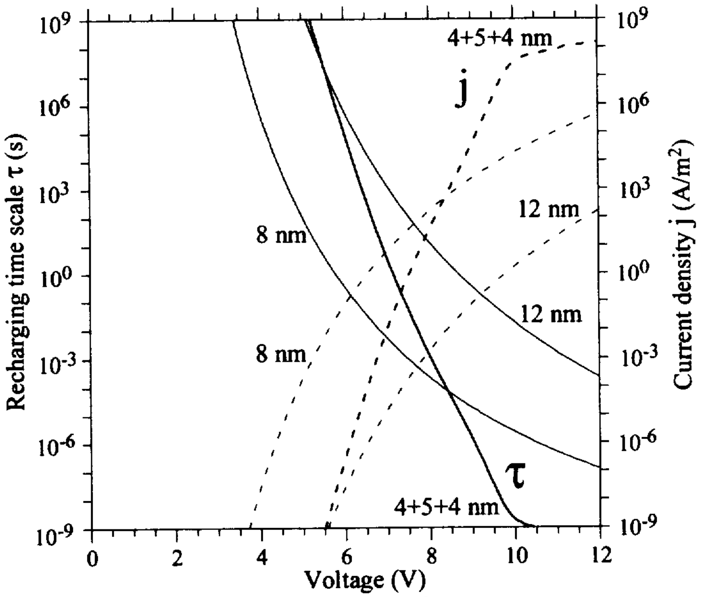


Figure 2. Tunneling current density j (in A/m^2, dashed lines) and the floating gate recharging time scale τ (in seconds, solid lines) for the barriers shown in Figs. 1a and 1b, as functions of applied voltage V, calculated using the quasiclassical theory. Thin lines: uniform barriers with parameters corresponding to $n^+Si/SiO_2/n^+Si$ ($U = 3.2$ eV, $m = 0.3$ m_0, $d = 8$ nm and 12 nm). Thick lines: trilayer crested barrier with parameters corresponding to $n^+Si/Si_3N_4/AlN/Si_3N_4/$ n^+Si ($U' = 2.0$ eV, $m' = 0.2$ m_0, $\varepsilon' = 7.5$, $d' = 4$ nm;

$U = 3.6$ eV, $m = 0.48\ m_0$, $\varepsilon = 8.5$, $d = 5$ nm). It is quite possible that crested barriers using other combinations of materials may have even better performance.
The results show that, for example, a 8-nm-thick barrier may provide a 10-year retention time (~3×10^8 s) for voltages below ~ 3.5 V, but the write time at the largest acceptable electric field $E_{\max}$ ~ 10 MV/cm [10] is above 1 μs. A change in the barrier thickness d to either side does not help (see, e.g., the results for d = 12 nm in Fig. 2), neither does a change in the barrier height. This relatively weak dependence of the barrier transparency on the electric field is due to the fact that the highest part of the barrier, closest to the electron source, is only weakly affected by the applied voltage: $U_{\max}(V) \approx U_{\max}(0)$ - see the dashed line in Fig. 1a.

Now consider a "*crested*" layered barrier (Fig. 1b) [11, 12]. Solid curves in Fig. 2 show that the current through this barrier changes much faster, so that a sufficiently long retention time at low voltages may be combined with 1-ns-scale write/erase at moderate electric fields (about 7 MV/cm). The physical reason for this dramatic improvement is that in the crested barrier the highest part (in the middle) is pulled down by the electric field very quickly: $U_{\max}(V) \approx U_{\max}(0) - eV/2$ – see dashed curves in Fig. 1b. As a result, the barriers may enable 1-ns-scale write/erase in electric fields as low as 7 MV/cm. In these low fields the barriers should have extremely high endurance, allowing a virtually unlimited number of write/read cycles.

3. ESTOR

Figure 3 shows the possible electrostatic data storage system (ESTOR) which combines the unique charge sensitivity of single-electron transistors and speed of recharging through crested barriers [11, 17]. The system includes a read/write head with an SET preamplifier loaded on a FET amplifier at a distance of a few microns. The data bits are stored as few-electron charges ($Q/e = n$ ~ 30) trapped in nanoscale conducting grains deposited on the top of a crested tunnel barrier. Since the charge is relatively large, and is stored in a few (~ 30) grains, their exact shape and location are not important, so the storage medium does not require nanofabrication: the grains of random size may be deposited, e.g., by metal evaporation.

Bit writing is achieved by the application of high voltage V_W in the moment when the head is passing over the specified location (at writing, $V_R = 0$, so that the SET is deactivated and works just as a single conductor delivering voltage V_W to the tip). The voltage suppresses the tunnel barrier, and inserts the charge into the group of grains. Nondestructive readout is achieved by the SET activation ($V_W = 0$, $V_R \neq 0$), turning the device into a sensitive electrometer [19].

Estimates show that with a 15-nm tip-to-substrate distance (typical for the advanced magnetic storage systems), the ESTOR system is capable of a density above 1 Tbit/in^2, i.e. at least one order of magnitude higher that the apparent upper density limit for magnetic systems [1]. The use of crested barriers may provide a very broad bandwidth of write/erase, up to 1 Gigabit per second per channel. The maximum reading speed, limited by the internal noise of the SET (with a signal-to-noise ratio of, say, 100), is even higher.

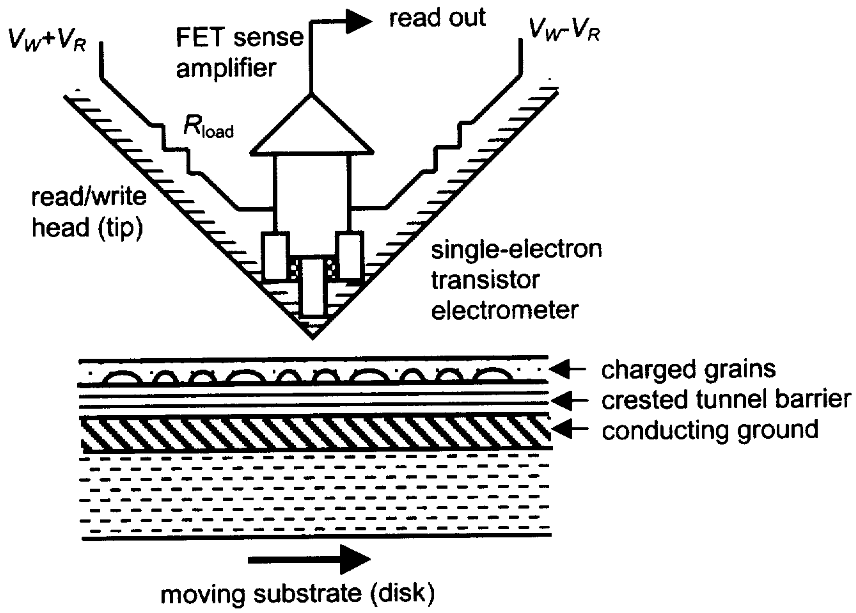


Figure 3. A possible ultradense electrostatic recording system ("ESTOR")

The recent experiments at Lucent Technologies [20] may be considered as the first step toward the implementation of the ESTOR. In these experiments a SET electrometer fabricated on a scanning probe was used for the detection of single-electron charges on Si and GaAs substrates. In these preliminary experiments, there was no FET amplifier close to the SET output, so that the available measurement bandwidth was very low. However, recently several groups have demonstrated the possibility of broadband SET/FET integration (so far, at low temperatures). It seems that the unification of these achievements with the progress in fabrication of room-temperature SET [7] and the standard mechanics developed for magnetic hard drives opens a straightforward way toward the implementation of practical ultradense electrostatic data storage systems.

References

[1] This work was supported in part by ONR/DARPA within the framework of Ultra Electronics and Advanced Microelectronics programs.

[2] A. V. Nurmikko and H. Goronkin. In: *Future Trends in Microelectronics*, ed. by S. Luryi *et al.* (New York: Wiley, 1999) 339-352.

[3] K. K. Likharev. *IEEE Trans. on Magn.* **23** (1987) 1142-1145.

[4] T. A. Fulton and G.D. Dolan. *Phys. Rev. Lett.* **59** (1987) 109-112.

[5] K.K. Likharev. *Proc. IEEE* **87** (1999) 606-632.

[6] V. A. Krupenin, D. E. Presnov, A. B. Zorin and J. Niemeyer. A very low noise single-electron electrometer of stacked-junction geometry, to be published in *Physica B* **284-288** (2000).

[7] J. Shirakashi, K. Matsumoto, N. Miura, and Konagi. *Appl. Phys. Lett.* **72** (1998) 1893-1895

[8] A. N. Korotkov, S. A. Vasenko, and K. K. Likharev. In: *Single-Electron Tunneling and Mesoscopic Devices*, ed. by H. Koch and H. Lübbig (Berlin: Springer, 1992) 45-60.

[9] *Nonvolatile Semiconductor Memory Technology*, ed. by W.D. Brown and J.E. Brewer (New York: IEEE Press, 1998).

[10] In stronger fields, the hopping ("Frenkel - Poole") conductance via deep localized states becomes essential, and eventually leads to the barrier degradation, sharply limiting the barrier endurance, i.e. the maximum number of write/erase cycles [9].

[11] K.K. Likharev. *Appl. Phys. Lett.* **73** (1998) 2137-2139. In that work, the effect of resonant tunneling via electron subbands (which are naturally formed in layered barriers in high electric fields – see Fig. 1b) has been ignored. More adequate calculations [13] have, however, confirmed the basic conclusions of Ref. 16; the resonant tunneling effect may be compensated by an increase of the side layer thickness.

[12] A positive effect of "graded" or "stepped" tunnel barriers on the speed of electron emission was noticed [14-16], and its use in floating gate memories has been suggested [14, 16], long ago. However, the authors of those works considered asymmetric triangular barriers. Though the injection characteristics of such barriers may be even better than those of symmetric crested barriers [11], this is only true for one current direction (say, "write"). The speed of the reciprocal process ("erase") is low, thus excluding the possibility of data storage applications.

[13] A.N. Korotkov and K.K. Likharev. *Appl. Phys. Lett.* 75 (1999) 2491-2493.

[14] T.H. DiStefano. *U.S. Patent #3,972,059* (1976); D.J. DiMaria. *J. Appl. Phys.* **50** (1979) 5826-5832.

[15] C.L. Allyn, A.C. Gossard, and W. Weigmann. *Appl. Phys. Lett.* **36** (1980) 373-375; R.J. Malik, T.R. AuCoin, R.L. Ross, K. Board, C.E.C. Wood, and L.F. Eastman, *Electron. Lett.* **16** (1980) 837-839.

[16] F. Capasso, F. Beltram, R.J. Malik, and J.F. Walker. *IEEE Electron. Dev. Lett.* **9** (1988) 377-379.

[17] A. N. Korotkov and K. K. Likharev. In: *Abstr. Of Int. Workshop on Comp. Electronics* (Tempe, AZ: U. of Arizona, 1995) 42; *VLSI Design* **6** (1998) 341-344.

[18] This proximity is necessary to reduce the interconnect capacitance which should be recharged through the high output impedance of the single electron transistor and hence to ensure high readout bandwidth.

[19] For this application, the randomness of the background charge, which is the largest obstacle on the way toward other digital applications of single-electron devices [3], is not important, since it may always be compensated by an additional gate voltage tuned to bias the SET into a point with maximum sensitivity.

[20] M.J. Yoo, T.A. Fulton, H.F. Hess, R.L. Willett, L.N. Dunklebrger, R.J. Chichester, L.N. Pfeiffer, and K.W. West. *Science* **276** (1997) 579-583.

Tertiary Storage Organization for Large Multidimensional Datasets

Sachin More[*] Alok Choudhary
Department of Electrical and Computer Engineering
Northwestern University
Evanston, IL 60028
{ssmore,choudhar}@ece.nwu.edu
tel +1-847-467-4129
fax +1-847-491-4455

1 Introduction

Large multidimensional datasets are found in diverse application areas, such as data warehousing [6], satellite data processing, and high-energy physics [9]. According to current estimates, these datasets are expected to hold terabytes of data. Since these datasets hold mainly historical and aggregate data, their sizes are increasing. Daily accumulation of raw data and jobs generating aggregate data from the raw data are responsible for this increase. Hence, estimates for the dataset sizes run into several petabytes. Though cost per byte as well as area per byte for secondary storage has been dropping, it is still not cost effective to store petabyte-sized datasets in the secondary storage [4].

Efficient storage organization for multidimensional data has been investigated extensively [8, 1, 5]. Chen et al [1] discuss organization of multidimensional data on a hierarchical storage system. The authors prove that the problem of efficient organization of multidimensional data on a one-dimensional storage system, such as tertiary storage, is NP-complete when arbitrary range queries are allowed. They present a five step strategy based on heuristics for the problem. Jagadish et al ([5]) investigated the problem of efficient organization of a data warehouse on secondary storage. The workload consists of a restricted set of range queries using hierarchies defined on the dimensions. They cast the problem as finding an optimal path through a lattice. They propose a dynamic programming based algorithm that determines how various dimensions are laid out.

We are not aware of any work that takes into consideration practical constraints like the order in which the data already exists or will be generated. Given an order in which data currently exists (or will be generated), and a limited amount of temporary storage space, we investigate issues in efficiently organizing multidimensional datasets on tertiary storage. We cast the problem as permutation of the input data stream using limited storage space. The rest of this document is organized as follows: The problem is formulated in Section 2. Section 3 describes our approach. In Section 4, we present performance results. Section 5 presents conclusions.

[*] Supported by DOE ASCI Alliance program under a contract from Lawrence Livermore National Labs B347875.

2 Background

Queries In a multidimensional dataset, each data item occupies a unique position in a n-dimensional hyperspace. A query selects a subset of the data items by selecting a subset of the domain in each dimension. A query is an instance of a *query type* [1]. A query type is a n-tuple whose values are drawn from $\{ALL, VALUE, ANY, RANGE\}$ for each dimension. ALL selects the entire domain of the dimension. $VALUE$ selects exactly one value from the domain. ANY is similar to $VALUE$, but choose all domain values with equal probability. $RANGE$ choose a set of values from the domain of the dimension. We assume that approximate execution probabilities of query types are known.

Native order and storage order A data source generates data items in a known order (e.g., in temporal order, when *time* is one of the dimensions). We call this ordering of the data items *native order.* Depending on the expected query types, this native order may not be the most efficient way to store the dataset. We call the order in which the data items are stored as the *storage order.* Data items need to be *permuted,* to transform the native order into storage order.

Storage model The storage model consists of secondary storage of size D pages and tertiary storage of size at least T pages, where T is the size of the dataset and $T > D$. The tertiary storage consists of a tape drive controlled by a robotic arm and a set of magnetic tapes. The data items generated by the data source temporarily reside on the secondary storage before they are stored on tertiary storage. The secondary storage pages can be viewed as temporary storage that is used to carry out data permutation.

3 Data clustering algorithm

Given native ordering of data items and the expected query workload, we derive a storage order that optimizes the I/O time of the workload subject to some constraints and assumptions [7]. The input data is read exactly once in the native order. We estimate the available temporary storage size to be of the order of the size of expected answer sets for queries in the workload. We use the knowledge about the expected query types and their execution probabilities to compute the storage order.

Though the storage order defines an ordering on data items, the process used to arrive at the order need not be a sorting process. Given a set of data items, it is important to identify the subsets of the data items that are accessed together. This is an important observation, since the success of transforming the native order into a storage order depends on amount of space available to carry out the transformation. We show that, given a dataset and a workload, there exist multiple storage orders that are equally good. For a given native order and amount of temporary storage space, only some of them are achievable [7]. In general, data items need to be clustered based on their coordinate values in a subset of dimensions. The clustering process needs to *match* coordinate values of the data items rather than *sort* the data items based on their coordinate values. This observation forms the basis of the data clustering process in this paper.

3.1 Computing affinity between data items

The actual process of matching data items uses a measure of *affinity* between data items. Affinity between two data items, I_i and I_j, is the probability that if I_i is accessed by a query then I_j is also accessed, and vice versa. Two data items that are always accessed together have an affinity equal to 1. Two data items that are never accessed together have an affinity equal to 0. We use the following formulae to compute affinity between two data items, I_1 and I_2, is $\sum_{Q_i=Q_1}^{Q_k}(p_i \times \prod_{D_j=D_1}^{D_n} \lambda_i^j)$. Where Q_i is the i^{th} query type and p_i is its execution probability. D_j refers to the j^{th} dimension. $Q_i(j)$ is the selection criteria of Q_i in dimension D_j. If $Q_i(j) = VALUE$ then v_i^j is the value parameter for Q_i from the domain of D_j. If $Q_i(j) = RANGE$ then r_i^j is the range parameter for Q_i from the domain of D_j. $I_1(j)$ and $I_2(j)$ are the coordinates of I_1 and I_2 in D_j. λ_i^j is calculated using the following table:

	λ_i^j
$Q_i(j) = ALL$	1
$Q_i(j) = ANY$ and $I_1(j) = I_2(j)$	1
$Q_i(j) = VALUE$ and $I_1(j) = I_2(j) = v_i^j$	1
$Q_i(j) = RANGE$ and $\lvert I_1(j) - I_2(j)\rvert \leq r_i^j$	1
otherwise	0

3.2 The heuristic approach to data clustering

[1] proves that a simpler version of the problem, one without temporary space constraint, is NP-complete. Hence, we use heuristics to design our algorithm and evaluate them experimentally. The generic algorithm inputs some data items in native order in each iteration. It also produces, in each iteration, some output data items in storage order. This process is repeated until all data items are output. A heuristic approach needs to answer the following questions:

1. If there are $d \leq D$ pages free in the temporary storage, how many data items to input in each iteration?
2. If there are k data items in the temporary storage how many and which data items to output in each iteration?

Greedy heuristic The greedy heuristic answers these questions as follows: *Input as many data items as possible to fill up the temporary storage during each iteration. Output data items only if the temporary storage is full.* While outputting, it chooses the data item that has maximum affinity to the last data item that was output. The rationale behind the greedy heuristic is as follows: It is important to keep the temporary storage filled to capacity, since that gives a wider choice to the output phase in choosing data items to be output, which will result in better decisions. Hence, the input phase should read in as many data items as possible, and output phase should output as few data items as possible. The greedy heuristic results in a simple and low complexity algorithm.

Look-back-m heuristic This is a generalization of the greedy heuristic. The greedy heuristic looks at only one data item, the one that it just output, to decide which data item to output next. The look-back-m heuristic remembers the last m data items that were selected for output. It uses a larger history to improve the quality of the solution. For each data item in the temporary storage, the algorithm computes its affinity to these m data items, called *backward affinity*. It outputs the data item that has the maximum backward affinity. Remembering the last m data item has the side effect of reducing the space that holds incoming data items. If M pages (output bin) are used to hold the previous m data items, then only $D - M$ pages (input bin) are available to hold the incoming data items. This reduces the choice available to select the next data item to be output.

Forward tuning During the execution of the Look-back-m heuristic, multiple candidate data items in the input bin can have same backward affinity. For example, when there are no data items in the output bin, in the initial phase of the algorithm, all data items in the input bin have zero backward affinity. The algorithm randomly chooses from candidate data items. The forward tuning technique uses data items in the input bin to improve the performance. For each data item in the input bin, it computes its affinity to the remaining data items in the input bin, called*forward affinity*. It outputs the data item having maximum forward affinity.

4 Performance results

We base our experiments on the Sequoia 2000 Storage Benchmark ([10]) for the results presented in this section. We use the national dataset accumulated over 100 half-months. This is a four-dimensional dataset. The dimensions are *time*, *band*, *X*, and *Y*, where X and Y are the two dimensions from the raster image. We make the following assumptions about the native order of the dataset: all the raster images are chronologically sorted, since they were captured in that order. Raster images for a half-month are not sorted in any particular order. The raster images are created in row-major order (that is Y changes faster than X in the image). Two query types with distinct access patterns are used. Instances of *Query Type 1* select all images belonging to a band. Each instance accesses $\frac{1}{5}^{th}$ of the dataset. Instances of *Query Type 2* select all images belonging to a half-month. Each instance accesses $\frac{1}{100}^{th}$ of the dataset. Based on these query types, we experimented with two kinds of workloads. *Workload 1* consists of majority (90%) of type 1 queries. *Workload 2* consists of majority (90%) of type 2 queries.

The results presented here compare the performance of our algorithms (*Look-back-m* and *Forward Tuning*) with three other algorithms. The naive algorithm, which we refer to as *Unoptimized*, preserves the native order while storing data. The *band-date* algorithm sorts data items on the *date* dimension, and within the date dimension they are sorted on the band dimension. The *date-band* algorithms reverses the inner and outer sort dimensions of the band-date algorithm. We also present the *best case* times, calculated by storing data for each query at the start of the media using as much data replication as necessary. The best case times represent a lower bound on the workload execution time.

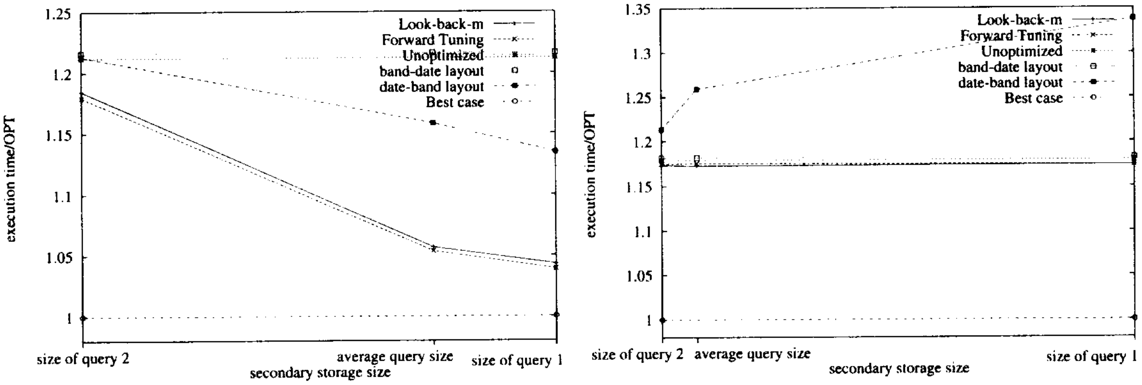
We use a simulator that uses the analytical model of Exabyte EXB-8505XL tape drive and EXB-210 tape library described in [2] and uses the *SORT* algorithm ([3]) for I/O scheduling during query execution. The time taken to execute the workload by an algorithm is plotted as a ratio of the actual execution time divided by *best case* time. We evaluate the performance of the algorithms by using temporary storage equal to the size of the instances of query type 1, query type 2, and the average query size of the workload.

Workload 1 Figure 1(a) shows that the performance of various algorithms is between 4 to 22 percent of *best case*. The wide range in performance is expected since the workload is dominated by larger queries that require major changes in the native order in order to be executed efficiently. The band-date scheme performs badly since it more or less preserves the native order; in fact its performance is worse than the *unoptimized* case. The date-band scheme improves its performance when given more temporary storage space. The date-band scheme favors query type 1, which dominates this workload. Hence, a closer transformation to the intended order (from native order) with increased amount of memory improves the performance. Our algorithms (*Look-back-m* and *Forward Tuning*) outperform other algorithms; moreover, they are within 6 percent of *best case* if the amount of memory available is more than average query size. The *Forward Tuning* algorithm does not provide any appreciable performance benefits over *Look-back-m*.

Workload 2 Workload 2 is dominated by query type 2, and the native order is favorable to it. The date-band scheme is not favorable to the majority of the queries; hence, we see a reduction in performance as the amount of available memory increases (increasing the closeness to a perfect date-band order) (figure 1(b)). Other algorithms perform similar to each other, the range of performance being around 18 percent of *best case*. The reason is that these algorithms tend to retain the native order; hence, the increase in the temporary storage size has no effect on the performance. None of the algorithms get close to *best case* because they do not take into account the size of the instance of the query types in the workload. A more sophisticated algorithm would have realized that 10% of the queries in the workload access approximately 66% of the total data accessed by all the queries in the workload and tried to optimize the storage layout for the *minority* query type (query type 1). Knowledge about the amount of data accessed by an instance of a query type requires advance information about the domain of each dimension. The algorithms presented in this paper were designed to work without such knowledge.

5 Conclusions

This paper investigates issues in efficient tertiary storage organization for multidimensional datasets. We show that the order in which the dataset is generated (or currently stored) affects the storage layout decisions due to limited amount of temporary storage available for data reorganization. We further show that efficient storage layout can be designed in this situation by considering data items that are accessed together rather than sorting the data items based on their coordinates. The experimental results have shown our techniques, based on heuristics, to be effective. They also reveal that when taking decisions about data



(a) Workload 1

(b) Workload 2

Figure 1: Performance Results

layout, one must consider the amount of data accessed by different queries in the workload besides the query characteristics and their execution frequencies.

References

[1] L. T. Chen, R. Drach, M. Keating, S. Louis, D. Rotem, and A. Shoshani. Efficient organization and access of multi-dimensional datasets on tertiary storage systems. *Information Systems*, 20(2):155–183, 1995.

[2] B. K. Hillyer, R. Rastogi, and A. Silberschatz. Scheduling and data replication to improve tape jukebox performance. In *Proceedings of the 15th International Conference on Data Engineering, Sydney, Austrialia*, pages 532–541, 1999.

[3] B. K. Hillyer and A. Silberschatz. Scheduling non-contiguous tape retrievals. In *Proceedings of Sixth NASA Goddard Conference on Mass Storage Systems and Technologies and Fifteenth IEEE Mass Storage Systems Symposium, University of Maryland, College Park, Maryland*, March, 1998.

[4] B. Inmon. The Role of Nearline Storage in the Data Warehouse: Extending your Growing Warehouse to Infinity. Technical white paper. StorageTek.

[5] H. V. Jagadish, L. V. S. Lakshmanan, and D. Srivastava. Snakes and sandwiches: Optimal clustering strategies for a data warehouse. In *Proceedings ACM SIGMOD International Conference on Management of Data, Philadephia, Pennsylvania, USA.*, pages 37–48. ACM Press, 1999.

[6] R. Kimball. *The Data Warehouse Toolkit*. John Wiley and Sons, Inc., 1996.

[7] S. More and A. Choudhary. Tertiary Storage Organization for Large Multidimensional Datasets. Technical Report CPDC-TR-9911-018, Center for Parallel and Distributed Computing, Northwestern University, November 1999.

[8] S. Sarawagi and M. Stonebraker. Efficient organization of large multidimensional arrays. In *Proceedings of the Tenth International Conference on Data Engineering, Houston, Texas, USA.*, pages 328–336. IEEE Computer Society, 1994.

[9] A. Shoshani, L. M. Bernardo, H. Nordberg, D. Rotem, , and A. Sim. Storage management for high energy physics applications. In *Computing in High Energy Physics*, 1998.

[10] M. Stonebraker and J. Dozier. SEQUOIA 2000: Large Capacity Object Servers to Support Global Change Research. Technical Report SEQUOIA 2000 Technical Report No. 1, Electronics Research Lab, March 1992.

Ferroelectric Molecular Optical Storage Nanotechnology

Michael E. Thomas
Colossal Storage
39224 Guardino Dr. Ste 212
Fremont, California
Tel 510 –794 – 3592
fedrive@hotmail.com
http://members.xoom.com/methomas/colossal.htm

Abstract

Colossal Storage has invented new ways of non - contact reading and writing with non destructive reading of information to a ferroelectric molecule.

1. Introduction

The Colossal FE Optical Drive density of 40 gigabits/sq.in. up to 500 gigabits/sq.in.[1] A comparison with harddrives of today is around 4 gigabits/sq.in. maxing at ~40 gigabits.[5] With optically assisted drives maxing at ~45 gigabits/sq.in. and contact recording AFM, STM, SPM or SFM, i.e. atomic force microscope and their derivatives, maxing practically out at about ~300 gigabits/sq.in..

2. Mywork

Colossal Storage uses the Einstein/Planck Theory of Energy Quantum Electrons to control molecular properties by an atoms electron movement/displacement. [6]
The Colossal Storage FeDrive - FeHead Semiconductor Integrated Optical Read / Write Head will use Ultraviolet/Blue laser diodes with Voltage transducer to write, and UV/Blue laser diode and Nanooptical transistor or Nanofloating gate Mos Fet to read.

2.1 Ferroelectric Molecular Optical Bits

The peripheral drive uses an ultra-violet or deep blue light source with an applied Electric field orientation transducer for writing. Reading is done by a second deep blue Or ultra-violet light source that is reflected off of the ferroelectric perovskovite molecule Surface to a nanoopto photo diode that is able to detect small changes in the diffraction of the Ultra-violet or deep blue light from the ferroelectric perovskovite molecule.[4]
Writing is done when the output of the ultra-violet or deep blue light source emits photons and ferroelectric molecules absorb the photons energy creating electron movement from the valence orbit to the conduction orbits of the ferroelectric molecule. When the applied field has a positive voltage potential the electrons move towards the transducer and vice versa for a negative potential. When the ultra-violet light source and applied fields are both turned off the ferroelectric molecule stays in the orientated direction and stores the random electric field positive/negative potential, i.e. a molecular or atomic switch, which also causes the ferroelectric molecule to physically elongation or shrink up to 1.5%. The stored electric field difference (voltage) of the ferroelectric molecule is permanently changed until ultra-violet or deep blue laser light and the applied field are turned on again to reorientate the direction of the potential difference. The dipoles electrical polarity of the ferroelectric molecule physically changes the transmiscivity, diffraction, surface morphology/topography, opacity, and reflection characteristics of ultra-violet or deep blue light on the ferroelectric molecule.[2][3]

Extremely small laser spots of 300 angstroms and less can be written and read using integrated optical head structure with densities of 40 gigabits sq.in. to 500 gigabits sq.in. being realized.[8] One method of reading is done with a second much lower Quantum energy ultra-violet or deep blue light source and a photo transistor or diode are used to detect differences in the diffracted photons of the ultra-violet or deep blue laser light being reflected back from the surface of the ferroelectric perovskovite molecule into the photo diode or transistor. Second method of reading is done by a floating gate mosfet transistor that is able to detect small changes in electric lines of force of the ferroelectric molecule. The electrostatic field (electric lines of force) from the ferroelectric molecule is sensed by the read mosfet transistor. The read voltage output is the recorded data in the ferroelectric molecule and is equal to the VCC of the floating mosfet transistor plus or minus the detected electrostatic field strength (electric lines of force) of the ferroelectric molecule. The read mosfet transistor is a source follower that does not destroy the stored electric field/voltage potential difference of the ferroelectric molecule. Third method of reading is done by yet another interesting variable by a second deep blue or ultra-violet light source which cause electrons of the ferroelectric perovskovite molecule dipoles to jump from one orbit to another. Niels Bohr Atom Postulates states, light excited electrons will stay in their higher energy orbits, UV or deep blue light with specific frequency and quantum energy excite the electrons of ferroelectric molecules into higher valence orbits and fall back to the normal lower energy orbits when the UV or deep blue light source is removed.[9] The stored internal dipole position (remnant displacement of central atoms - remnant polarization) further amplifies any higher orbit electron electrical field potential either positive or negative depending on the dipole position in the ferroelectric molecule and the distance from the UV or deep blue integrated read/write head. A mosfet nanotransistor that is able to detect small changes in the electrical field potential of the ferroelectric molecule when ultra-violet or deep blue light source is focused on the ferroelectric perovskovite molecule. Removal of the second UV light source (Quantum energy is characterized lower - not to induce electron movement into the conduction band) leaves the ferroelectric molecule in its initial electrical field stored state. The stored electrical field potential of a ferroelectric molecule can be made to represent at least four electrostatic field states equal to binary information.[10]

2.2 Ferroelectric Molecular Optical Read / Write Head

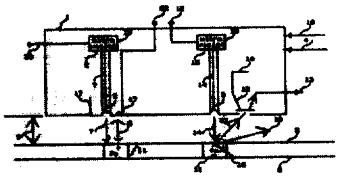


FIGURE 1

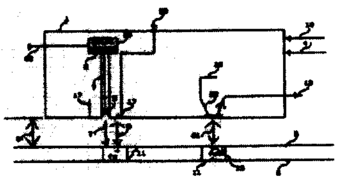


FIGURE 1

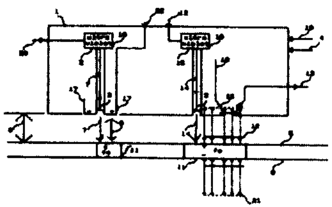
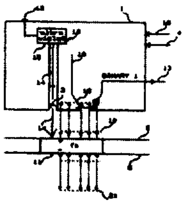
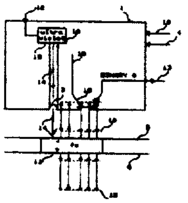


FIGURE 1



READ PART OF HEAD
FIGURE 2



READ PART OF HEAD
FIGURE 3

3. Related Work

Optical disk drives of today use laser light and a wide array of objective, Polarizing, and newly invented solid immersion lens (SIL). Laser light and Photon characteristics have allowed data storage peripherals to store enormous amounts of data. Sometimes the data written could only be written once, on magno-optical drive that data could be rewritten a limited amount of times by raising the temperature of the entire track and thereby causing an erasure of data. The latest means for increasing areal densities is done by a multitude of lens arrays finally feeding into a solid immersion lens (SIL). A focused infrared spot is obtained at the base of the SIL head (Terastor(Quantum)(Imation/3M). (Quinta) (Seagate) (Read-Rite) very small aperture lens (VSAL) method of technology for ferroelectric photon optical storage uses the BRAGG effect and a contact electrode. Quinta, Siros Technology, IBM, and Ioptics place single or multiple layers of recording magnetic media within a fraction of a wavelength distance from the SIL or VSAL head base, and by using a inductive transducer cause the electrons of the ferromagnetic material to take on a (clock wise rotation)-north magnetic polarity or (counter clock wise rotation)-south magnetic polarity. When a infrared photon of the right energy level hits the ferromagnetic electrons it is reflected, whereby the infrared photon takes on a light polarization property, which can be measured, the KERR effect. In contrast, the Thomas Colossal semiconductor integrated optical head is able to produce much smaller spots than infrared-based storage devices. The ferroelectric molecules not only have small size, fast switching speeds, but can store voltage much like a variable voltage battery allowing for bit voltage data compression schemes, increasing densities even further, allowing further advances into holographic storage research. Ferroelectric molecular write activity is influenced by the introduction Of ultra-violet or deep blue laser light, Einstein/Planck Theorem of Energy Quantum. An induced electrical field further alters the ferroelectric molecular Materials properties such as conductivity and electrical properties. Removal of the light source and induced electric field leave the ferroelectric molecule in an altered electrical state potential which is non-volatile. A second much lower Quantum energy ultra-violet or deep blue light source and a photo transistor or diode are used to detect differences in the diffracted photons of the ultra-violet or deep blue laser light being reflected back from the surface of the ferroelectric perovskovite molecule into the photo diode or transistor.

4. Future Work

The applications for the Colossal Read/Write head are still evolving and encompass much more than just data storage and fast random image xerographic replication. Colossal Storage is also working on holographic concepts for future storage peripheral products.[7] The Colossal Storage FE Semiconductor Read/Write head for Ferroelectric Molecular Electrostatic Field Random Reorientation can be used for many more applications than data storage, examples might be, high speed imaging and offset printing, lithography, copiers, and printers. Future integrated circuits could be made and verified that have ferroelectric wiring 1 molecule wide with the ability to polarize the wire for new switching, molecular optical wire, logic state definitions, and I/O Data Transfers states. Ferroelectric interconnects can do it cheaper, with less power, and in much higher densities.

5. Conclusions

The Colossal Storage FE Optical Drive will offer symmetrical infinite double sided disk or tape non-destructive read and writes for the retention of data storage for ten-years or more. The Colossal FE Optical Drive density of 40 gigabits/sq.in. up to 500 gigabits/sq.in. The Colossal Ferroelectric Molecular Optical NanoTechnology Drive will be able to hold more data than any other type of drive and will deliver data much faster.

Bibliography

[1] Writing on the Fringe - Interfering electrons could lead to atomic data storage Scientific American *October 1995, p. 40*
Michael W. Noel and Carlos R. Stroud of the University of Rochester

[2] *In Situ* Measurements of Electric Fields at Individual Grain Boundaries, D. A. Bonnell, B. Huey, D. Carroll, Solid State Ionics 74 (1994) 35-42.

[3] Measurement of the Amplitude and Phase of a Sculpted Rydberg Wave Packet T. C. Weinacht, J. Ahn, and P. H. Bucksbaum Phys. Rev. Lett. 80, 5508 (22 June 1998).

[4] Study on Photoferroelectric Ceramics of High Performance, Kyushu National Industrial Research Institute, AIST, MITI, Dr. Tadahiko Watanabe

[5] IBM's magnetoresistive and giant magnetoresistive head technologies enable data storage products with the industry's highest areal densities. By Jim Belleson, IBM Storage Systems Division, & Ed Grochowski, IBM Almaden Research Center.

[6] *Laboratoire de Céramique* (LC) - Matériaux , Prof. Nava Setter, Research Activities 1996/7 in ferroelectrics.

[7] Persistent holographic recording in doubly-doped lithium niobat crystals
Ali Adibi, Karsten Buse, Demetri Psaltis , Nature Vol 393, pp 665-668, June 98.

[8] Laser Interactions with Polymers Surface modifications, patterning, periodic structures, coatings Dr. E. Arenholz, Dipl. Phys. S. Klose, A. Kirchebner, Ch. Ortwein, Johannes Kepler Universitat Linz.

[9] Theoretical background of the optical charging spectroscopy method used for investigation of trapping levels, I. Pintilie, L. Pintilie, D. Petre, C. Tivarus, T. Botila, Applied Physics Letters, 73(2), 1685-1687, (1998).

[10] Ferroelectric domain reversal of a photorefractive crystal for bit-oriented three-dimensional optical memory, Masaki Hisaka, Kawata Lab, Department of Applied Physics, Graduate School of Osaka University.

Theory of metal insulator transistion in strongly correlated electron systems, M. Gulacsi and K.S. Bedell, Integrated Ferroelectrics, 1995, Vol 6, pp 265-279.

Effects of Laser radiation on Photo-conductivity in PZT thinfilms, Li Li and Chhiu-Tsu Lin, Integrated Ferroelectrics, 1995, Vol 7, pp 33-44.

Defect chemistry model of the ferroelectric electrode interface, Ciaran J. Brennan Integrated Ferroelectrics, 1995, Vol 7, pp 93-109.

Effects of optical illumination of fatigued lead zirconate titanate capacitor, C.R. Peterson, S.A. Mansour, and A. Bement Jr. Integrated Ferroelectrics, 1995, Vol 7, pp 139-147.

Light scattering from sol-gel processed PZT thin film, M.B. Sindari, D. Dimos, B.G. Potler Jr., and RW Schwartz, Integrated Ferroelectrics, 1995, Vol 7, pp 225-236.

Photo electric effects in BaTiO3 crystals, WL Warren, D. Dimos, Integrated Ferroelectrics, 1995, Vol 6, pp 237-246.

Neuron mos multiple valued memory technology for inteligent data processing, Ria Au, Takeo Yamashita, IEEE 1994 Intl., ISSCC 94, Vol 37, pp270-271.

A Stack Model Based Replacement Policy for a Non-Volatile Write Cache

Jehan-François Pâris
Department of Computer Science
University of Houston
Houston, TX 77204
+1 713-743-3350
FAX: +1 713-743-3335
paris@cs.uh.edu

Theodore R. Haining[†]
Darrell D. E. Long
Computer Science Department
Jack Baskin School of Engineering
University of California
Santa Cruz, CA 95064
+1 831-459-4458
FAX: +1 831-459-4829
haining@cse.ucsc.edu
darrell@cse.ucsc.edu

Abstract

The use of non-volatile write caches is an effective technique to bridge the performance gap between I/O systems and processor speed. Using such caches provides two benefits: some writes will be avoided because dirty blocks will be overwritten in the cache, and physically contiguous dirty blocks can be grouped into a single I/O operation. We present a new block replacement policy that efficiently expels only blocks which are not likely to be accessed again and coalesces writes to disk. In a series of trace-based simulation experiments, we show that a modestly sized cache managed with our replacement policy can reduce the number of writes to disk by 75 percent and often did better. We also show that our new policy is more effective than block replacement policies that take advantage of either spatial locality or temporal locality, but not both.

1 Introduction

As processors and main memory become faster and cheaper, a pressing need arises to improve the write efficiency of disk drives. Today's disk drives are larger, cheaper, and faster than they were 10 years ago, but their access times have not kept pace. The microprocessors of today have a clock rate 50 times faster than their predecessors of 10 years ago. At the same time, the average seek time of a fast hard disk is at best between one half and one third of its predecessors from the same period. Some technique must be found to bridge the performance gap if I/O systems are to keep pace with processor speed.

[†] Supported in part by the Office of Naval Research under grant N00014-92-J-1807 and by the National Science Foundation under grant PO-10152754.

The effects of this huge write latency can be reduced by delaying writes indefinitely in non-volatile memory before being sent to disk [5]. The longer that data is held in memory, the more likely it will be overwritten or deleted, reducing the necessity for a write to disk. It is also more probable that data in the cache can be grouped together to yield larger, more efficient writes to disk. Therefore, a non-volatile write cache can substantially decrease the number of reads and writes actually serviced by the disk. This substantially reduces the amount of disk latency by eliminating some of the time necessary to reposition the disk head for each write.

A cache replacement policy is required to manage such a cache. Any cache replacement policy must control two things, namely which entities to expel from the cache (the so-called *victims*) and when to expel them. The latter is very important when the processes accessing the storage cannot be delayed. In this case, any write occurring when the cache is full will *stall* and must wait while victims are being cleaned to the disk. If the selection of these victims is not performed carefully, blocks recently written into the cache will be flushed to disk. Once flushed to disk, the cleaned blocks can be reused and overwritten as additional writes are made. If overwritten, the data from victim blocks will not be present in the cache even though temporal locality dictates that they are the most likely to be accessed again.

The cost of writing from the cache to disk is an important factor in selecting victims. Each write operation incurs a penalty due to seek time and rotational delay. Single blocks in the cache are very expensive to reclaim; each requires a write operation. Groups of blocks that can be coalesced into larger contiguous segments are prime candidates because they can be written in a single I/O operation.

We propose a block replacement policy that is segment-based. To simplify the presentation of our policy, we define a segment as a set of contiguous blocks located on the same track. By using a track-based approach, cost is associated with one seek of the disk head and one rotation of a disk platter. This has the advantage of making the cost of writing an individual block inversely proportional to the size of the segment to which it belongs.

Our replacement policy is based on the following three observations:

1. Writes to blocks in the cache exhibit spatial locality: blocks with contiguous disk addresses tend to be accessed together,

2. Writes to blocks in the cache also exhibit temporal locality: the probability that a block in the cache will be accessed again is a decreasing function of the time interval elapsed since it was accessed last, and

3. The curve representing the hit ratio of the cache as a function of its size exhibits a knee: once a given minimum cache size is reached, further increases of its size lead to much smaller increases in the hit ratio (see Figure 1).

Based on these three reasonable assumptions, we develop a model of cache behavior that divides segments into *hot* and *cold* groups. Using this concept of hot and cold groups, we present a new cache replacement algorithm. The model and algorithm are described in §2. We experimentally test the algorithm using a trace-based simulation. Experimental observations and results are found in §3. Section 4 discusses related work. The final section summarizes our major results.

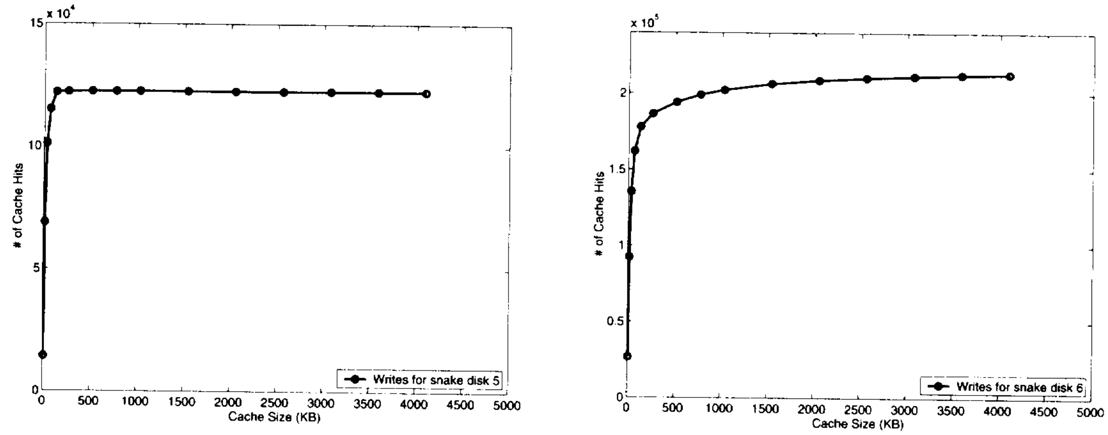


Figure 1: Simulated hit ratios as non-volatile write cache sizes increase for the two disks used in our experiments.

2 Cache Behavior

Given a cache exhibiting the properties of temporal locality, spatial locality, and a diminishing benefit to hit ratio described above, consider the m segments in S residing at any given time in a write cache. Assume that they are sorted in LRU order so that segment S_1 is the most recently referenced segment. Let n_i represent the size of segment S_i expressed in blocks. If accesses to segments in the cache follow the LRU stack model, each segment has a probability p_i of being referenced next. This probability will decrease with the rank of the segment i.e. $i < j$ implies $p_i > p_j$. Note that p_i represents the probability that keeping segment S_i in the cache will avoid a cache miss at the next disk write.

The contribution of each block in segment S_i to the expected benefit of keeping segment S_i in the cache is given by the ratio p_i/n_i. It makes sense to keep all the segments with the highest p_i/n_i ratios in the cache because this strategy makes the most efficient use of cache space. Conversely the segment with the *minimum* p_k/n_k ratio should be expelled. The blocks of that segment have the lowest probability of avoiding a cache miss during the next write.

Using these probabilities, we partition the segments residing in the cache into two groups. The first group contains segments recently written into the cache; these are the most likely to be accessed again in the near future. These *hot* segments should remain in the cache. The second group contains the segments not recently accessed and therefore much less likely to be referenced again. These *cold* segments are all potential victims for the replacement policy.

We identify these two groups of segments based on the knee in the curve representing the hit ratio of the cache as a function of its size. Let s_{knee} be the size in blocks of the cache at the knee and let C_j be the sum of the sizes of the first j segments in the LRU stack ordering of all segments in the stack:

$$C_j = \sum_{i=1}^{j} n_i.$$

The hot segments are the k most recently referenced segments that could fit in a cache of size s_{knee}, that is, all segments S_i such that $i \leq k$ where k is given by:

$$\max\{j \mid j \geq 1 \text{ and } C_j \leq s_{knee}\}.$$

All cold segments are assumed to be good candidates for replacement. We infer from the hit ratio curve that the p_i/n_i ratios for cold segments differ very little from each other. We would expect to see a greater increase in hit rate where the hit rate is nearly constant past the knee otherwise. Therefore the most efficient choice is to clean the largest segment in the cold region. This cleans the most cache blocks for the cost of one disk seek and at most one rotation of the disk platter.

A replacement policy that never expels segments until the cache is full can often cause writes to stall for lack of available space in the cache. This can be avoided by setting an upper threshold on the number of dirty blocks in the cache to force block replacement to begin. This clean space, say 10 percent of the cache, is able to absorb short-term bursts of write activity and prevent stalls. Our cache replacement policy then has two thresholds: one to determine when replacement should begin in order to keep a minimum amount of clean space, and one to determine when it ends based on the location of the knee.

The algorithm to select the segment to be expelled can thus be summarized as follows:

1. Find s_{knee} the size of the cache for x-value of the knee of the hit ratio curve.

2. Order all segments in the cache by the last time they were accessed: segment S_1 is the most recently accessed segment.

3. Compute successive $C_j = \sum_{i=1}^{j} n_i$ for all $C_j \leq s_{knee}$.

4. Let $k = \max\{j \mid j \geq 1 \text{ and } C_j \leq s_{knee}\}$.

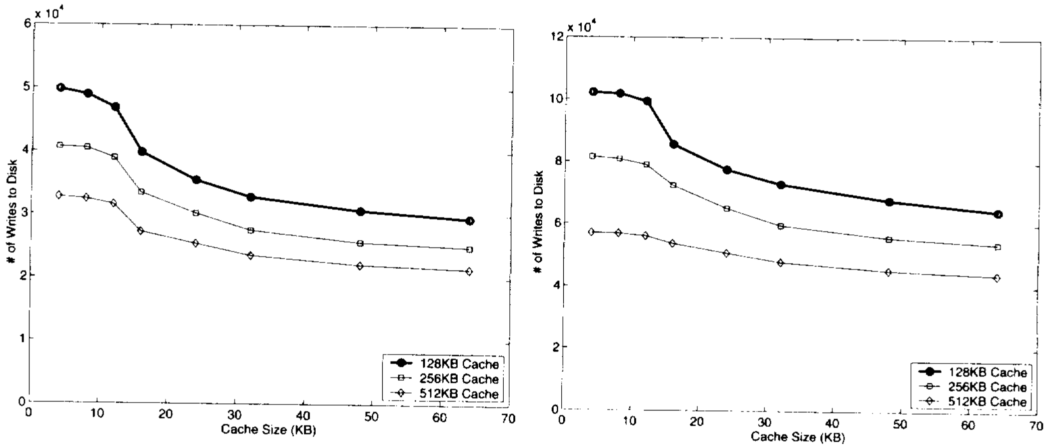
5. When 90 percent of the cache is full of dirty pages, expel the segment S_v such that S_v has $\max\{n_i \mid i > k\}$ until $S_v = S_k$.

3 Results

We investigated the effects of new cache replacement policy on the utilization of the disk with a specific emphasis on the overall activity of the cache and the disk. We measured the number of times that writes were made to the disk to clean the cache and the size of each write used to clean the cache. We also recorded the number of cache hits and cache misses for the non-volatile write cache.

To run our experiments, we implemented our own model of the HP97560 disk drive. We modeled the components of the I/O subsystem of interest: the disk head, the disk controller, the data bus, and the read and write caches. We based our models on techniques described by Ruemmler and Wilkes [7] and an implementation by Kotz, Toh, and Radhakrishnan [6]. We then used the Snake 5 and Snake 6 file system traces collected by Ruemmler and Wilkes [8] to drive a detailed simulation of the disk subsystem.

To validate our approach of grouping segments into hot and cold regions, we looked at the effect of manually varying the size of the hot region of our cache. If this approach is correct, the number of writes to disk should be high when the hot region is small because the large hot segments are frequently being replaced. As cache size increases and approaches the knee, the number of writes to disk should decrease rapidly because more hot segments fit into the hot region. The number of writes should then decrease gradually past the knee as all the hot segments are now in the hot region. The results (see Figure 2) showed precisely this type of behavior, with decreases of as much as 25 percent in the number of writes before the knee and as little as 4 percent after it.



(a) Snake disk 5

(b) Snake disk 6

Figure 2: The effects of varying the size of the hot region of the cache for three different cache sizes with snake disks 5 and 6. The dotted line indicates the location of the knee in the hit ratio curve.

We compared the performance of our replacement policy to those that use temporal locality or spatial locality but not both. We implemented two other policies for points of comparison: the *least recently used* (LRU) replacement policy and the *largest segment per track* (LST) replacement policy. The LRU policy uses temporal locality to replace segments and always purges the most stale data first. The LST policy replaces the most cost effective segments based on segment size first.

We expected the LRU and LST policies to perform worse than our policy overall. The LRU policy handles hot segments well, but makes costly small writes to disk. The LST policy makes efficient writes of large segments, but this is only useful when the segments are cold. Since our policy attempts to deal with both hot and cold segments, we expect that it performs comparably (at least) to the best metrics for LRU and LST. A comparison of the results showed this was true for the number of writes made to disk and the number of stalled writes (see Table 1). This comparison also showed that our new policy consistently overwrote data in the cache more often than the LRU or LST policies.

4 Related work

Systems using non-volatile caches have been discussed in several contexts. The Autoraid system developed by Hewlett-Packard used an NVRAM cache with a RAID disk array to

Table 1: A comparison of three metrics for snake disks 5 and 6 for a 128KB cache.

	Snake disk 5			Snake disk 6		
	LRU	LST	New	LRU	LST	New
writes to disk	33538	33895	32758	57059	54057	59592
stalled writes	418	85	97	398	0	0
cache overwrites	79678	76061	79980	143191	141814	145871

produce a high performance, high reliability storage device [9]. There has been considerable interest in non-volatile caches for use in memory based file systems [3] and in monolithic and distributed disk file systems [1], [2]. The advantages of the use of non-volatile caches with on-line transaction processing (OLTP) systems has been investigated [4].

Despite a large interest in non-volatile memory, little comparative work has been done with cache management policies in file system applications [2], [8]. Thresholds were found to significantly improve the performance of non-volatile caches which only take advantage of spatial locality [2]. Work by the authors with such policies showed that temporal locality and large write size can both be used to strongly improve overall write performance [5].

5 Conclusions

Non-volatile disk caches can reduce disk workload more effectively than conventional caches because they allow disk writes to be safely delayed. This approach has two benefits. First, some writes will be avoided because the blocks will be overwritten or deleted while they are still in the cache. Second, the remaining disk writes can be organized more efficiently by allowing contiguous blocks to be written in a single I/O operation.

We have presented a block replacement policy that organizes efficiently disk writes while keeping the blocks that are the most likely to be accessed again in the cache. Our policy partitions the blocks in the cache into two groups: the hot blocks that have been recently referenced and the remaining blocks that are said to be cold. Hot pages are guaranteed to stay in memory. Whenever space must be made in the cache, the policy expels the cold blocks that belong to the largest set of contiguous segments within a single track until enough free space has been made.

Experimental results showed that by using our replacement policy with a correctly tuned, modest sized cache, it is possible to reduce writes to disk by 75 percent on average and the policy frequently did better. The results showed the new policy to be more effective than cache replacement policies which exploited either spatial or temporal locality, but not both. In particular, data was overwritten in the cache more often using our policy than the others, saving writes to disk. The new replacement policy also gave the relative benefits of such policies without their unattractive features. It often provided the least number of writes to disk of any of the policies we used for comparison. At the same time, it often produced no stalled writes when other policies produced hundreds.

Acknowledgements

We are very grateful to John Wilkes and the Hewlett-Packard Company for making their file system traces and the libraries to read them available to us.

References

[1] Mary Baker, Satoshi Asami, Etienne Deprit, John Ousterhout, and Margo Seltzer. Non-volatile memory for fast, reliable file systems. *Operating Systems Review*, 26(Special issue):10–22, Oct 1992.

[2] Prabuddha Biswas, K. K. Ramakrishnan, and Don Towsley. Trace driven analysis of write caching policies for disks. *Performance Evaluation Review*, 21(1):12–23, Jun 1993.

[3] Peter M. Chen, Wee Teck Ng, Subhachandra Chandra, Christopher Aycock, Gurushankar Rajamani, and David Lowell. The Rio file cache: surviving operating system crashes. *SIGPLAN Notices*, 31(9):74–83, Sep 1996.

[4] G. Copeland, T. Keller, R. Krishnamurthy, and M. Smith. The case for safe RAM. In *Proceedings of the 15th International Conference on Very Large Databases*, pages 327–35. Morgan Kaufmann, Aug 1989.

[5] Theodore R. Haining and Darrell D. E. Long. Management policies for non-volatile write caches. In *1999 IEEE International Performance, Computing and Communications Conference*, pages 321–8. IEEE, Feb 1999.

[6] David Kotz, Song Bac Toh, and Sriram Radhakrishnan. A detailed simulation model of the HP 97560 disk drive. Technical Report PCS–TR94–220, Department of Computer Science, Dartmouth College, 1994.

[7] Chris Ruemmler and John Wilkes. An introduction to disk drive modeling. *IEEE Computer*, 27(3):17–28, Mar 1994.

[8] Chris Ruemmler and John Wilkes. Unix disk access patterns. In *USENIX Technical Conference Proceedings*, pages 405–20. USENIX, Winter 1993.

[9] John Wilkes, Richard Golding, Carl Staelin, and Tim Sullivan. The HP AutoRAID hierarchical storage system. *Operating Systems Review*, 29(5):96–108, Dec 1995.

Switched FC-AL: An Arbitrated Loop Attachment for Fibre Channel Switches

Vishal Sinha
sinha@cs.umn.edu
Department of Computer Science and Engineering
University of Minnesota Minneapolis, MN 55455
7481 Lee Hwy, Falls Church, VA 22042
Tel: +1-703-698-9749
Fax: +1-703-245-4545

David H. C. Du
du@cs.umn.edu
Distributed Multimedia Research Center
Computer Science and Engineering Department
University of Minnesota, Minneapolis, MN 55455
Tel: +1-612-626-7523

Introduction

Fibre Channel Arbitrated Loop (FC-AL) is a loop architecture. It can support any combination of hosts and storage devices, up to a maximum of 127 nodes. With 100 Mbytes/sec bandwidth, the loop structure enables rapid exchange of data between the devices.

A major problem with the existing FC-AL is scalability. Theoretically, a maximum of 127 devices can be attached to a loop. However, past research has indicated that the loop can be saturated by as few as 32 devices. This number is expected to decrease rapidly, as the disks become faster and faster. To overcome this limitation, we investigated the possibility of extending the existing FC-AL protocol.

Switched FC-AL protocol aims at using a switch to accommodate Arbitrated Loop devices without requiring a true fabric connection. Ideally, both fabric and loop devices should be able to share a single switch and enjoy their own 100Mbytes/s segments on each switch port, and yet be logically grouped together for high-speed transactions.

We designed and simulated two different approaches to implement the switched FC-AL protocol. In the first approach, which is a circuit switched approach, the initiator establishes a connection with the target on the remote loop before initiating the data transfer. This approach is useful for applications that have hard Quality of Service (QOS) requirements. In the second approach, which is a packet switched approach, the exchange of data takes place in hops. The initiator needs to establish a local connection only. Each of these approaches has it's own pros and cons. The objective of our research is to investigate the design issues and trade-offs of both these approaches.

Switched FC-AL design:

The two design considerations are the circuit switched approach, in which the Initiator wins arbitration on it's local loop and establishes a connection with the Target on the remote loop via the switch. The second approach is based on the packet switched concept, in which the Initiator wins arbitration on its local loop and transmits data to the Switch Connected Node (SCN) attached to the switch. The SCN buffers the data and inturn arbitrates on the remote loop to deliver the data to the Target. A detailed description of the two approaches are presented below.

Circuit switched approach:

Figure 1 illustrates this approach. The Initiator X, sends out an arbitrate primitive ARB(X) to gain access to the loop. Upon winning the arbitration, by receiving the ARB(X) primitive back, it sends out the OPEN(Y) primitive. If the target Y is not on the local loop, the Switch Connected Node, SCN(L_1) intercepts the OPN(Y) primitive and sends a request to the switch to access the remote loop. If the remote loop is not busy, the switch sends an acknowledgement back to the SCN(L_1) and a connection request is sent to SCN(L_n) via the switch. If the remote loop is busy, the switch sends a negative acknowledgement back to the SCN(L_1) and a connection tear down request is sent to Initiator X by SCN(L_n). Else, the SCN(L_n) becomes the initiator of this remote request on the local loop Ln. It next proceeds to establish a connection with the Target Y. If successful, the Target Y responds with R_RDY primitive which gets routed all the way back to the Initiator X . At this stage the connection is established and all the data transfer takes place directly between the Initiator and the Target using the usual FC-AL protocol. Tear down of the connection can be initiated by either the Initiator or the Target by sending the CLS primitive.

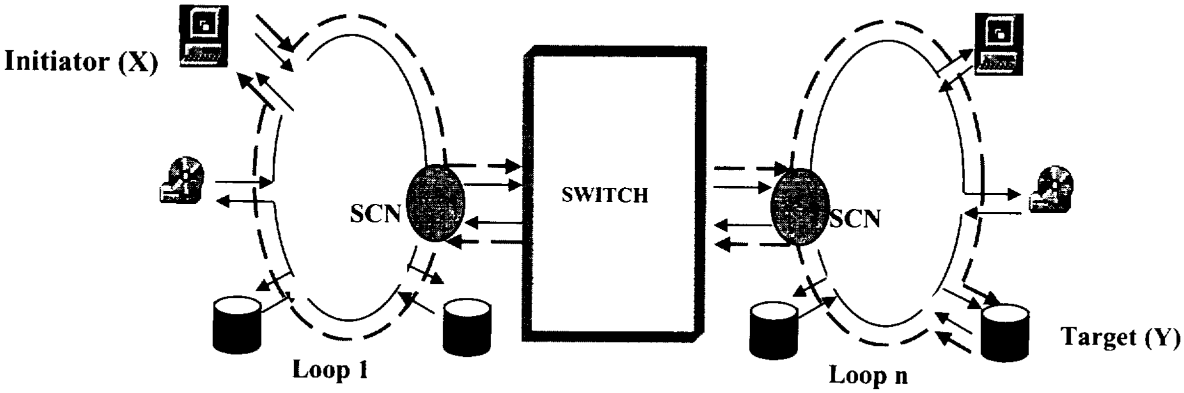


Figure 1: The simulation model for the circuit Switching based approach.

Packet switched approach :

It follows the store-and-forward technique of data transfer. In this approach, the Initiator X, sends out the arbitrate primitive ARB(X) to gain access to the local loop. On winning the arbitration, it send out the OPEN(Y) primitive. If the target Y is not on the local loop, the Switch Connected Node, SCN(L_1) intercepts the OPN (Y) primitive and sends a request to the Switch to access the remote loop.

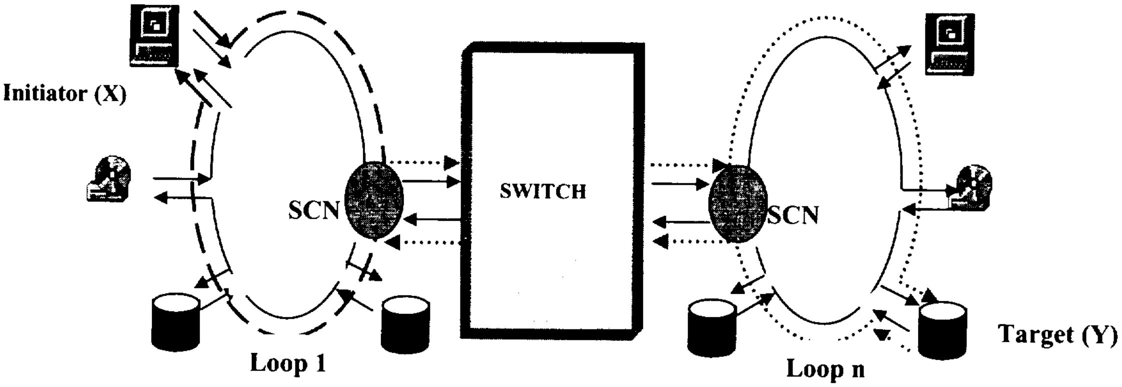


Figure 2: The simulation model of the packet switching based approach

At the same time, it sends a local acknowledgement back to the Initiator in terms of R_RDY. At this stage, SCN (L_1) becomes the local target for the global request in loop L1 and all the data transfer takes place directly between the Initiator X and the SCN(L_1) using the FC-AL protocol. If the remote loop is not busy, the switch sends an acknowledgement back to the SCN (L_1) and a connection request is sent to SCN(L_n) via the switch. The SCN (L_n) next proceeds to establish a connection with the Target Y. If successful, the Target Y responds with R_RDY primitive which gets routed back to SCN

(L_1). Now data transfer takes place directly between the SCN (L_1) and Target Y. The advantage of this approach lies in the fact that both the stages of data transfer can take place asynchronously there by providing some means of parallelism. Also, the acknowledgement sent is segment-by-segment, therefore a negative acknowledgement from the switch does not need to tear down the connection.

Simulation Model

The detailed simulation model is shown in Figure 3. Each loop consists of one host, one Switch Connected Node (SCN)[1] and a variable number of disks. The number of disks attached to a loop is one of the parameters of our study. SCN has been given the highest priority and does not implement the fairness algorithm. All other nodes, including the host are assigned priority randomly. Table 1 shows the values of the parameters used for the simulation.

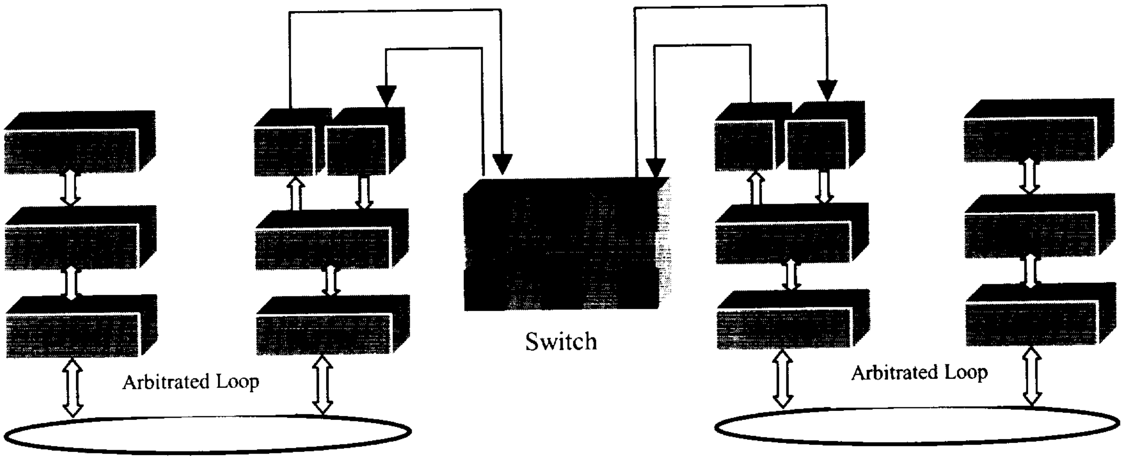


Figure 3. The detailed simulation model used for both the schemes.

Parameters	Default Values	Description
Propagation delay	3.5ns	The propagation delay between two nodes
Per node delay	6 words time	The delay for the interface to forward the frame.
Bandwidth	100 Mbytes/sec	The FC-AL link bandwidth
Maximum outstanding commands	8 times the number of disks on the loop.	The maximum number of commands allowed at the initiator. Used for the stress test.

Table 1: Switched FC-AL simulation model parameters

Simulation Results

We studied the performance of switched FC-AL under two different load scenarios namely, light load and heavy load. In the first scenario, the system is under extremely light load. The host generates a request and waits till it gets serviced, before generating the next request. Thus, the storage subsystem has only one outstanding command at a time. This allowed us to compare the total latency of both the approaches as the percentage of global traffic is varied. It also gave us an idea of the latency

[1] SCN is the interface between the loop and the switch. It's like a normal node with extra buffers.

overhead involved. We also studied the effect of the number of loops on the total latency of the system.

In the second scenario (heavy load), we studied the scalability of the existing FC-AL protocol and compared it with that of switched FC_AL. We heavily loaded the loop by setting the number of outstanding commands per disk to 8, and increased the number of disks per loop. The purpose of this test was to discover the number of disks (all the three types) needed to saturate the loop and to find out how the two approaches of switched FC_AL perform near saturated conditions.

Performance under Light Load

In the light load scenario, the target disk and loop for any command is chosen randomly according to a uniformly distributed random number. We use 64KB as the request size to represent a normal request. We want to see how the loop latency and total latency change as the percentage of local and global traffic changes and also the effect of number of loops on the total and loop latency of the storage subsystem.

Percentage Global Traffic	Maximum circuit_1 establish time	Maximum circuit_2 establish time
0	0.6875	0.697754
10	1.584229	1.805176
20	2.085144	2.28125
30	2.462872	2.783691
40	2.754395	2.988281
50	3.386963	3.453369

Table 5: Maximum circuit establish time for both the phases of circuit switched approach.

Table 5 shows the effect of increasing global traffic on the circuit establish time for circuit switched approach when the number of disks on each loop is 8 and the request size is 64KB. Evidently, the circuit establish time[2] increases exponentially with the percentage of global traffic. To understand this behavior, we counted the number of attempts needed by an initiator to establish a circuit. Graph 2 shows the observed behavior.

	Circuit switched approach				Packet switched approach			
%global traffic	Disk time	Loop time	Total time	# of re attempts	Disk time	Loop time	Total time	# of attempts
0	4.86336	0.822962	5.686322	0	4.86336	0.822964	5.686324	0
10	4.969455	0.851242	5.820697	1.964646	4.917598	0.89883	5.816428	1.006711
20	5.078583	0.880659	5.959242	4.49596	5.010529	0.993514	6.004043	1.013423
30	5.091376	0.926795	6.018171	10.15657	5.053116	1.123394	6.17651	1.020134
40	5.112677	0.985283	6.09796	18.69596	5.067917	1.251087	6.319004	1.026846
50	5.149949	1.050099	6.200048	28.51717	5.098782	1.379847	6.478629	1.033557

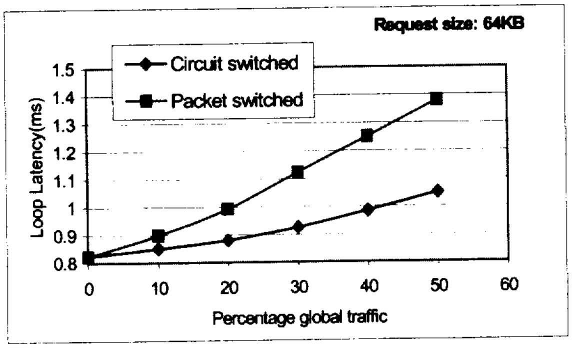
Table 6: Average latencies of both approaches with varying percentage of global traffic.

latency. The disk access time consists of the command queuing time, disk seek time, disk rotation latency and the data transfer time. The loop latency consists of the time to win

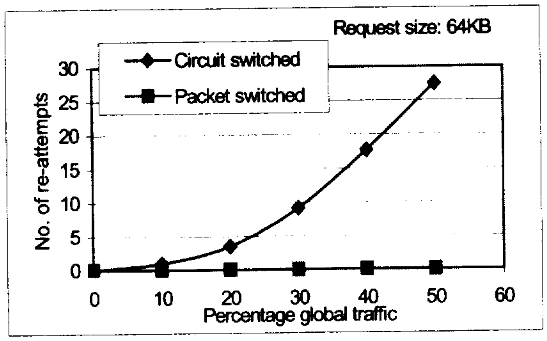
[2] Circuit establish time is defined as the total time an initiator takes to win arbitration, open the target and get the R_RDY back.

the arbitration and the transmission time of control and data frames. In order to avoid concurrent data fetch and transfer feature of FC-AL, we restricted our request size to 64KB.

From Table 6, we found that the disk access time dominates the total time for any percentage of global traffic. This is because the bandwidth of FC-AL loop is 100 MB/s, and it takes just 0.64 ms to transfer the 64KB data, whereas the average seek time alone of the disk is 2.475 ms. This clearly indicates the need for dividing a transaction into two phases. In the first phase, a host can send the request to the disk and then relinquish control over the loop. In the second phase, the disk, after fetching the data, can transfer the data back to the host, thereby allowing multiple I/O operations to fully utilize available bandwidth of FC-AL at the same time.



Graph 1: Comparison of loop latency of both the approaches.



Graph 2: Comparison of "number of re-attempts needed " for both the approaches.

We also observe that both disk access time and loop latency increase as the percentage of global traffic increases. The increase in disk access time is primarily due to the increase in command queuing time. As the percentage of global traffic increases, the probability of two requests being generated for a particular disk increases and this contributes to the increased queuing time.

Graph 1 shows the loop latency of both the approaches. The rapid increase in loop latency, in the case of circuit switched approach, is due to the exponential rise in number of attempts needed to establish the circuit. In the packet switched approach, the increase though exponential, is very small, and is mainly contributed by the additional store and forwarding time of the frames at the SCN. In the light load scenario, the circuit switched approach exhibits better total request servicing time than the packet switched approach.

We also investigated the effect of increase in the number of loops on the total request service time. The percentage of global traffic was set to 40% and request size to 64KB. From table 7, we observe that as the number of loops increases, the number of re-attempts needed to establish a connection increases more rapidly for the circuit switched approach than the packet switched approach. So the increase in total latency is more for the former than in the latter.

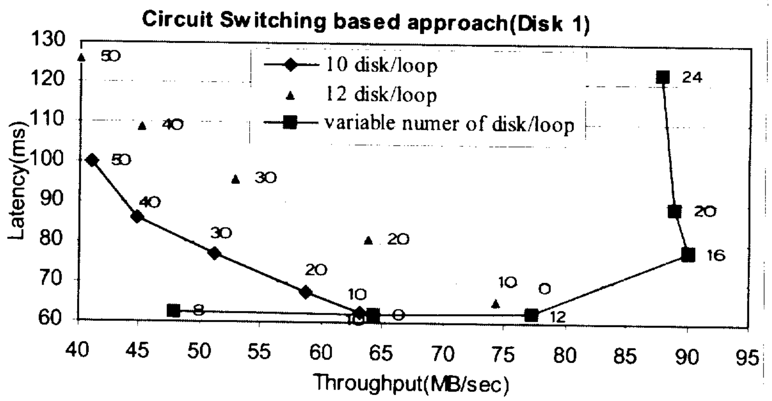
	Circuit switched approach				Packet switched approach			
No. of loops	Disk time	Loop time	Total time	# of attempts	Disk time	Loop time	Total time	# of attempts
2	5.029166	0.980776	6.019942	45	5.00825	1.210549	6.298799	26
4	5.122677	0.985283	6.107959	17519	5.067917	1.241087	6.319005	29
8	5.12838	1.014671	6.143051	42740	5.107891	1.272409	6.372301	49

Table 7: Average latencies of both approaches with number of loops.

Performance under Heavy Load

With the advancement in disk technology, fewer disks are now capable of saturating an arbitrated loop. To investigate this scenario, we studied the performance of FC-AL with three different types of disks. At the beginning of the simulation, the commands were generated one after the other until the number of outstanding commands reached a pre-defined upper limit. This value was set to be the number of disk on the loop, multiplied by 8. We maintained the same number of outstanding commands throughout the entire simulation. When one command is completed, a new command is generated right away. We used two uniformly distributed random number sequences, which are generated with different seeds. They are used to determine the target disk and loop for any command.

The simulation results of circuit switched and packet switched approaches for disk1 are shown in graph 1 and 2 respectively. The graphs depict three series. The first one, from the right, shows the number of disks needed to saturate the loop with a given 64KB request size while the next two series show the throughput vs. latency behavior of the two approaches before saturation. Each of the data points in a graph is labeled with a number indicating either the number of disks per loop (for first one) or the percentage of global traffic (for series 2 and 3). The percentage of global traffic is varied from 0 to 50, in steps of 10.

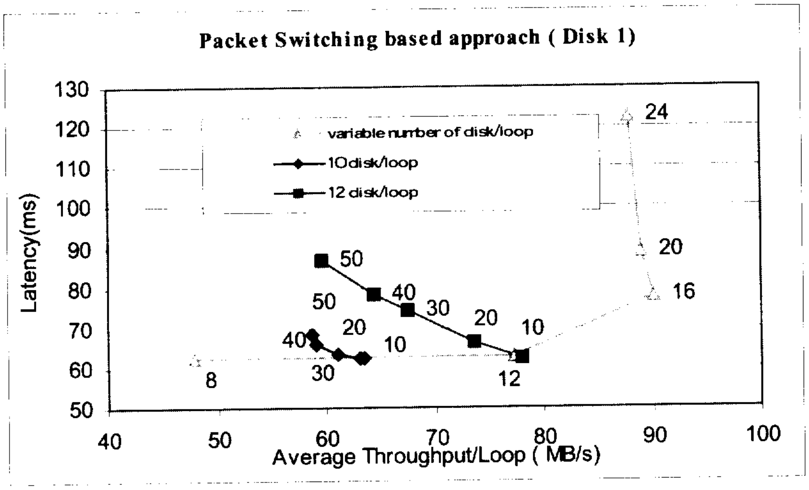


Graph 3: Throughput vs. Latency behavior of the circuit switching based approach.

From graph 3, we can see that the FC-AL loop starts getting saturated with 16 disks of type 1. The maximum throughput obtained is 90.04 Mbytes/sec. After saturation, with more disks attached to the loop, longer latency is observed.

We investigated the performances of both the circuit switched and the packet switched approaches, when the number of disks on the loop is 12 and 10. From graph 3 and 4, we observe that as the percentage of global traffic increases, the throughout decreases while the latency increases. This is mostly due to the following two factors: increased arbitration overhead and blocking. Since we need to win arbitration on both the loops for each global request, the arbitration time is doubled. The second factor contributing to this delay is the number of blocked requests. In case of circuit switched approach, a request is blocked if the remote loop is busy servicing another global request while in case of packet switched approach, a request is blocked for the duration in which data is transferred between the host and the target SCN. Since the blocking time is much longer for the circuit switched approach, it incurs steep rise in latency as the percentage of global request increases. As a result, the packet switched approach shows better throughput and latency characteristics than circuit switched approach.

We also observe that the latency of the packet switched approach for 10 disk/loop is 68.52 ms while that of 12 disk/loop is 86.52ms. This big difference is due to the fact that as the number of disk on a loop increases, not only does the propagation delay and per node delay increase but so does the number potential initiators on the loop[3]. Hence, the number of L-Ports arbitrating at a time increases forcing the arbitration latency to increase.



Graph 4: Throughput vs. Latency behavior of the packet switched approach.

Conclusion and Future Work

In this paper, we presented some initial results of our study on the performance of the circuit and packet switched approaches to implement switched FC-AL. The light load test indicated that even with vast improvement in the disk technology, the disk is still a bottleneck and we need to multiplex several I/O operations to fully utilize the available

[3] In our implementation, a disk is an initiator in the second phase of data transfer.

bandwidth. We also see that with an increase in the percentage of global traffic the number of attempts needed to establish a circuit increases rapidly. This is the main limitation of the circuit switched approach. Packet switched approach, on the other hand needs very few attempts to send the request to the remote loop but it's main overhead is the store and forwarding time. Also, with an increase in the number of loops connected to the switch, there is a marginal increase in the overall latency for both the approaches.

The scalability issue of the switched FC-AL protocol was studied using the heavy load test. It is evident that with faster disk, the number of disks needed to saturate the loop is very low. Hence, to use large numbers of disks and still have better throughput vs. latency behavior, the switched topology is a good option. For both the approaches, the aggregate throughput for as much as 50% of global traffic was quite high but clearly the packet switched approach outperformed the circuit switched approach under our set of assumptions.

We next plan to investigate the effect of variable number of hosts per loop on the performance of both approaches. Other related issues like larger request size, inverse priority of SCN and higher bandwidth of the loop also need to be studied. Finally, a comparison of switched FC-AL and FC-AL 3 will be worth doing.

References:

[1] ANSI X3.272-199x, "Fibre Channel - Arbitrated Loop (FC-AL), Revision 4.5", American National Standard Institute, Inc., June 1, 1996.

[2] ANSI X3.272-199x, "Fibre Channel - Arbitrate d Loop (F C-AL-2), Revision 5.1" , American National Standard Institute, Inc., March 26, 1996.

[3] ANSI X3.230-1994, Fibre Channel Physic al and Signaling Interfac e (F C-PH)" American National Standard Institute, Inc., 1994.

[4] David H.C. Du, Jenwei Hsieh, Taisheng Chang, Yuewei Wang and Sangyup Shim, Performance Study of Emerging Serial Storage Interfaces: Serial Storage Architecture (SSA) and Fibre Channel - Arbitrated Loop (F C-AL)", Submitted .

[5] Chris Ruemmler and John Wilkes , "An introduction to disk drive modeling" IEEE computer, march 1994, pp. 17-28.

[6] Gadzoox Network's white paper on the Denali Fibre Channel Switch. http://www.gadzoox.com/links/products/switch/denali.html

[7] Vixel's white paper "Arbitrated Loop Attachment for Fibre Channel Switches" http://www.vixel.com/whitepapers/wpaper2.html, July 1998

Evaluating Backup Algorithms *

Zachary Kurmas
College of Computing, Georgia Tech
801 Atlantic Ave. NW
Atlanta, GA 30332-0218
tel 1-404-894-9390
fax 1-404-385-1253
kurmasz@cc.gatech.edu

Ann L. Chervenak
USC Information Sciences Institute
4676 Admiralty Way, Suite 1001
Marina del Rey, CA 90292-6601
tel +1-310-822-1511
fax +1-310-823-6714
annc@isi.edu

Abstract

We present a trace-driven simulator that evaluates the performance of backup algorithms. We use this simulator to compare the performance of the commonly-used *level* scheme (our name for the algorithm used by the UNIX dump utility) with that of a new algorithm called the *Z scheme*. We show that the Z scheme has better backup performance than the level scheme and slightly worse restore performance. We also show that the Z scheme consumes less media for backups than the level scheme.

1 Introduction

Given the unprecedented increases in magnetic disk drive capacities, university, enterprise, and government data repositories are growing rapidly. Protecting these data repositories using traditional backup techniques is a growing challenge [3].

In this paper, we evaluate four backup strategies. Our eventual research goal is to demonstrate the limits of current backup algorithms and propose new algorithms both for sequential magnetic tape media and for random access alternatives, including write-once technologies such as CD-ROM and DVD. Initially, we focus on backups to sequential magnetic tape media.

Every backup algorithm makes a tradeoff between backup performance and the performance of *restore* operations, in which files are retrieved from the backup medium. At one extreme, an algorithm optimized for fast backup might simply write periodic *incremental backups*, or copies of files that have changed since the last backup. Each day, this *pure incremental* approach copies to the backup medium only the files that have been modified that day. The disadvantage of this simple, fast backup scheme is that the performance of restore operations may suffer. If files in a single directory are modified on different days, the pure incremental algorithm may disperse those files widely on the sequential backup media. Later, to restore the entire directory, reading large amounts of sequential media may be necessary.

*This work was supported in part by NSF CAREER Award CCR-9702609.

At the other extreme, an algorithm optimized for fast restore would take pains to lay out data carefully on the backup medium, copying all files in a given directory to the same backup medium and grouping together directories that are logically connected. An algorithm of this type might even perform daily *full backups*, in which all files and directories are written to the backup medium. The goal of these algorithms is to reduce the time required to restore a given directory or file system by minimizing the quantity of sequential media that must be accessed during the restore operation. The disadvantage of such algorithms is that the backup phase requires substantially more media and takes much longer, given the bandwidth limitations to sequential tape media (e.g. the speed at which the tape drive can write).

Between these two extremes are many other backup algorithms that attempt to strike a balance between backup performance, restore performance, and media requirements. We evaluate two of these algorithms here: the traditional backup scheme used by many programs, including UNIX dump, which we call the *level* scheme, and a new algorithm we developed which we call the *Z scheme* [6, 1].

To evaluate these backup algorithms, we wrote a trace-driven backup simulator. Inputs to this simulator include traces made by Tim Gibson [4] as well as traces of backup activity collected in the College of Computing at Georgia Tech. Our simulation results evaluate the performance of the full backup, pure incremental, level, and Z schemes. As expected, our results show that the full backup algorithm has poor backup performance but performs restores efficiently. In contrast, the pure incremental scheme performs well for backups but poorly for restore operations. The level scheme performs well for all metrics, with backup performance close to that of the pure incremental scheme and restore performance close to that of the full backup scheme. The Z scheme performs backup operations better than the level scheme, and restore operations almost as well as the level scheme. We will argue that the Z scheme's better backup performance makes up for its slightly worse restore performance, and, therefore, is a better algorithm.

2 The Backup Simulator and File System Traces

The trace-driven backup simulator is written in C++. Its modular, object-oriented design simplifies implementing new backup algorithms, accommodating new trace formats, and gathering different performance metrics.

Input traces for the backup simulator contain periodic records of the state of a file system hierarchy — specifically, daily images of the file system metadata at the time backups are performed. Using these records or snapshots of the state of the file system, the simulator traverses the file system hierarchy, examines file metadata, and decides which files should be backed up according to the specified algorithm. The algorithm uses a tape object to keep track of the layout of files on the tape. During a restore operation, the simulator uses Bruce Hillyer's model of a DLT 4000 tape drive to estimate the seek time to access a particular file [5]. Metrics for evaluating the algorithms include the time to perform backup and restore operations and the amount of tape media required for backup.

In this paper, we present simulation results that use Gibson's traces as input [4]. Gibson's traces include a set of files for each day observed. Each trace file contains the name, size, owner, group, inode, access time, modification time, and change time of each file on the traced file system on a given day.

3 Backup Algorithms

We now describe the backup algorithms that we evaluate in this paper.

- **Pure Incremental Scheme:** Once every day (or, more generally, once every period or epoch specified by an algorithm), this scheme copies only those files that have changed since the last incremental backup to the backup medium. We use this scheme as a basis for evaluating optimal backup times.

- **Daily Full Backup Scheme:** Every day, this scheme copies all files in the file system to the backup medium. In general, this scheme puts files in the same directory close together on the backup medium. We use this scheme as a basis for evaluating optimal restore times.

- **Level Scheme:** The level scheme is our name for the traditional backup algorithm that alternates between periodic full backups and incremental backups of different "levels", where a backup at a given level includes all files that have changed since the last backup at that level.

- **Z Scheme:** The Z scheme is the name we give to our algorithm. It is a simplification of an algorithm developed at UC Berkeley by Costello, Umans, and Wu [2]. This scheme performs concurrent backups to several *backup streams*. The Z scheme uses a parameter b, called the *base*. A particular file is written to backup stream i if it was last modified *exactly* b^i days ago. For example, every day the algorithm writes to stream 0 those files that were last modified yesterday. If the base b is 2, the algorithm writes to stream 3 those files last modified exactly $2^3 = 8$ days ago.

To restore every file currently in a file system, a backup algorithm must read the most recent version of every file from tape. In a pure incremental backup scheme, we found that restoring a file system required reading a large number of tapes. In particular, we found that for an incremental backup scheme that uses one tape per day, there is a high probability that each backup tape contains at least one file that was last modified on that date (usually, there are several files). To restore a file system on day x, the restore operation must load and read at least one file from approximately x tapes. Since the time to load and seek on a tape often exceeds the time to read files, this results in poor restore performance for the pure incremental scheme.

The level scheme trades backup performance for improved restore performance by redundantly backing up files in such a way that the number of tapes that must be read to restore a file system is bounded by a constant (provided that the size of the file system is also bounded).

As we will see, the level scheme has very good performance; however, its performance is hindered by two inefficiencies, both of which are addressed by the Z scheme: First, the daily bandwidth needed by the level scheme varies greatly. The amount of bandwidth needed for a level 0 (or full) backup is about 25 times the amount needed by a daily level 9 (or incremental backup). Therefore, a user of the level scheme must maintain enough backup equipment to handle a level 0 backup, even though that equipment will remain idle for most of the year. Second, the lower level backups (e.g. levels 0, 5, and 7), back up all files modified since the last backup of a lower level regardless of whether those files are likely to be modified in the near future. For example, consider a file F that is modified every day, and, therefore, sent to tape by a level 7 backup every Sunday. Because file will be modified again Monday, any restore will have to pass over file F's spot on the tape used for the level 7 backup. This takes time that could be saved if the file was simply not backed up by level 7. The Z scheme addresses this problem by not writing files to higher level streams until they have not been modified for several days.[1]

4 Evaluation of Algorithms

In this section, we present comparisons of backup algorithm performance for the three metrics of interest: the amount of sequential access tape media required to perform backups and the time required to perform backup and restore operations. We evaluate the performance of four algorithms: the pure incremental, full backup, level, and Z schemes. For this paper, the Z scheme was run with a base of 8.

The results below reflect backup and restore performance for a single file system used by graduate students at the University of Maryland at Baltimore County [4]. This file system contains about 1.5 gigabytes at the beginning of the trace and approximately 4 gigabytes at the end of the trace.

The restore results use Hillyer's model [5] to determine seek time. The current simulation results assume a tape drive read rate of 1.5MB/s and a load time of 30 seconds per tape.

4.1 Media Requirements for Backup

First, we compare the cumulative total of bytes written to the backup media for each algorithm. This metric is used for comparison purposes. In practice, some algorithms might discard or re-use tapes that are no longer needed for restore. For example, both the level and the full backup schemes could discard older tapes, assuming they retain at least one subsequent full backup of the file system. However, the pure incremental scheme must retain all tapes that contain active files.

Figure 1 shows the number of bytes written by the four backup algorithms. As expected, the full backup scheme uses, by far, the most media while the pure incremental scheme uses the least. The media requirements of the level scheme are quite low, because it performs frequent incremental backups and only occasional full backups. By the end of the 8-month period in the graph, the level scheme uses approximately twice as much media as does the pure incremental scheme — about 38GB compared to about 18GB The Z scheme's performance is even closer to the pure incremental scheme, using only 25GB

[1] The intuition into why this is effective comes from Gibson, who showed that files modified today will, with high probability, either be modified within a few days, or never modified again. Similarly, files that have not been modified in several days, with high probability, not be modified again [4].

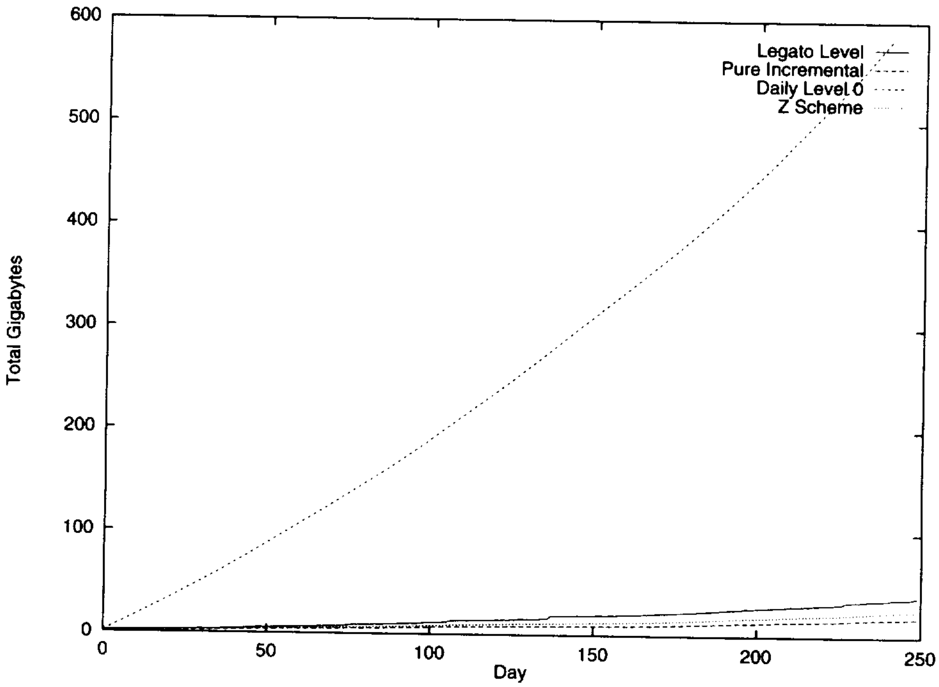


Figure 1: Total Amount of Media Consumed by Day x

4.2 Backup Performance

Figure 4.2 shows how much media each algorithm uses daily. This statistic is directly proportional to the time required to perform daily backups, and hence, to backup performance. The figure shows that the pure incremental scheme, which is optimized for fast backup, minimizes the number of bytes that need to be transferred on a given day. By contrast, the full backup scheme, which is optimized for fast restore performance, requires much more time to perform daily backups. On most days, the performance of the level scheme is similar to that of the pure incremental scheme. However, one day each week, the scheme performs a weekly "level 7" backup that requires about five times as much media as the pure incremental scheme. On one day each year, the level scheme performs a full backup that requires 25 times as much media as the pure incremental scheme does on that day.

Figure 4.2 also demonstrates the bandwidth consistency of the Z scheme (relative to the level scheme). Although on most days the Z scheme uses more media than the level scheme, it does not have the weekly and monthly peaks that the level scheme has. Therefore, as we saw in figure 1, the amount of hardware/bandwidth needed by the Z scheme is less than that of the level scheme.

One rough method of quantifying this difference in consistency is to compute the standard deviation of the amount of backup media used daily. A backup algorithm that writes the same amount of data to tape every day would have a standard deviation of zero. The level scheme has a standard deviation of 75,243, while the Z scheme has a standard deviation of about 12,284. The pure incremental scheme as a standard deviation of 11,226.

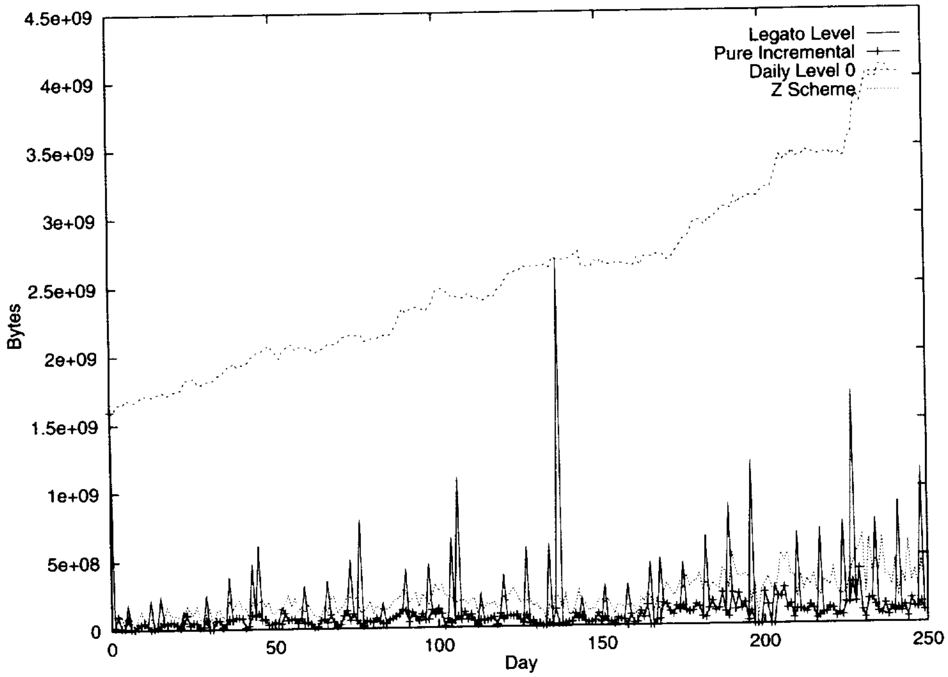


Figure 2: Amount of Media Consumed on Day x

4.3 Restore Performance

Finally, figure 3 shows how long a restore of the entire file system on day x will take. As expected, the full backup scheme, which was optimized for fast restores, has the best restore performance. The pure incremental scheme has the worst performance, because a large number of tape media must be accessed to restore all files in the file system. The level scheme performs very well on restores, within a factor of two of the full backup scheme. The restore performance of the Z scheme is almost as good as that of the level scheme. In fact, the graph of Z scheme restore time seems to connect the peaks in the graph of the level scheme restore times. This makes sense because the level scheme will have the best restore performance after a backup of low level;[2] however, because the Z scheme backs up a consistent amount of data every day, its restore times are more consistent.

Because restores occur rarely, we believe that a daily savings in backup time and media will compensate for a possible, but unlikely, doubling of restore time. Thus, we argue that the Z scheme's improvement in backup performance (especially the reduction of the peaks in bandwidth requirements) outweighs the decrease in restore performance.

5 Future Work

Our current project is to develop a distributed version of the simulator. Currently, each simulation can only simulate one file system. Although we can run several simulations in

[2] Notice that in figure 3, the restore performance dips to that of the Daily level 0 on day 135, when a level 0 backup is performed.

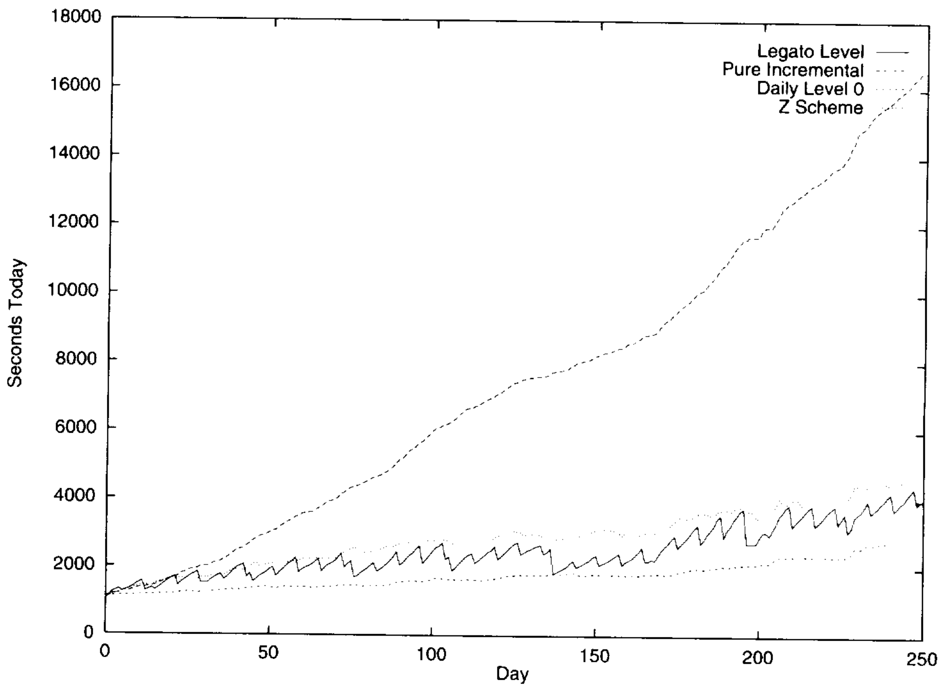


Figure 3: Time of Restore on Day x

parallel, simply summing the output of each simulation does not reasonably approximate the restore performance of a large file system (especially on a serpentine tape, such as DLT). Our distributed simulator will allow a Tape Drive object running in one process to accept read and write requests from File System objects running in several other processes. By carefully coordinating the writes to the Tape Drive object, we can make several File System objects behave as if they were subdirectories in a single, large, file system.

After we finish the distributed version of the simulator, we plan to implement several more backup algorithms. Given our results of running the Z scheme with different values for the base (not presented here due to space constraints), and Gibson's findings on file modification patterns [4], we believe that basing the actions of each stream on a power of some base is too restrictive. Therefore, we will implement the Generalized Z scheme, in which, given an array of integers A, stream i backs up those files that have not been modified in exactly A_i days. We expect that we can choose the values of A carefully so as to improve both backup and restore performance.

We plan eventually to generate results that use the larger backup traces we collected at Georgia Tech's College of Computing; however, because the Georgia Tech traces are generated by a backup algorithm, we are still working on a method to remove the influence of the backup algorithm on the traces.

Finally, our long-term goal is to progress to studying random-access media, such as CD-ROM and DVD. The cost of writers for these media is decreasing to the point where it may soon be more cost-efficient for small companies to use these media instead of magnetic tape.

6 Conclusions

Efficient backup algorithms must strike a balance between backup and restore performance. We have used a trace-driven backup simulator to evaluate four backup algorithms using traces collected by Tim Gibson. At one extreme, we showed that a pure incremental scheme performed backup operations efficiently, but performed restores inefficiently. At the other extreme, an algorithm that performed full backups every day provided efficient restores operations but slow backups.

We showed that a traditional *level* backup scheme that alternates frequent incremental backups at various "levels" with occasional full backups performs well for both backup and restore operations. For one file system trace, we showed that the total number of bytes stored by the level scheme was within a factor of two of the number of bytes stored by the pure incremental scheme. In addition, except for monthly "level 5" and annual "level 0" full backups, the daily backup performance of the level scheme was within a factor of five of the pure incremental performance. Finally, the level scheme performs restore operations efficiently, within a factor of two of the performance of the full backup scheme.

We also showed that our new *Z scheme* performs better than the level scheme for backup operations, but slightly worse for restore operations. However, because backups must take place daily, while restores take place rarely, we believe that the Z scheme's improvements in backup performance, both by using less backup media, and by having a more consistent daily bandwidth requirement, more than make up for its slight decrease in restore performance.

References

[1] A. L. Chervenak, V. Vellanki, Z. Kurmas, and V. Gupta. Protecting File Systems: A Survey of Backup Techniques. In *Proceedings*. Joint NASA and IEEE Mass Storage Conference, March 1998.

[2] A. Costello, C. Umans, and F. Wu. Online backup and restore. Unpublished Class Project at UC Berkeley, May 1998.

[3] J. da Silva and O. Guomundsson. The Amanda Network Backup Manager. In *Proceedings of USENIX Systems Administration (LISA VII) Conference*, pages 171–182, November 1993.

[4] T. Gibson, E. L. Miller, , and D. D. E. Long. Long-term File Activity and Inter-Reference Patterns. In *CMG98 Proceedings*. Computer Measurement Group, December 1998.

[5] B. K. Hillyer and A. Silberschatz. On the Modeling and Performance Characteristics of a Serpentine Tape Drive. In *Proceedings SIGMETRICS*. ACM, May 1996.

[6] DUMP(8). Unix System V man page.

An Overview of Version 0.9.5 Proposed SCSI Device Locks

Andrew Barry, Mike Tilstra, and Matthew O'Keefe
Parallel Computer Systems Laboratory, University of Minnesota
200 Union Street SE, Minneapolis, MN 55455
+1-612-626-7180, {barry, okeefe, tilstra}@borg.umn.edu
Kenneth Preslan - Sistina Software
Gerry Houlder and Jim Coomes - Seagate Technology
James Wayda - Dot Hill Systems Corp.

Abstract

Symmetric shared disks file systems are functional, versatile, and just plain cool. For these file systems to work, they require a global lock space, acessable to all clients. This paper describes a proposal for the implementation of the SCSI device lock command which provides a global lock space on storage devices. This paper details how Dlocks behave. This paper is a condensed overview of the 0.9.5 version of the proposed SCSI device locks specification. The full paper includes a thorough description of Dlock implementation, optional features of Dlocks, and a discussion of how Dlocks are used by the Global File System. For a postscript version of the full 0.9.5 Dlock specification, please visit our web site http://www.globalfilesystem.org . This paper is an evolution of the 0.9.4 Specification of SCSI Device Locks, which can also be found at the GFS web site.

1 Dlock Concepts

1.1 Expiration

In a shared disk environment, a failed client may not be allowed to indefinitely hold whatever locks it held when it failed. Therefore, each holder must continually update a timer on the disc. If this timer expires, other lock holders may begin error recovery functions to eventually free the lock. Expiration is alternately refered to as 'timing-out', and the act of updating the timer is often refered to as 'heartbeating'.

1.2 Client IDs

The Client ID is a unique identifier for each client. The client id is completely opaque to the Dlock device. In GFS the client ID is used both as an identifier and to store the IP address of the client, allowing inter-machine communication. The Client ID can be any arbitrary 32-bit number that uniquely identifies a machine.

1.3 Version Numbers

Associated with every lock is a version number. Whenever the data associated with a lock is changed, the version number is incremented. Clients may use cached data instead of re-reading from the disk as long as the version number on the dlock is unchanged since the data was read. This is reported by the lock device, so that the clients know not to use cached data.

1.4 Conversion Locks

The conversion lock is a simple- single stage queue used to prevent writer starvation. There is an awkward case where one client is trying to acquire an exclusive lock and can't because other clients are constantly acquiring and dropping the lock shared. If there is never a gap where no client is holding the lock shared, the writer starves. To correct this, when a client unsuccessfully tries to acquire a lock, and no other client holds that lock's conversion, the conversion is granted to the unsuccessful client. Once the conversion is acquired, no other clients can acquire the lock. All the current holders eventually unlock, and the conversion holder can get the lock. All of a client's conversions are lost if the client expires.

1.5 Enable

In the event that a lock device is turned off, and comes back on, all the locks on the device could be lost. Though it would be nice if the locks were stored in some form of persistent storage, it is unreasonable to require it. Therefore, lock devices should not accept dlock commands when they are first powered up. The devices should return failure results, with the enabled bit of the dlock rdf cleared, to all dlock actions except refresh_timer, until the dlock action enable is issued to the drive. In this way, clients of the lock device are made aware that the locks on the lock device have been cleared, and can take action to deal with the situation.

2 The SCSI Dlock Command

The SCSI dlock command is a method of data synchronization. The Dlock interface is defined by three main parts: the CDB, the return data format, and the mode page.

2.1 The Dlock CDB

The Dlock command has a 16 byte Command Descriptor Block (CDB). It is shown in Table 1. Its fields are:

Operation Code The SCSI Operation code for Dlock is 83h.

Action This describes the action being requested. The possible values of this field are shown in Table 2.

Lock Number This is the number of the lock on which to operate.

Client ID An application-defined 32-bit number that identifies the client issuing the Dlock command.

Allocation Length The number of bytes that the initiator has allocated for data returned from the command. Note that the Allocation Length field can be used to control how much of the Reply Data is returned to the initiator. If the Allocation Length is too small for the amount of Reply Data that needs to be returned, only Allocation Length bytes of the Reply Data are returned. The Dlock command completes just as it would have if there was more space allocated. This is not an error.

2.2 Dlock Actions

The possible actions that each Dlock command performs are listed in Table 2.

Nop Return Holders Do not change the lock specified in the CDB, but report its state. The Client ID List in the reply data contains the Client IDs of the current non-expired holders of the specified Dlock.

Nop Return Expired Do not change the lock specified in the CDB, but report its state. The Client ID List in the reply data contains the IDs of the clients which expired while holding the specified Dlock.

Nop Return Conversion Do not change the lock specified in the CDB, but report its state. The Client ID List in the reply data contains the ID of the current holder of the Conversion lock for the specified Dlock.

Lock Shared Acquire the specified Dlock in the Shared state. If the the Client ID specified in the CDB does not hold the conversion for the lock, and there is a conversion holder, the action fails. If the lock is already held exclussively by another client, the action fails. If the action is unsuccessful in acquiring the Dlock, the conversion for the Dlock is acquired if available.

Byte,Bit	7	6	5	4	3	2	1	0
0	Operation Code (83h)							
1	Reserved			Action				
2	(MSB)	Lock Number						
3								
4								
5								(LSB)
6	(MSB)	Client ID						
7								
8								
9								(LSB)
10	(MSB)	Allocation Length						
11								
12								
13								(LSB)
14	Reserved							
15	Control							

Table 1: Dlock CDB

Code	Action	Description	Acts On	Client ID List
00h	Nop Return Holders	No change, return list of live holders	Lock Number	Holders
01h	Nop Return Expired	No change, return list of expired holders	Lock Number	Expired
02h	Nop Return Conversion	No change, return Client ID of conversion holder	Lock Number	Conversion
03h	Lock Shared	Acquire shared lock	Lock Number	Holders
04h	Lock Exclusive	Acquire exclusive lock	Lock Number	Holders
05h	Promote	Promote a shared lock to exclusive	Lock Number	Holders
06h	Unlock	Release lock	Lock Number	Holders
07h	Unlock Increment	Release lock and increment version number	Lock Number	Holders
08h	Demote	Demote an exclusive lock to shared	Lock Number	Holders
09h	Demote Increment	Demote an exclusive lock to shared and increment version number	Lock Number	Holders
0Ah	Refresh Timer	Refresh timer for Client	Client ID	None
0Bh	Reset Expired	Reset Expiration flags for a given Client	Client ID	None
0Ch	Report Expired	Report which Clients have expired	Whole Device	Expired
0Dh	Enable	Enable Dlock operation	Whole Device	None
0Eh	Drop Conversion	Removes the conversion on a lock	Lock Number	Holders
0Fh–1Fh	Reserved	Unused		

Table 2: Dlock Actions

Lock Exclusive Acquire the specified Dlock in the Exclusive state. If the lock is in the unlocked state and the Client ID in the CDB is the holder of the conversion (if any), the lock is acquired in the exclusive state. If other clients hold the lock, the action is unsuccessful. If the action is unsuccessful in acquiring the Dlockand the conversion for the Dlock is available, the conversion for the Dlock is acquired.

Promote Promote the specified Dlock from the shared state to the Exclusive state. If 1) the Dlock is in the shared state, 2) the Dlock is held by only the Client ID specified in the CDB, and 3) Client ID specified in the CDB is holder of the converion lock for this Dlock (if it is held at all), then the lock is promoted to the exclusive state. If the action is unsuccessful in acquiring the Dlock, the conversion lock for the Dlock is acquired if it is available.

Unlock Unlock the Dlock specified in the CDB. If the lock is held in a shared state and there are other holders, the state remains Lock Shared (but the holder count and Client ID list is changed).

Unlock Increment Behaves like the Unlock action, but also increments the Version Number for the specified Dlock.

Demote If 1) the specified Dlock is in the Lock Exclusive state and 2) the Dlock is held by the Client ID specified in the CDB, then lock is demoted to the Lock Shared state.

Demote Increment If 1) the specified Dlock is in the Lock Exclusive state and 2) the Dlock is held by the Client ID specified in the CDB, then lock is demoted to the Lock Shared state and the Version Number is incremented.

Refresh Timer This action heartbeats the device and prevents the specified Client ID (and all the locks it holds) from expiring.

Reset Expired Take the specified Client ID out of the Expired Lists of all Dlocks. The Client ID shouldn't show up in the results of any Nop (Expired) or Report Expired actions, unless the client expires again.

Report Expired Return a list of the IDs of all the clients which have expired while holding Dlocks on this device.

Enable This action allows the lock device to accept dlock commands. Until this command is issued to the drive, all commands fail. Enable is summarized more explicitly in the Dlock Concepts section.

Drop Conversion Remove any conversion on the Dlock specified in the CDB. This simplifies error recovery.

2.3 The Dlock Reply Data Format

The Reply Data Format for the Dlock command is shown it Table 3. Its parts are:

Version Number This is the version number of the lock.

Result This bit is one if the action succeeded and zero if the action failed.

Enabled This bit is zero when the lock device is powered on. It remains zero until the enable action is issued to the lock device.

List Type This field describes the type of clients returned in the Client ID list. The possible values of the field are *None*, *Holders*, *Expired*, and *Conversion*. The numerical representations of these values can be found in Table 5. The value returned in this field is dictated by the Action set in the Dlock CDB. Table 2 indicates which Actions result in which list types.

Have Conversion This bit is a one if the Client ID issuing the Dlock command possesses the conversion lock for this Dlock. It's zero otherwise.

Conversion This bit is a one if any client possesses the conversion lock for this Dlock.

State The values of the state of the lock are shown in Table 4.

Byte,Bit	7	6	5	4	3	2	1	0
0	(MSB)	Version Number						
1								
2								
3								(LSB)
4	Result	Enabled	List Type		Hv Conv	Conv	State	
5	Reserved							
6	(MSB)	Number of Live Holders						
7								(LSB)
8	(MSB)	Number of Expired Holders						
9								(LSB)
10	(MSB)	Client ID List Length (n – 11)						
11								(LSB)
List of Client IDs								
12	(MSB)	Client ID (first)						
13								
14								
15								(LSB)
...	(MSB)	...						
...								
...								
...								(LSB)
$n-3$	(MSB)	Client ID (last)						
$n-2$								
$n-1$								
n								(LSB)

Table 3: Dlock Reply Data Format

Code	Description
0h	Unlocked
1h	Locked Shared
2h	Locked Exclusive
3h	Reserved

Table 4: Values of the State field

Code	Name	Description
0h	None	No list is returned
1h	Holders	List of clients holding the lock is returned
2h	Expired	List of expired clients is returned
3h	Conversion	The ID of the client holding the conversion lock is returned

Table 5: Values of the List Type field

Byte,Bit	7	6	5	4	3	2	1	0
0	PS	Resvd	Page Code (29h)					
1	Page Length (0Ah)							
2	(MSB)	Maximum clients per lock						
3								(LSB)
4	(MSB)	Number of locks						
5								
6								
7								(LSB)
8	(MSB)	Client Timeout Interval (ms)						
9								
10								
11								(LSB)

Table 6: Mode Page Data

Number of Live Holders This is the number of un-expired clients currently holding this lock.

Number of Expired Holders This is the number of clients that expired when holding this lock.

Client ID List Length The length in bytes of the returned Client ID List.

Client ID List This is a list of Client IDs. The meaning of the list is specified by the List Type field (and the Dlock Action issued). If the value of the List Type field is *None*, this list is empty. If List Type is *Holders*, the list is made up of the current (un-expired) holders of the lock. If the List Type is *Expired*, the list is made up of the IDs of the clients which expired while holding this lock. (In the case of the "Report Expired" action, the list is made up of the IDs of all the clients which expired while holding any lock on this device.) If the List Type is *Conversion*, the list contains the ID of the client (if there is one) who holds the Conversion lock for this Dlock.

The number of IDs in the list is zero (for List Type *None*), "Number of Live Holders" (for List Type *Holders*), "Number of Expired Holders" (for List Type *Expired*), or zero or one depending on whether or not there is a holder of the conversion lock (for List Type *Conversion*). If the Allocation Length specified in the CDB is too small to hold the whole Client ID list, as much as possible is returned (with the "Client ID List Length" set to the appropriate value). This is not an error.

2.4 Mode Page

The Mode Page returns configuration information about several lock parameters. The page is shown in Table 6.

Maximum number of clients able to share a lock This is a 16-bit field, storing how many clients can simultaneously hold a lock in the lock shared state.

Number of locks on the device Returns the number of Dlocks on the device. If this value is hexidecimal 0xffffffff, the device supports a sparse lock space.

Client Timeout Interval The number of milliseconds after which a client ID will timeout. If the value is zero, client IDs never time out.

All dlock implementations must support changeable Client Timeout Intervals as part of the Mode Select Command. Implementers may optionally implement changeable Max Clients Per Lock and Number of Locks fields. Dlock devices can also support optional sparse lock spaces. Any Mode Select Command which sets one or more of the parameters in this mode page must also clear all locks on the device, and reset the enable bit.

2.5 State Transition Diagrams

The state transition diagrams are shown in Figure 1 and Figure 2.

Figure 1 This figure details the possible state changes possible due to normal lock operations. The conversion states are entered when a lock shared operation fails because the lock is held exclusively by another client, or when a lock exclusive operation fails because the lock is held by another client. If a conversion lock is held (i.e., the lock is in one of the conversion states), most actions fail that would normally succeed, except when issued by the conversion holder. The exceptions are unlock and demote.

Figure 2 This figure demonstrates the state transitions that occur when a client expires. These are not operations issued by a client, but are the actions performed by the lock device when a client ID fails to refresh it's timer within the timeout interval. Each arrow in this diagram represents a client timing out and therefore being removed from the holder list of the lock, and being placed in the expired holder list for the lock. Expired clients also lose any lock conversions.

References

[1] X3T10 SCSI committee. Document T10/ 98-225R1 – Proposed SCSI Device Locks. http:// ftp.symbios.com/ ftp/ pub/ standards/ io/ x3t10/ document.98/ 98-225r1.pdf, October 1998.

[2] Roy G. Davis. *VAXCluster Principles.* Digital Press, 1993.

[3] Andrew Barry et al. *Proposed SCSI Device Locks Version 0.9.5.* University of Minnesota, Parallel Computer Systems Laboratory, http:// www.globalfilesystem.org/ pubs/ dlock-0.9.5.ps, 1999.

[4] Kenneth Preslan et al. A 64-bit, shared disk file system for linux. In *The Sixteenth IEEE Mass Storage Systems Symposium held jointly with the Seventh NASA Goddard Conference on Mass Storage Systems & Technologies*, San Diego, California, March 1999.

[5] Kenneth Preslan et al. *Proposed SCSI Device Locks Version 0.9.4.* University of Minnesota, Parallel Computer Systems Laboratory, http:// www.globalfilesystem.org/ pubs/ dlock-0.9.4.ps, 1999.

[6] Kenneth Preslan, Steven Soltis, Christopher Sabol, and Mat thew O'Keefe. Device locks: Mutual exclusion for storage area networks. In *The Seventh NASA Goddard Conference on Mass Storage Systems and Technologies in cooperation with the Sixteenth IEEE Symposium on Mass Storage Systems*, San Diego, CA, March 1999.

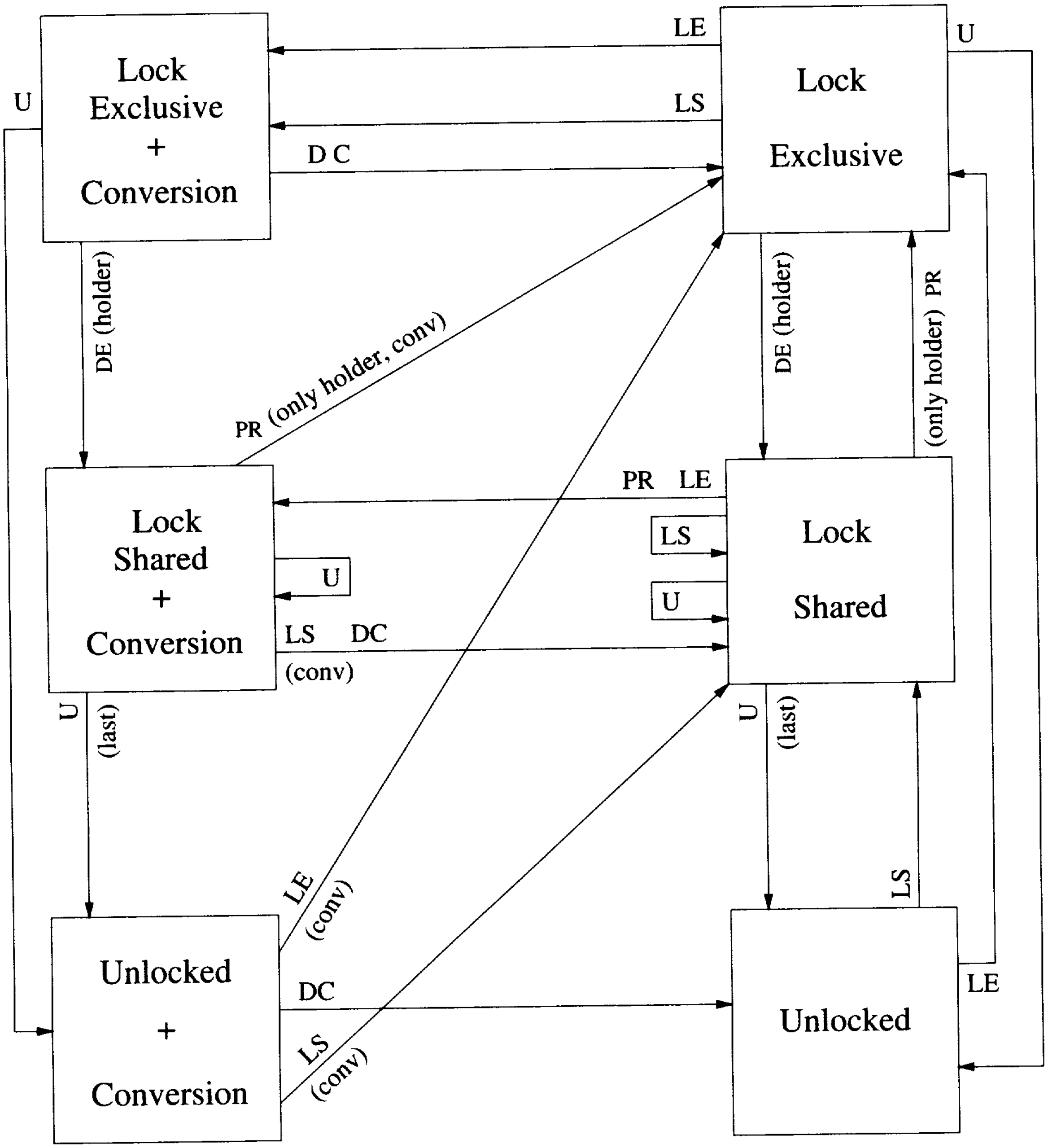


Figure 1: A state transition diagram that describes the possible transitions due to Lock Shared, Lock Exclusive, Unlock, Promote, Demote, and Drop Conversion actions. Unlock Increment has the same transistions as Unlock and Demote Increment has the same transitions as Demote.

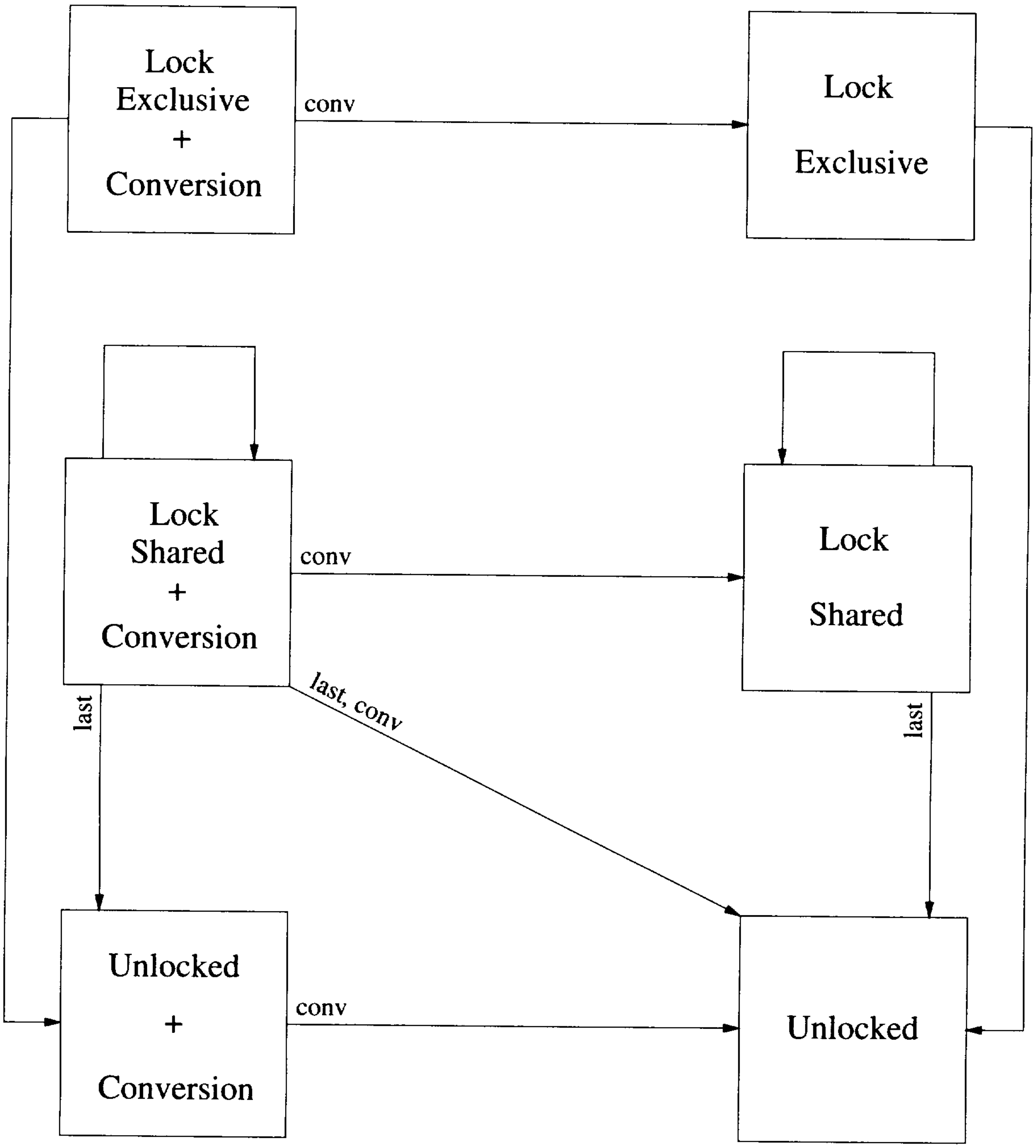


Figure 2: A state transition diagram that describes the possible transitions due to expirations.

A Scalable Architecture for Maximizing Concurrency

Jonathan M. Crawford
Lockheed Martin Space Operations
Upper Marlboro, MD 20774
jcrawfor@eos.hitc.com
tel +1-301-925-1074
fax +1-301-925-0651

Abstract

This paper describes a design that addresses limitations inherent in the initial implementation of the Archive for the Earth Observing Systems (EOS). The design consists of two elements: a Request Manager client interface and a Thread Manager design pattern. In combination, these design elements provide dramatic improvements in reliability and scalability. Moving to a transaction processing-based design also provides substantial gains for operability.

1 Introduction

The EOS Archive is a multi-site distributed data warehouse of Earth-oriented satellite images and science algorithms/reports. Data holdings in the archive are expected to exceed two petabytes by 2002 [1]. Within the EOS Core System (ECS), the Storage Management (STMGT) Configuration Item has the sole responsibility for moving data into and out of the archive, and for both the physical and electronic distribution of media.

The projected load on the system consists of over 76,000 granules per day – more than two terabytes – inbound to the archive, and over 30,000 granules per day of retrieved data. Each granule consists of one or more image files plus a metadata file. In aggregate, more than 400,000 requests will be serviced daily by STMGT servers. These requests are prioritized, and must be serviced in priority order, particularly as system resources become increasingly scarce.

The ECS software uses Distributed Computing Environment (DCE) Remote Procedure Calls (RPCs) as the mechanism for inter-process communications, with software processes deployed on both Sun and SGI platforms. Server processes are multi-threaded (using DCE threads), and accept requests from client processes via synchronous RPCs (see Fig. 1). Since most STMGT operations are I/O-intensive, these tend to be long-

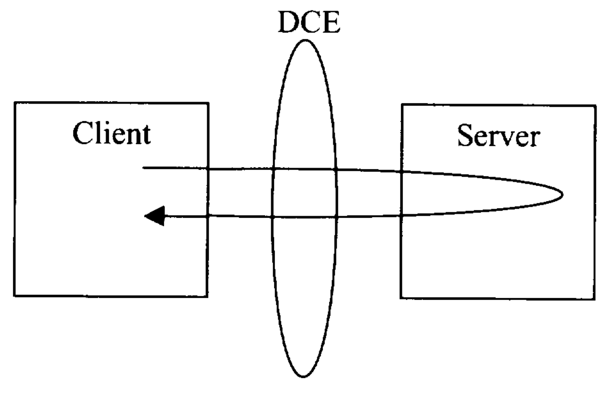


Figure 1: Current Design

running RPCs. Deployment of ECS has revealed significant scalability issues with the current STMGT implementation arising out of the use of synchronous DCE connections. DCE sizes the inbound request queue based on the number of listen threads available. Servers must either be configured with a large number of listen threads – consuming proportionately increasing system resources – or the number of concurrent requests must be limited, risking queue overflow and jeopardizing the system's ability to service the targeted request volume levels [2]. In some cases, the optimal number of listen threads is proportionate to the available resources. For example, a server that generates media for physical distribution of acquired data ideally has the number of listen threads equal to the number of devices available for generating media. However, due to the queue depth restrictions imposed by DCE, the number of listen threads must be configured to reflect the request queue depth, which may be considerably greater than the available resources.

2 Solution Elements

Our challenge was to redesign STMGT to improve scalability without disrupting client interfaces. At the same time, we sought to improve reliability and operability. This effort led to two major design elements: Request Manager, a means of preserving the existing client interfaces; and Thread Manager, a design pattern with highly desirable properties.

2.1 Request Manager

Request Manager provides the only DCE-based interface into STMGT. Existing STMGT client libraries were re-implemented to format requests as database transactions and submit them to the Request Manager via a single RPC. Request Manager executes each transaction, which checkpoints the request to a relational database and returns the server chosen to service the request. A single, centralized database is used for all STMGT requests, though multiple Request Manager processes may be used for request checkpointing and routing.

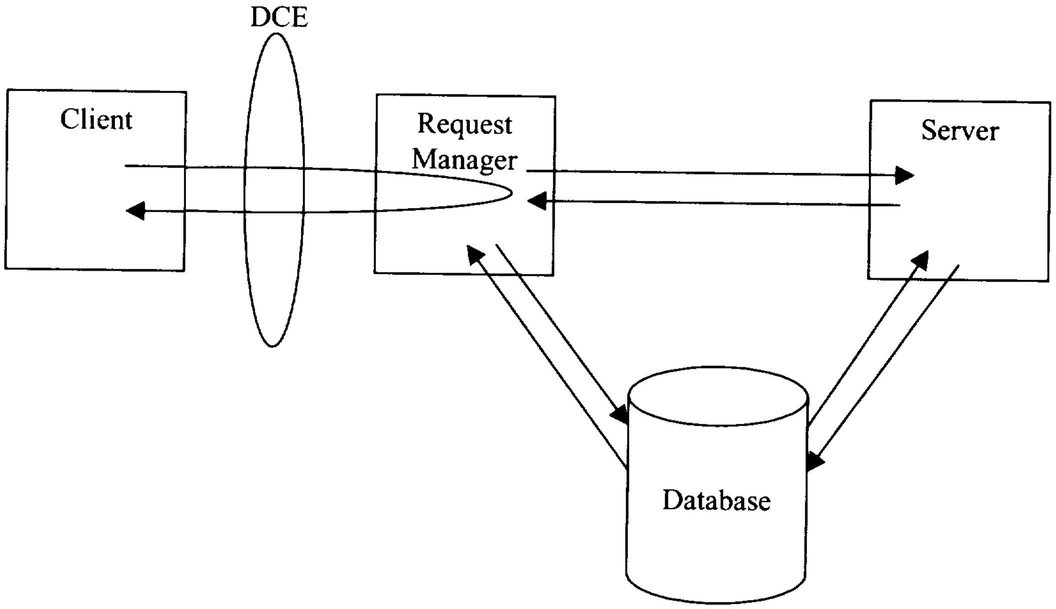


Figure 2: Request Manager Design

By using the database as the request queue, the Request Manager permits virtually unlimited requests to be queued. Moreover, priority-based dequeueing can be enforced via the SQL stored procedures that are used to select the next available request for servicing. Stored procedures are also used to implement the routing logic, permitting adjustments to routing to be made strictly through database patches.

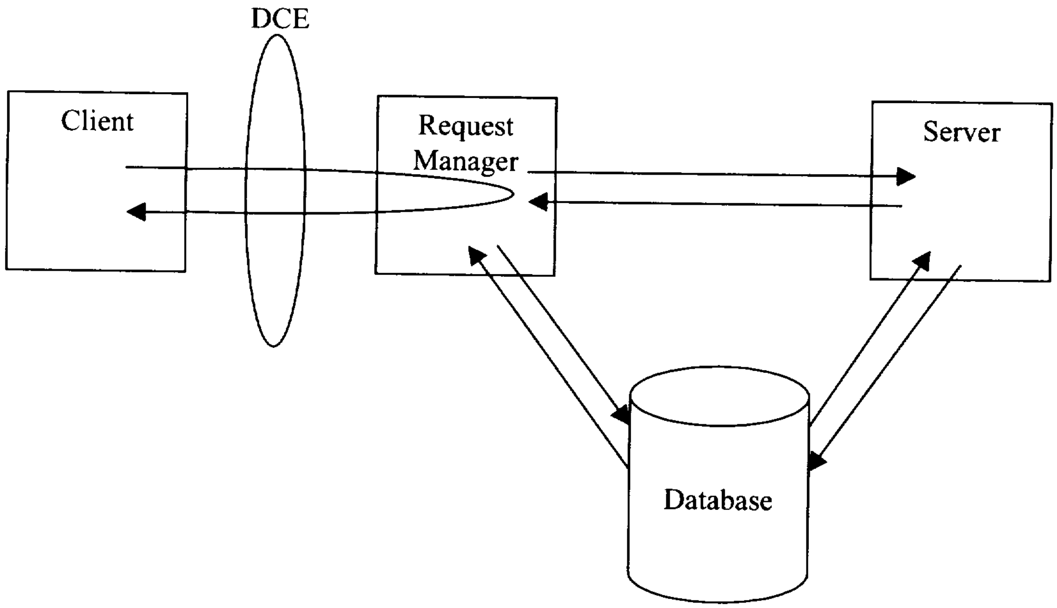


Figure 3: Request Manager Design

Once the server or servers have been identified for request processing, the Request Manager notifies each asynchronously using standard TCP/IP socket connections. No data is actually passed across the socket connection; the socket connection is merely used as an asynchronous event to notify the remote server that work is ready for processing. Figure 2 illustrates the interfaces within the new design.

2.2 The Thread Manager Design Pattern

The Thread Manager design pattern is comprised of four elements: a receptionist, a manager thread, and a pool of service threads. Upon creation, the receptionist allocates a port for listening and registers the server in the database as available for work. When the Request Manager or other STMGT servers assign work to the server, they notify the server by establishing a socket connection on the port on which the receptionist is listening. The receptionist then wakes the manager thread via a process-shared condition variable.

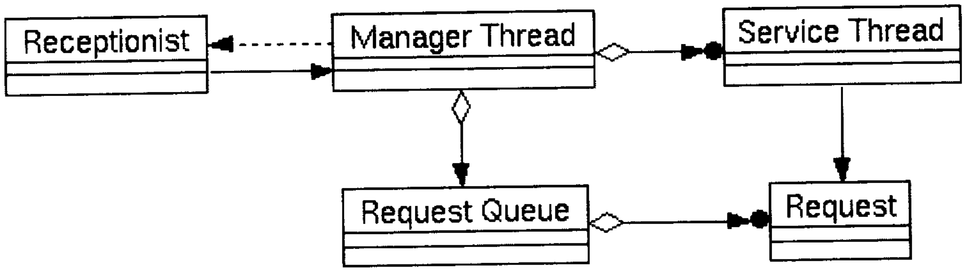


Figure 4: Thread Manager Design Pattern

The manager thread controls the assignment of requests from the queue to the pool of service threads. It waits on a condition variable until such time as it is notified by the receptionist that new work has arrived in the queue, or by a service thread that the service thread has completed its work and is ready for a new assignment. Restrictions are placed neither on the number of requests in the queue, nor on the number of service threads available for assignment.

Requests are assigned to service threads, which identify the type of work specified by the request and do the necessary processing to service the request. Service threads checkpoint as they perform request processing. If a request reaches a state whereby further processing must wait for another task to complete, the service thread will checkpoint the request in its current state and move on to another request. Once the related task has completed, the original request is released and re-assigned by the manager thread to a service thread.

In the Thread Manager design pattern, the pool of service threads is allocated at startup, and each service thread remains alive, even if it is idle. This eliminates the overhead of creating and destroying threads each time a request arrives. The number of service threads allocated to the pool need not be related to the expected volume of requests. For example, in media distribution servers, the number of service threads typically equals the number of devices available for writing. Each service thread may also be assigned a minimum priority for servicing, thus ensuring that a glut of lower priority requests does not starve higher priority requests.

3 Benefits

Software failover for STMGT is now supported through simple database mechanisms. When a server is brought down, a monitoring process automatically identifies the server as unavailable. When the alternate server instance is brought up, its receptionist registers it as available for work. Since the Request Manager relies on stored procedures to perform request routing, subsequent requests are automatically routed to the alternate server instance. On server termination, the manager thread can also be implemented to un-assign any requests associated with that server, thus enabling the dynamic reassignment of in-progress requests to alternate server instances.

The Request Manager design redefined request submission in terms of a checkpoint to the database, and the Thread Manager design pattern enforces checkpointing at each request state. This ensures maximal recoverability in the event of failure, while minimizing reprocessing.

In the prior implementation, clients notified each server when they were restarted after termination, in order to reclaim any persistent resources that may have been leaked. This caused particular problems for clients which retrieve data from the distributed archive, since such processes talk to a variety of server instances, and often had no record of which server instances they were in communication with at the time of termination [3]. With the Request Manager/Thread Manager approach, the client notifies the Request Manager, which then notifies all affected servers based on any requests that are still in the database for that client.

The Thread Manager design pattern provides a request dependency table. This permits requests to be related, so that processing of one request is suspended until one or more other requests have completed processing. When a request becomes dependent on another request, the service thread can abandon processing of the dependent request and work off other requests. This dependency tracking also eliminates redundant I/O, such as when multiple requests arrive concurrently to copy or FTP the same file.

The Thread Manager approach uses no polling, reducing the CPU load. It also relies on asynchronous communications between STMGT servers, eliminating the blocking and associated resource usage associated with long-running RPCs. Service threads are never left blocked, occupying memory resources or swap space and preventing other requests from executing while the blocked thread is effectively idle. In essence, the design is such that requests seek the I/O bottlenecks. Processing rates are anticipated to approximate the maximum hardware throughput supported by the physical configuration.

The centralized request tracking provides unprecedented operability gains. The STMGT GUI can now observe the processing progress of every request in every server. Future enhancements can utilize expected processing times to rapidly and automatically detect requests which are "frozen."

4 Future Work

The Request Manager was introduced as a process solely in order to preserve the existing, synchronous, DCE-based client interfaces to STMGT servers. However, the logic to checkpoint requests to the database and notify appropriate servers is encapsulated in base classes which are integrated into the Thread Manager design. All STMGT servers now use asynchronous client interfaces to communicate within STMGT, bypassing the Request Manager process entirely. As external clients to STMGT servers are able to adapt to an asynchronous request processing model, the Request Manager may eventually be removed altogether, supplanted by the set of asynchronous client interfaces which checkpoint requests directly to the database.

5 Conclusions

By using a transaction processing model within STMGT, we have dramatically increased both the operator's and our ability to monitor system activity. We expect to leverage these operational benefits by capturing check-pointed event sequences for regression testing and debugging. This ability to reproduce site-specific error conditions is expected to enable us to provide unprecedented turnaround time to correct complex errors that manifest at the deployed sites.

By replacing the DCE-based architecture with the Thread Manager design pattern, we were able to dramatically improve the scalability and reliability of ECS with regard to its storage management capabilities. As ECS continues to deploy satellites which feed the EOS Archive, the stability and reliability of STMGT will become increasingly crucial; we believe this new design will ensure the continued success of the ECS system.

5 Acknowledgements

The author would like to thank Donald Brown of Lockheed Martin Space Operations for his support and technical insights in the development of this new design.

References

[1] *Functional and Performance Requirements Specification for the Earth Observing System Data and Information System (EOSDIS) Core System* (http://spsosun.gsfc.nasa.gov/ESDIS_Pub.html, 1999) Appendix C.

[2] *OSF DCE Application Development Guide – Core Components, Rev. 1.1* (Cambridge, MA: Open Software Foundation, 1994).

[3] A Lake, J Crawford, R Simanowith and B Koenig. "Fault Tolerant Design in the Earth Orbiting Systems Archive", *Eighth NASA Goddard Conference on Mass Storage Systems and Technologies* (2000).

Using Track-Following Servo Technology on LTO Tape Drives

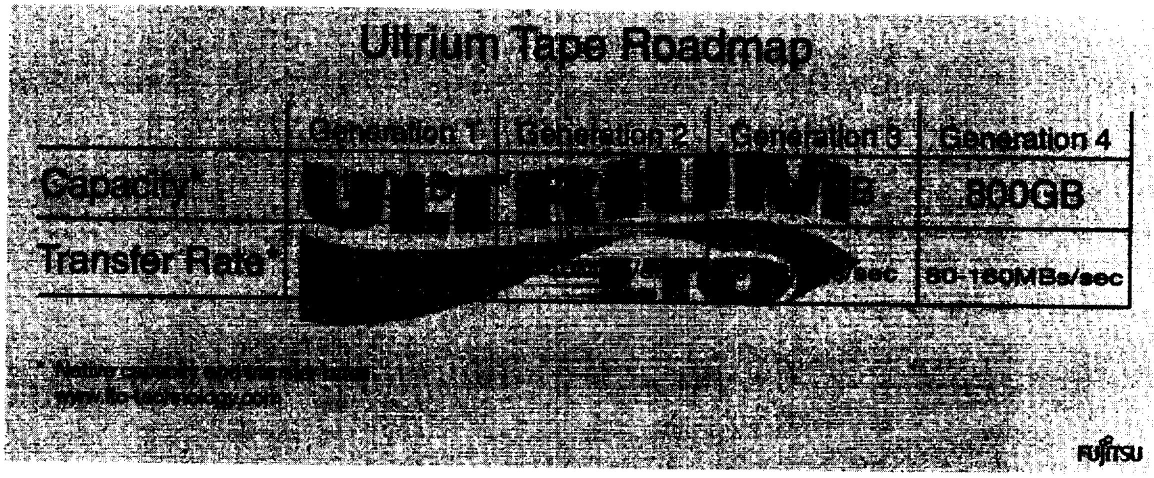
Randy Glissmann
Fujitsu Computer Products of America
1751 S. Fordham Street, Suite 100
Longmont, CO 80503
rglissmann@intellistor.com
tel +1-303-682-6555
fax +1-303-682 6401

Abstract

Fall COMDEX was the center of numerous showings of new tape technologies offering significant increases in tape capacity and performance. Of interest to users of midrange tape technology was the showing of Linear Tape Open (LTO) tape drives. Just as the upstart Linux operating system is nurturing a small but growing base of users, LTO-compliant products promise to secure a significant portion of the backup marketplace. The strength of LTO is in its organization and its technology. This paper will highlight these strengths and compare them to other tape products.

Introduction

In 1997, LTO technology was developed jointly by HP, IBM and Seagate to promote a tape technology that is accessible to any drive or media manufacturer through a license arrangement.[1] This is in contrast to the typical policy of tape manufacturers to keep their technology proprietary. Users will benefit from the cooperative and competitive environment created by an initiative organized to promote LTO. The technology used on LTO products is the result of the collective drive development expertise of the three founding companies. The roadmap of the technology is carefully plotted against future technologies to enable customers to replace their tape products for significantly higher capacities and performance without forgoing compatibility to earlier LTO written media.



Ultrium Tape Roadmap

	Generation 1	Generation 2	Generation 3	Generation 4
Capacity*	[illegible]	[illegible]	[illegible]	800GB
Transfer Rate*	[illegible]	[illegible]	[illegible]	80-160MBs/sec

* Native capacity and transfer rates
www.lto-technology.com

ULTRIUM
LTO
FUJITSU

Track-following Technology Enables Significant Capacity Increases

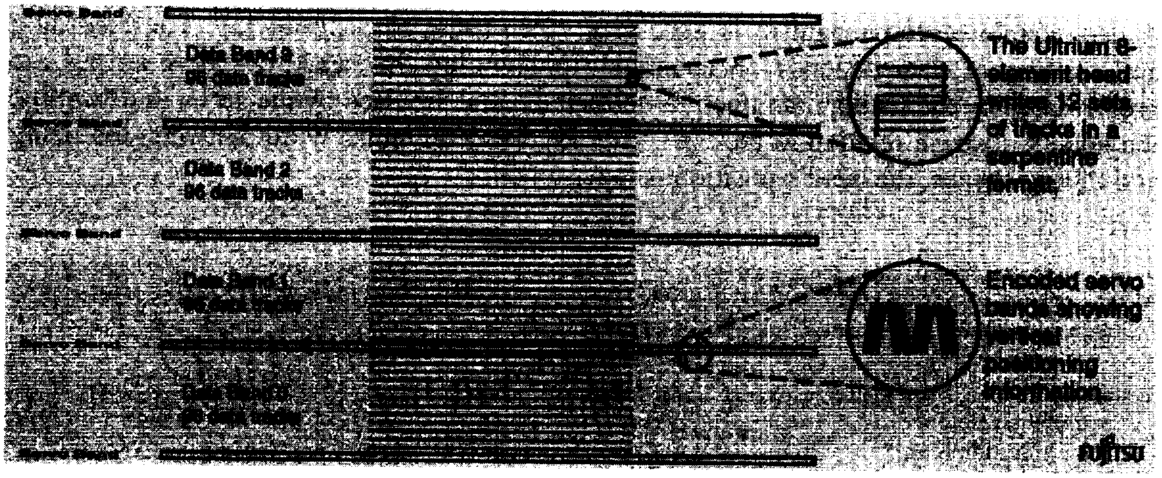
Track density, bit density, and media surface area are factors that determine the capacity that tape can support. Until recently, track density has been constrained on linear tape drives to around 400 tracks per inch due to the risk of overwriting neighboring tracks. Even with noteworthy engineering efforts to ensure accurate guidance of the tape moving across the recording head, overwriting and data recovery problems can be caused by small lateral tape movements which occur due to mechanical tolerances, environmental factors, and general wear of the media and of the drive mechanism. Increasing track densities only exasperates the problem.

In track-following technology, the head moves to compensate for any lateral tape movement so that higher track densities can be achieved. Current drives using track-following are IBM's Magstar^{TM} 3590 and Storagetek's 9840 drive which support 256 and 288 tracks respectively. Quantum's DLT8000 ^{TM} does not use track-following and achieves its high capacity through a high bit density along the track. Only 208 tracks are used on the DLT8000.

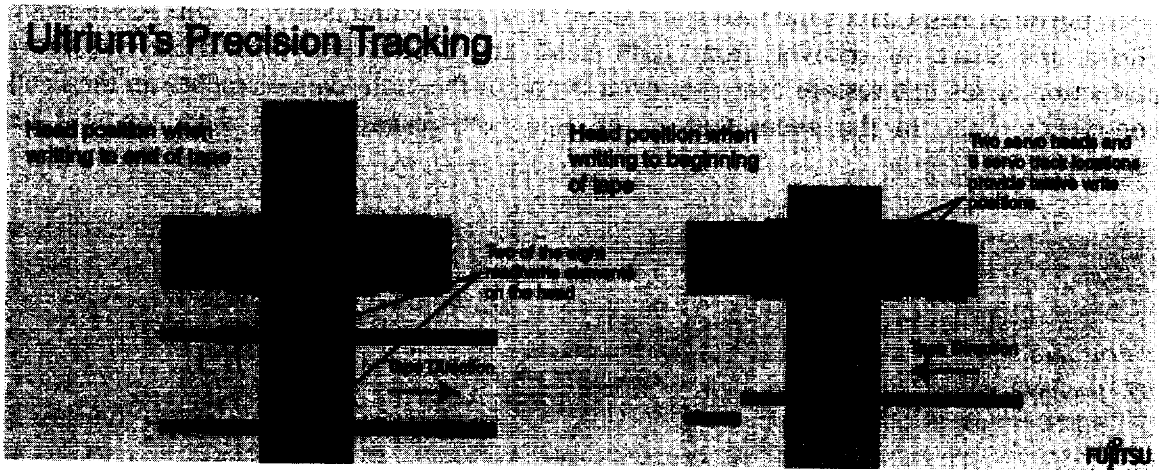
LTO has defined two tape technologies: Accelis and Ultrium. The Accelis format uses dual reels in its cartridge for fast time to data. The Ultrium implementation is similar to Magstar and DLT where a single reel holds a long length of ½" wide media. In addition to the three founding manufacturers, Fujitsu has indicated that they will manufacture Ultrium drives. Seven media manufacturers including Emtec, Fuji, Imation, Maxell, Sony, TDK, Verbatim, have indicated that they will produce Ultrium media. First generation Ultrium drives will be capable of storing 100GBs on a tape.

Quantum, the market share leader in tape drive shipments, has also indicated that their next generation Super DLTtape product will support at least a 100GB capacity. While both Ultrium and Super DLT technologies will use track-following techniques, their implementations are radically different. Table 1 shows the differences in their technologies.

[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]
[illegible] Redundancy	[illegible]	[illegible]
Media	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]
Recording Heads	[illegible]	[illegible]

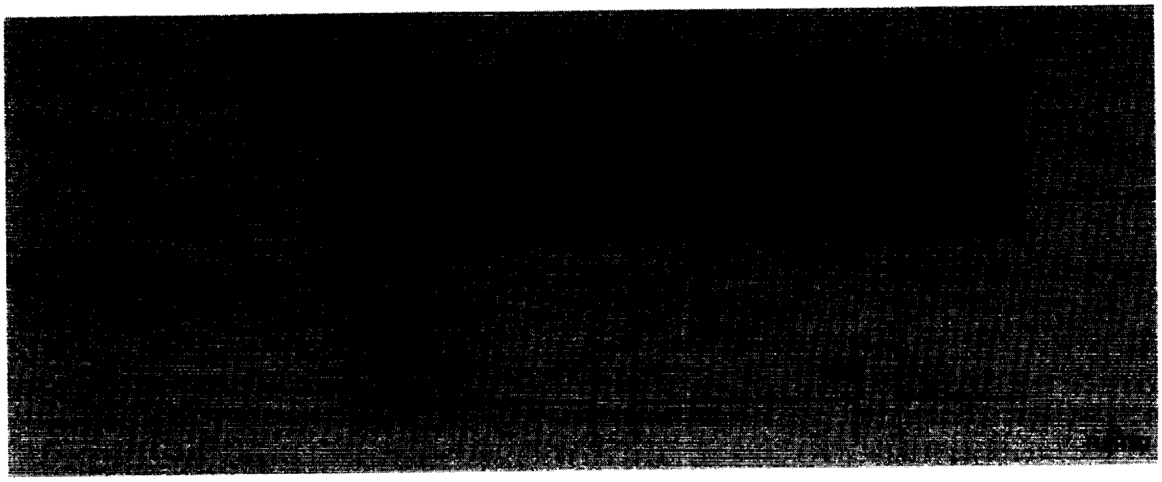


First generation LTO (Linear Tape Open) tape drives will achieve track densities of 768 tracks per inch.[2] Special elements on the drive's head monitor the servo tracks on the tape and detect whether the tape is moving laterally. The servo will automatically move the head to compensate for the tape movement. Additional elements on the head, used for reading and writing data, are located at precise locations relative to the reference tracks. Within four data bands, 96 tracks are written in twelve end-to-end passes over the media. Altogether across the four bands, 384 tracks are written. The heads are manufactured using photolithography, used commonly in fabricating integrated circuits, to provide the accuracy required enabling LTO's aggressive roadmap of increased capacity.


Ultrium's Precision Tracking

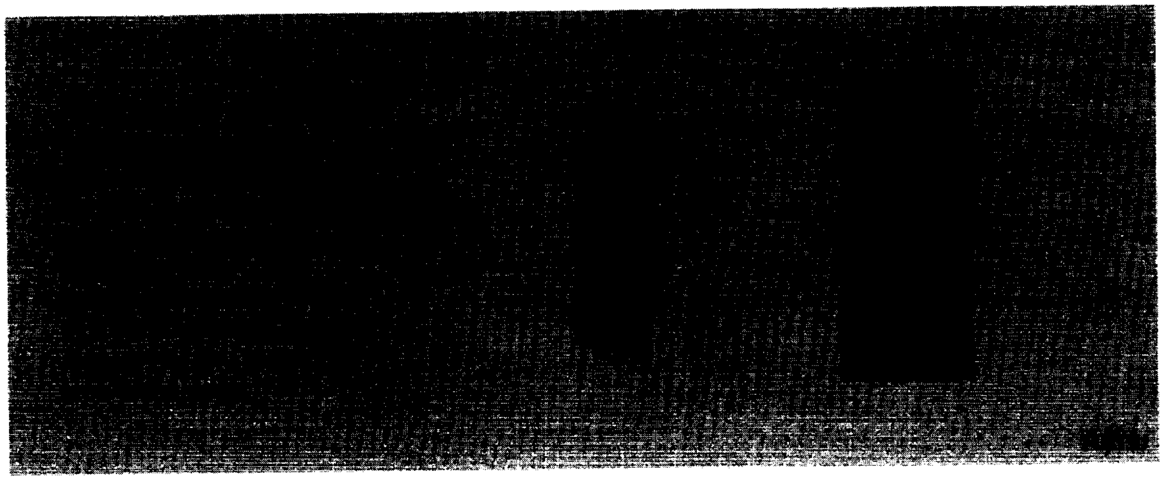
Head position when writing to end of tape
Two of the eight read/write elements on the head
Tape Direction
Head position when writing to beginning of tape
Two servo heads and 5 servo track locations provide relative write positions.
Tape Direction
FUJITSU

A dependable servomechanism is necessary when using track-following. The recording and reading of data depends on accurately positioning the head. Diagonal stripes are preformated by the media manufacture on the tape and are used by the servo to determine the head's position relative to the tape. By measuring the pulse width between the rising and falling stripes, the head's position is determined.

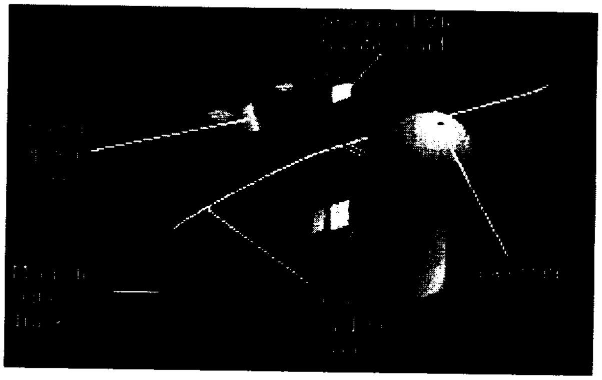


Reliability Inside

The recording head has two separate servo elements that can be used to determine the head position. Each data band also has a servo track above and below it. This redundancy allows the drive to recover from both head and media problems. Powerful error correction codes can restore user data even if one of the eight data elements on the head completely fails. In the event that 1" of tape is unreadable, user data can still be recovered.



Super DLTtape will reportedly use a revolutionary laser guided servo system.[3] Optical tracks are embedded on the back of the tape and a laser is used to detect lateral tape movement. Since the optical pickup and recording head are on opposite sides of the tape, maintaining mechanical tolerances without the accuracy provided by photolithography will be challenge.[4] Robustness will be an issue without multiple lasers to provide redundancy.



Source: Quantum

Conclusion

Track-following techniques will enable significant increases in tape capacity to be possible. By offsetting the position of data tracks from magnetic reference tracks, higher track density will be obtainable and the Ultrium format will be able to offer four generations of capacity increases. The level of redundancy that been designed into the Ultrium format will rival the robustness of mainframe tape products.

Trademarks:

Brand names are trademarks of their respective owners.

[1] For more information, see http://www.lto-technology.com.

[2] Teale, Altree, and Thompson; "Tape Drive Makers Adopt Open Format", Data Storage, June 1998: 51-54.

[3] Steve Scully, "Laser Guided Magnetic Recording; Carrying Super DLT into the next Millennium", THIC Conference, July 1999, http://www.thic.org/pdf/Jul99/quantum.sscully.990714.pdf.

[4] Randy Glissmann, "Fundamental Differences Between SDLT and LTO Tape Drive Servo Systems", Computer Technology Review, August 1999: 46.

Optical Head Design for 1TB Optical Tape drive

Mahdad Manavi
Aaron Wegner, Qi-Ze Shu, Yeou-Yen Cheng
LOTS Technology, Inc.
1274 Geneva Drive
Sunnyvale CA 94089
manavim@msn.com
tel +1-408-747-1111
fax+1-408-747-1687

Abstract

A multi-channel Optical Head has been developed to be incorporated with an Optical Tape Drive that enables greater than 1TB capacity and greater than 25 MB/s data transfer rate on a 0.5 mil thick, 0.5 inch wide optical tape that fits in a 3480 style cartridge. The optical design has been optimized to perform at 532 nm with maximum through put in the write path. With use of a 0.6 NA objective lens, sub-micron size marks can be written on a WORM type phase change medium. A single laser, a hologram, and a multi-channel modulator increase reliability and manufacturability of this design.

Introduction

The need for a high capacity and a fast data transfer rate tertiary storage device is continually increasing. With the abundance of inexpensive high capacity magnetic disk drives, the explosive growth of information content on the Internet, and the need to save and keep data electronically for a long period of time, an increasing demand is continuously generated for high capacity tertiary data storage devices. We are developing a high performance Optical Tape Drive at LOTS Technology as a solution to this huge data storage problem. This Optical Tape Drive provides a nominal 25 MB/s data transfer rate and a nominal 1TB capacity in a 3480 style cartridge form factor.

Some of the advantages of optical tape drive are as follows. Optical tape provides much higher capacity per removable media unit than any storage device in the market. Very high data transfer rate is readily available for a given tape speed with increasing number of laser beams acting as parallel channels. Media is archival with greater than 30 years life span. The optical tape drive offers non-contact recording and read-back which result in zero head and tape wear. The tape transport mechanics have minimal physical contact with the tape further enhancing tape operational lifetime by reducing tape wear. Last but not the least, the high tape capacity results into fewer media mounts; thus causing less wear in robotics and cartridge.

The Optical Tape Drive consists of an optical head, a tape transport, and electronics. A multi-channel Optical Head has been developed to be incorporated with the Optical Tape Drive that supports the underlying required system specifications. I will discuss the intricate system design tradeoffs as they relate to the Optical Head design.

Design approach

In optical data storage, minimum mark spacing is one of the factors that determine the smallest signal amplitude available to the read channel. In a conventional peak detection method, the convolution of the read beam with written marks determines the data signal amplitude. Once minimum mark spacing and track spacing are established, the capacity is determined less the overhead. Obviously encoding efficiency is also a contributor to the bit density calculation.

To set a data transfer rate, two degrees of freedom are available. Once is the tape speed and the other is number of channels. At high tape speeds, the tape control becomes difficult and optical tracking servo bandwidth may become unreasonable. For a large number of channels, a flat field objective lens becomes necessary to focus the beam array to the flat tape surface. However, to achieve a high capacity, it is desirable to use a high numerical aperture (NA) objective lens to allow recording of sub-micron size marks. Unfortunately, increasing the numerical aperture of the objective lens and increasing the field flatness go against each other. In other words, the design task becomes extremely difficult to fulfill both criteria of high NA and field flatness. Therefore, the final design becomes a compromise among the tape behavior, optical tracking bandwidth, and the objective lens design.

Table 1 shows some options in changing the minimum mark spacing and track spacing to achieve the required capacity. For a 1 TB (Terra Byte) user capacity and 35% overhead, the raw capacity must be 1.35 TB. This is equivalent of 10.8 Tbits. For 450 m of tape, the areal density becomes 1.44 Gbits/in^2. Keeping this number constant, one can calculate various mark spacing and track spacing that result in 1 TB user capacity. Figure 1 shows various combinations of number of channels and tape speed for a given minimum mark spacing to achieve desired user data transfer rates. PPM (2,7) channel and 35% overhead were assumed for this calculation.

minimum mark spacing (um)	TPI	track spcing (um)
0.8	30236	0.84
0.85	32126	0.79
0.9	34016	0.75
0.95	35906	0.71
1	37795	0.67

Table 1 –Mark spacing & track spacing that result in 1 TB capacity

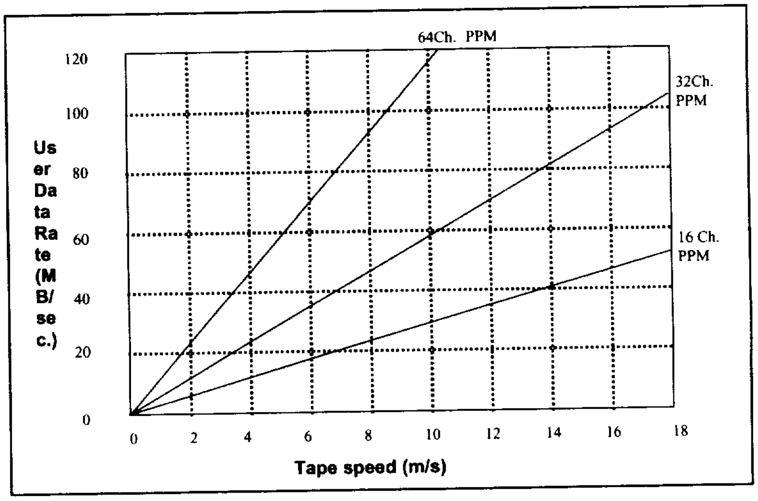


Figure 1 –Number of Channels and Tape Speed Versus Data Transfer Rate

Determining factors on choice of minimum mark spacing and track spacing are data optical resolution and track to track data crosstalk.

Drive structure

Drive layout is shown in Figure 2. It consists of an optical head, a tape transport, and supporting electronics

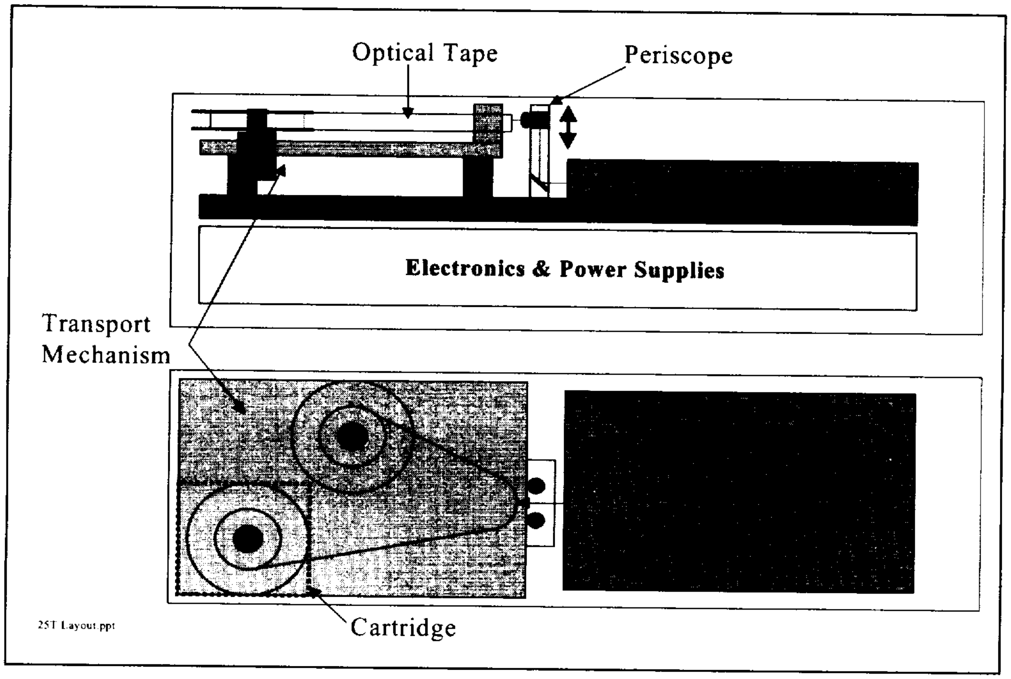


Figure 2 – Optical Tape Drive Layout

Focus of this article is to discuss some intricate design approaches for the optical head.

Optical Head Structure

The optical head is the instrument that writes and reads information onto and from the tape. Its design must support the system level specifications: capacity and data transfer rate. It must also be able to handle tape motion in terms of maintaining tight focus and track servos during record and read back. A simple optics layout is shown in Figure 3.

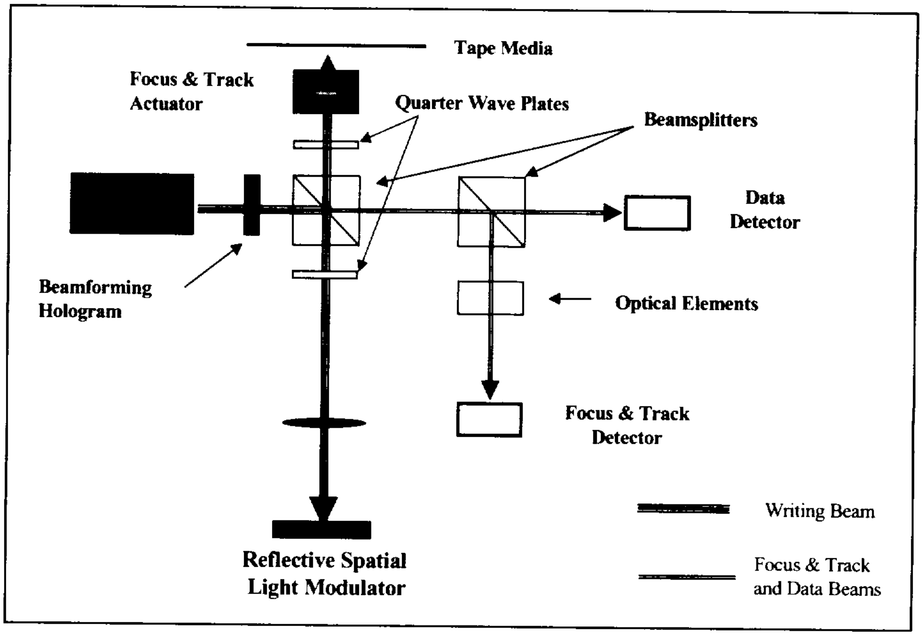


Figure 3 - A simple optics

A single laser is used as the beam source. It is a diode pumped doubled YAG laser generating a beam with a wavelength of 532 nm. The choice of doubled YAG laser, as an added benefit, removes the necessity of a wavelength migration path from a red laser diode to a green or blue laser diode. However, more important than the elimination of the need for a new optical design (due to removal of the wavelength migration path) is avoidance of all the reliability issues and the beam profile variations of a high power laser diode. Figure 4 shows a 200 mW doubled YAG laser used in this design.

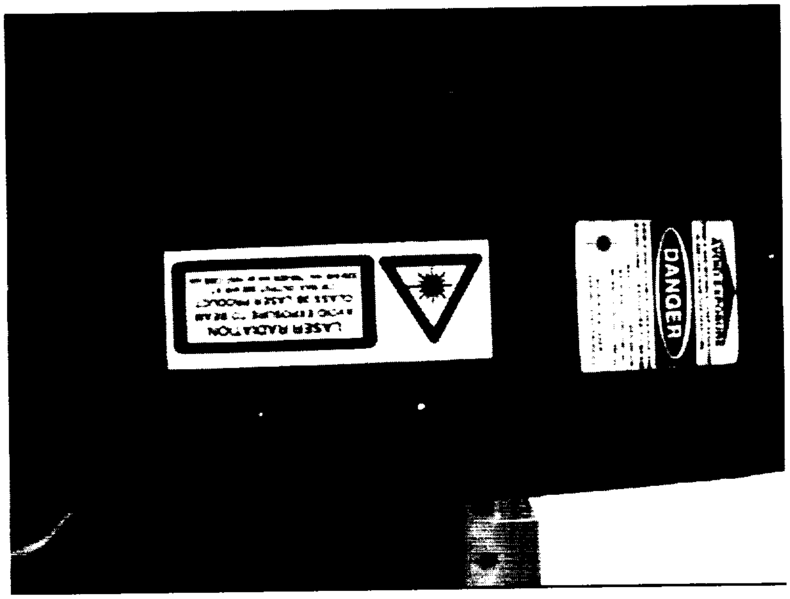


Figure 4 – Doubled Nd:YAG Laser

A 2-dimenational hologram multiplies the single laser beam into an array of collimated beams each of which is capable of read/write operation. Combination of the single laser with a hologram to generate multiple beams increases reliability of the system in contrast to using a 2-dimensional laser diode array. Furthermore, collimation optics design becomes simpler in the former case than the latter case.

The laser beam array after the hologram is imaged onto an electro-optic modulator array, each of which upon activation rotates polarization of the incident laser beam by 90 degrees. Data from the host computer is encoded and fed to the modulator driver. Each laser beam is pulsed according to the streams of ones and zeros that it receives from the electronic write channel. The polarization modulated laser beam array will become intensity modulated after passing through some polarization optics as the beam array reaches the tape plane. The laser beams that are "on" will write mark on the tape, and the beams that are "off" will not.

Figure 5 shows electrical and optical pulses as input and output from a single modulator. The optical pulse rise time shown in the laboratory is about 7 ns, which was limited by the electrical response of the system. Figure 6 shows optical response of four modulators running simultaneously at 12 MHz. The pulses are square shaped and well defined.

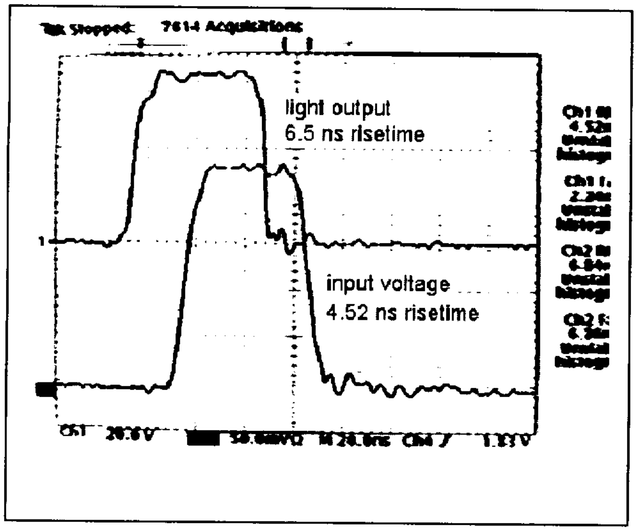


Figure 5 – Voltage Input & Optical Output

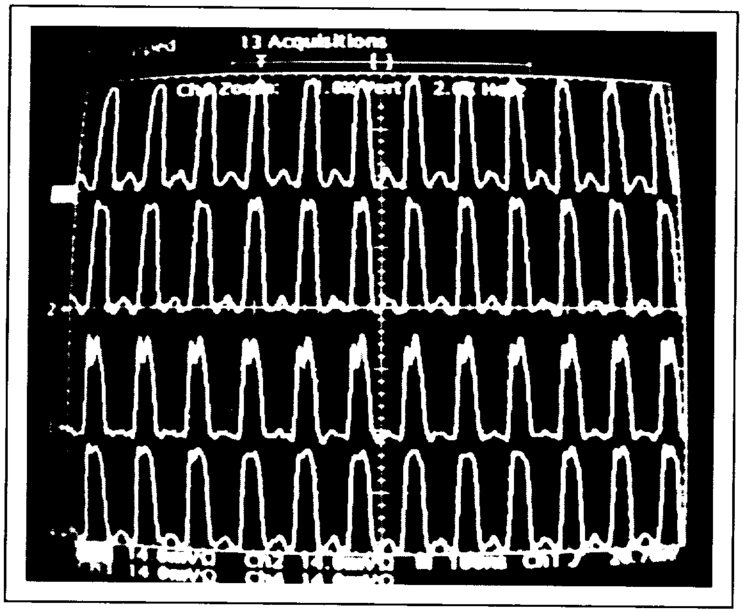


Figure 6 – Optical Pulses at 12 MHz

To decrease access time in traversing width of the tape, the optical head design is split in two parts: fixed optics, and moving optics. The fixed optics consists of the laser, hologram, various passive optics, and detectors. The moving optics is a periscope in effect, which includes a tracking actuator, a focus actuator, and a high NA objective lens. A stepper motor driven carriage that has attached to it the focus and the track actuators traverses width of the tape for various read/write operations. Figure 7 shows the concept of the periscope design.

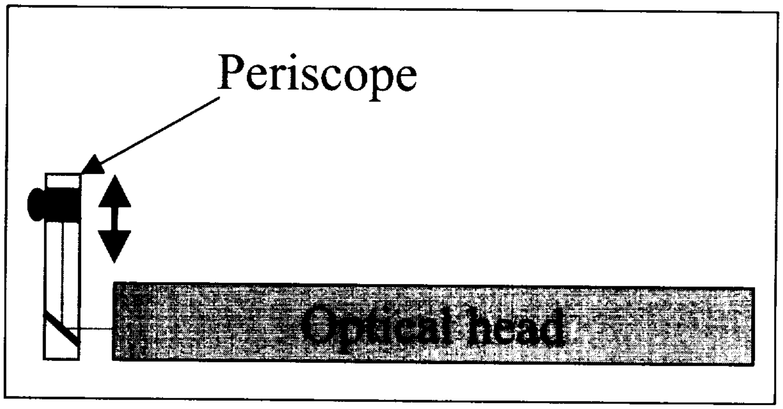


Figure 7 – Periscope Layout

A high bandwidth galvanometer has been designed as the track actuator. As the beam array reflects from the mirror that is attached to the galvanometer, and as the galvanometer rotates about a symmetry axis, the beam array pointing changes; thus, causing a beam displacement at the tape plane. The purpose of the track actuator is to make sure that center of the read beams is always at the center of written tracks.

The focus actuator has a high numerical aperture (NA) objective lens attached. The purpose of the focus actuator is to maintain a minimum spot size at the recording layer of the tape at all times by moving the objective lens along the optic axis relative to the tape plane.

A high NA objective lens has been designed to accommodate a large field. All the beams within the array must be at best focus at the tape plane. In general, a spherical lens forms an image of an extended object on a spherical image plane. Furthermore, as the numerical aperture of the lens increases, the filed flatness of the image plane reduces. However, in our case, the spot array forms a rectangle of dimensions of 6 x 50 μm^2 on the tape surface. The tape thickness is 12.7 μm. Therefore, the aspect ratio of thickness to area necessitates a flat region where the spot array is incident. This issue required us to design a 0.6 NA objective lens that has a flat field up to 1 degrees. Figure 8 shows wavefront error (WFE) data of a fabricated 0.6 NA lens using an off the shelf interferometer.

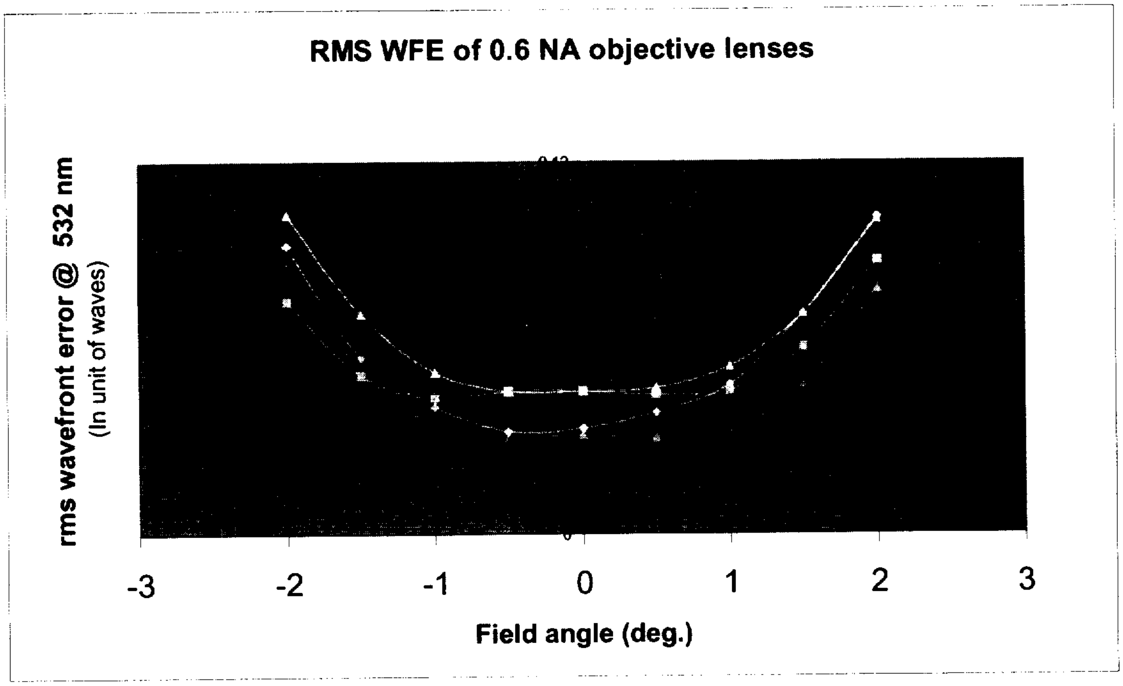


Figure 8 – WFE of the 0.6 NA Objective lens as a function of field angle

Media Structure[1]

The active material is a Kodak proprietary vacuum deposited amorphous thin film of SbSnIn (Antimony, Tin, Indium) alloy. Recording of data is accomplished by heating with a focused laser beam to form sub-micron size crystalline marks. This is possible because the crystallization rate of amorphous SbSnIn alloy thin film depends strongly on temperature. Heating the material to about 250 °C using a focused laser beam causes an instantaneous crystallization while an undisturbed amorphous film stays amorphous almost indefinitely. Figure 9 shows the media structure.

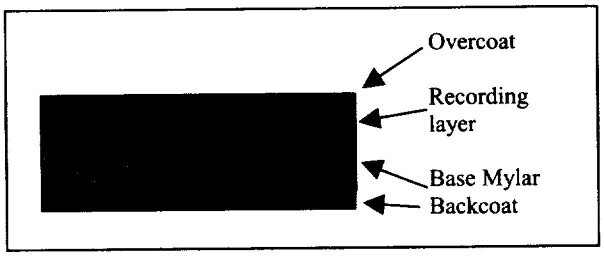


Figure 9 – media structure

The media is very simple in structure. It consists of a Mylar based web sputter coated with the alloy thin film (the recording layer), and then coated with an overcoat for protection of the recording layer. There is a backcoat applied to the other side of the base to avoid electrostatic charge accumulation. Base reflectance of the optical tape is a function of the overcoat thickness. Figure 10 shows the optical tape reflectance as a function of the overcoat thickness for both amorphous and crystalline states.

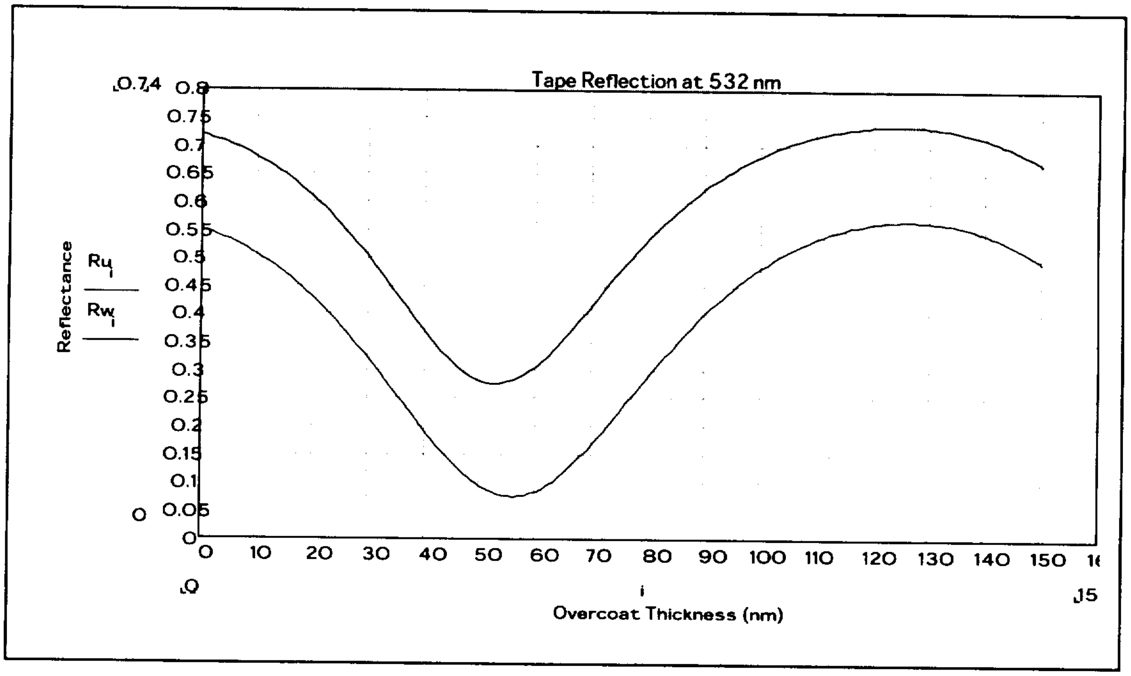
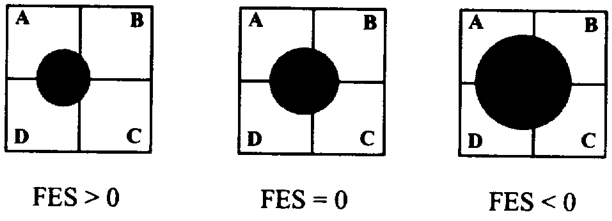


Figure 10 – Tape Reflectance versus Overcoat Thickness at 532 nm

Experimental data

An optically generated error signal controls the distance between the objective lens and the tape plane by moving the focus actuator. Using a modified Wax-Wane method in which the servo beam is offset relative to a bicell, Figure 11, a focus error signal is generated. Size of the servo beam spot at the detector changes as the distance between the tape and the objective lens varies from nominal. It is this size change that is detected, and is converted to an S-curve for focus error detection.

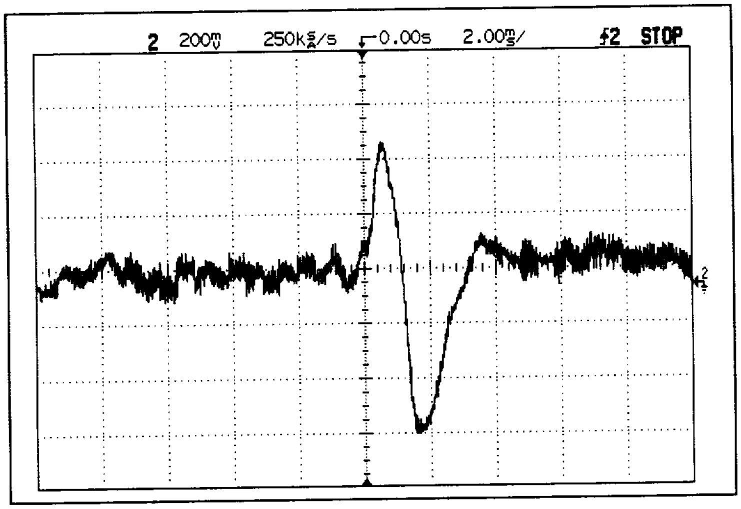


FES = (A + D) – (B + C)*G

Where G is an electrical gain factor

Figure 11 – Modified Wax-Wane schematic

Figure 12a shows the focus S-curve, which has a capture range of 3 μm. Figure 12b shows the residual focus signal while the servo loop is closed with the optical tape running at 10 m/s. As can be calculated, the residual focus error signal is 0.1 μm peak to peak, which is well within depth of focus of the objective lens.



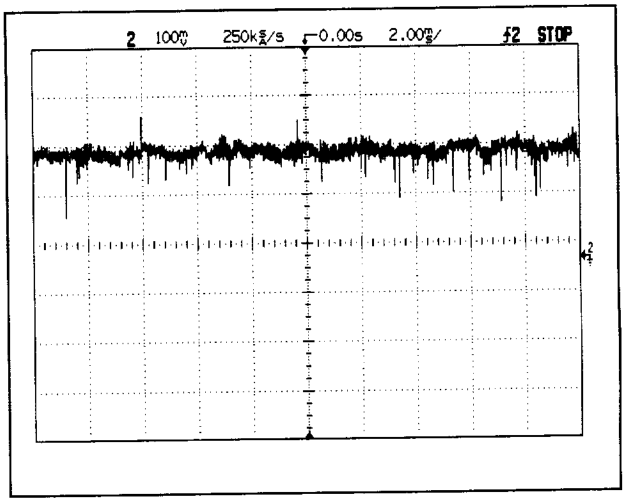
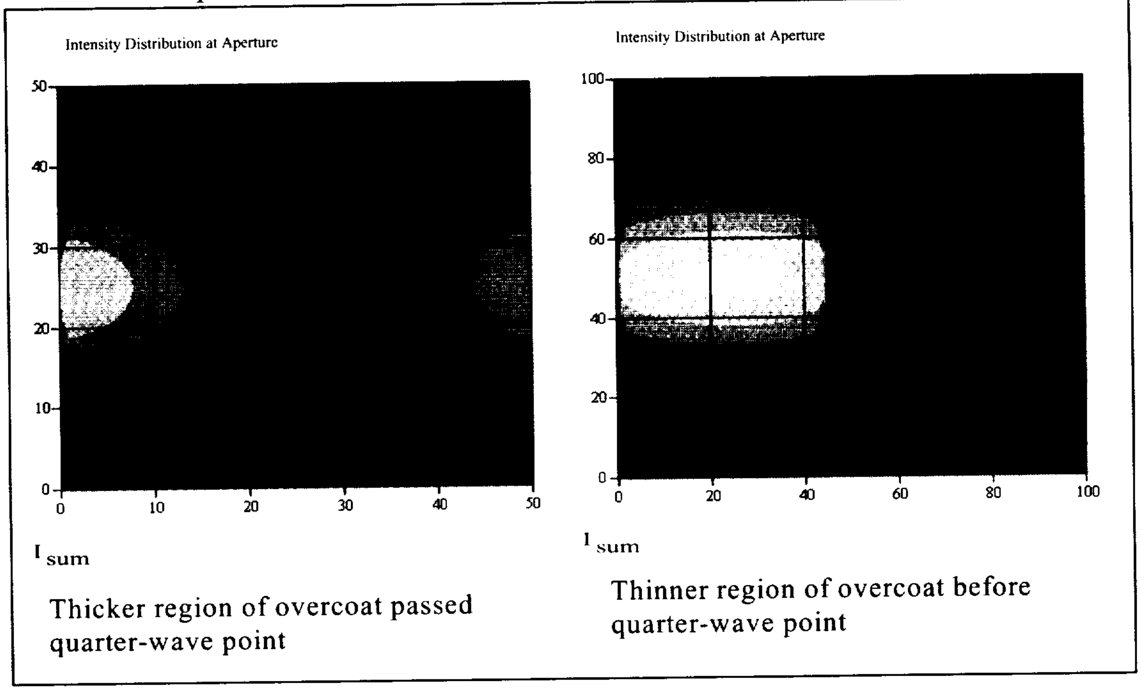


Figure 12a – Focus S-curve **Figure 12b - Residual Focus Signal at 10 m/s**

Tracking error signal is generated by using diffraction and interference of coherent light from a grating; otherwise, known as the push/pull method. Since the crystalline structure of the phase change material on the tape has a different complex index of refraction than the amorphous structure, a series of written lines act as a weak phase grating. The tape overcoat can be optimized for maximum push/pull track error signal for a given



Thicker region of overcoat passed quarter-wave point

Thinner region of overcoat before quarter-wave point

Figure 13 – simulation result for push/pull TES versus overcoat

wavelength. Figure 13 shows simulation of intensity distribution of reflected beam from a written tape at the plane of the objective lens aperture. The interference between diffracted orders, 0th and ± 1st, generates bright or dark intensity profiles that when detected properly results in the push-pull tracking error signal. The Figure on the left-hand side shows a weaker contrast than the one on the right hand side indicating a weaker push-pull TES. Table 2 lists the tracking error signal amplitude normalized to the servo sum signal for various tape overcoat thicknesses as measured in the laboratory. The trend found in the simulation is validated by the experimental results.

overcoat Thickness (nm)	NTES (V)
33	4.1
50	4.2
78	0.6
89	0.7

Table 2 – Experimental result for push/pull TES versus overcoat

Figure 14a shows the tracking error signal S-curve derived from four parallel data lines written. The pattern is pseudo random with a maximum frequency of 10 MHz. Figure 14b shows the residual track error signal with both the focus and track loops closed and tape traveling at 10 m/s. The top trace is the residual tracking error signal, and the bottom trace is a data line. In this example, the track spacing is 0.8 um. Therefore, one can calculate the residual error signal to be 0.07 um peak using a sine wave approximation.

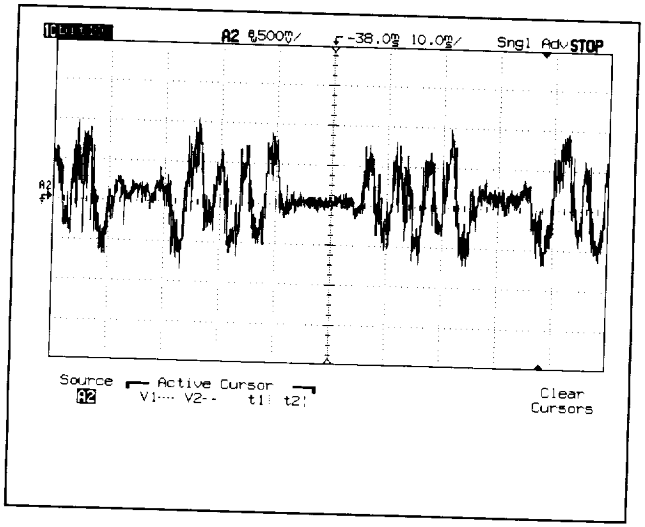


Figure 14a – Track Error Signal from 4 Data Lines

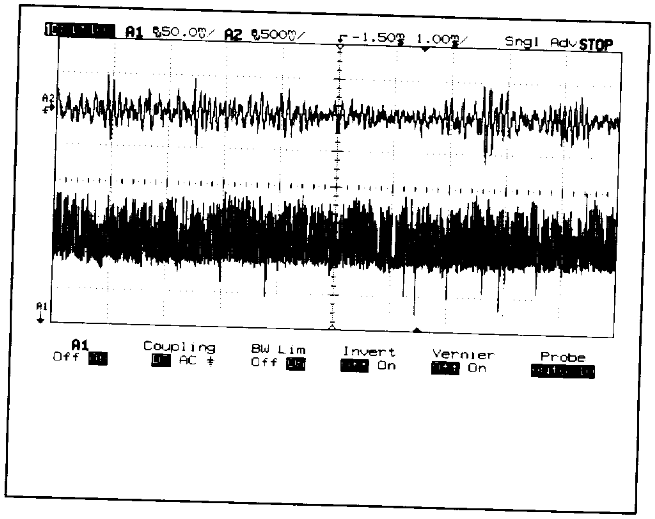


Figure 14b – Residual Tracking Signal

Figure 14c is a magnified version of Figure 14b where individual data marks are shown.

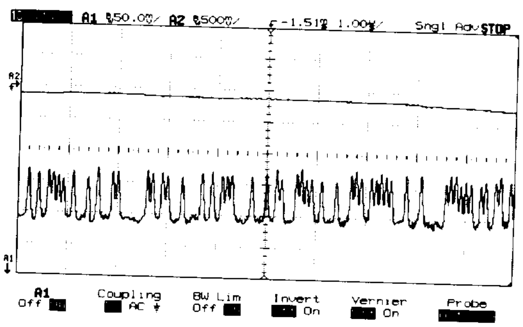


Figure 14c – Magnified version of Figure 14b

As was mentioned before, the writing process takes place by modulating the laser beam array through an electro-optic modulator. Each beam is modulated independent of rest of the array members. To insure proper spacing of the written marks on the tape, the beam array must be skewed relative to the tape direction of motion. For example, if there are m columns and n rows in a beam array, then the array must be tilted by $\tan^{-1}(1/m)$ to insure even spacing among the written tracks. Figure 15 illustrates this point.

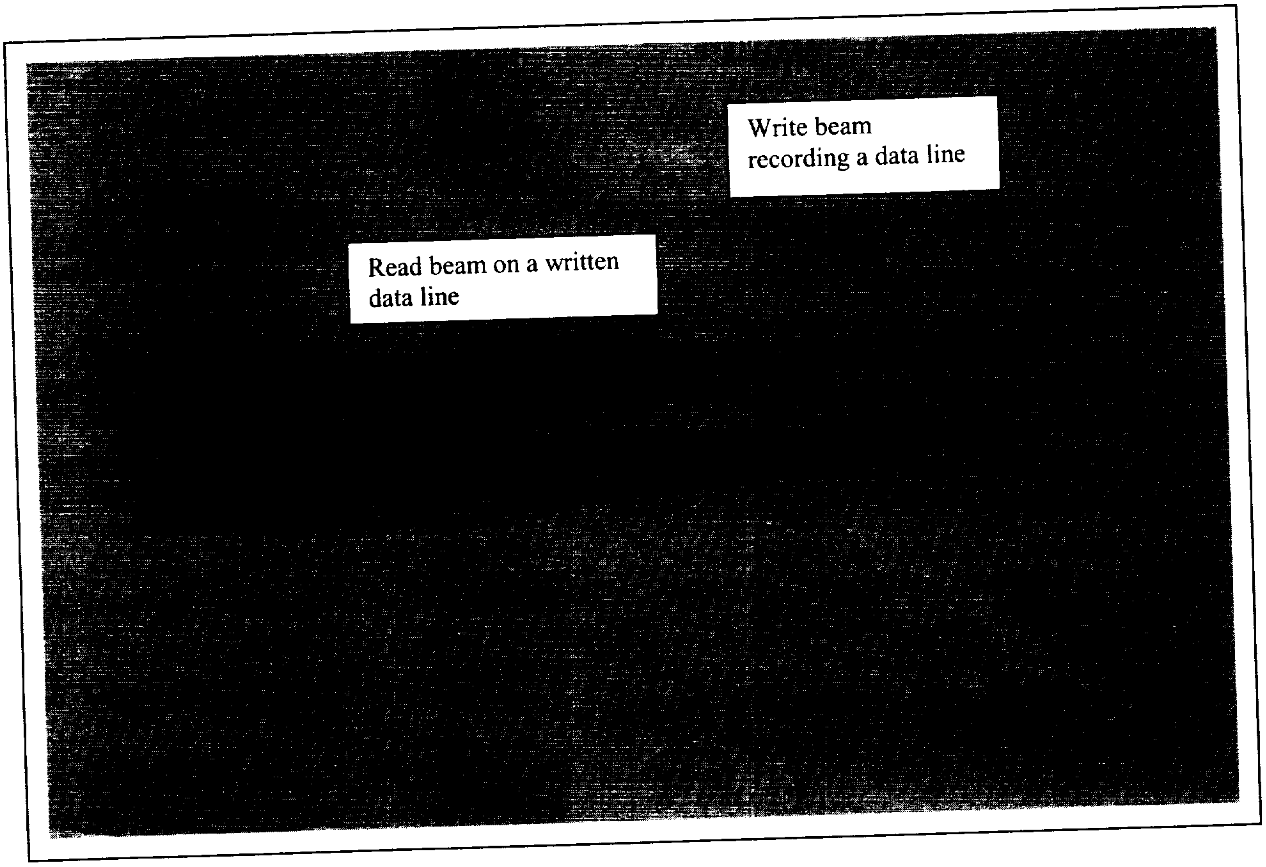

Write beam recording a data line
Read beam on a written data line

Figure 15 – Beam Array Skew

Although there are many ways of skewing the beam array relative to the tape, in this paper, the optical head was tilted by 7.125 degrees.

In one example, 8 solid lines with 0.8 μm spacing were written simultaneously on the tape at 10 m/s, which is shown in Figure 16.

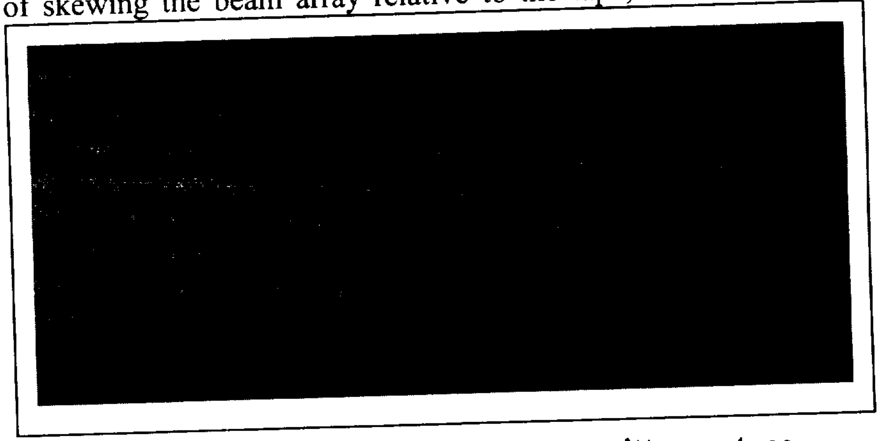


Figure 16 – picture of 8 solid lines written on tape

Figure 17a shows many track groups of 4 data lines written on the tape at 10 m/s. The minimum mark spacing is 1 μm and track spacing is 0.8 um in this experiment. Figure 17b is an enlarged photo of a four line simultaneous recording. In this picture the minimum mark spacing is 0.85 μm.

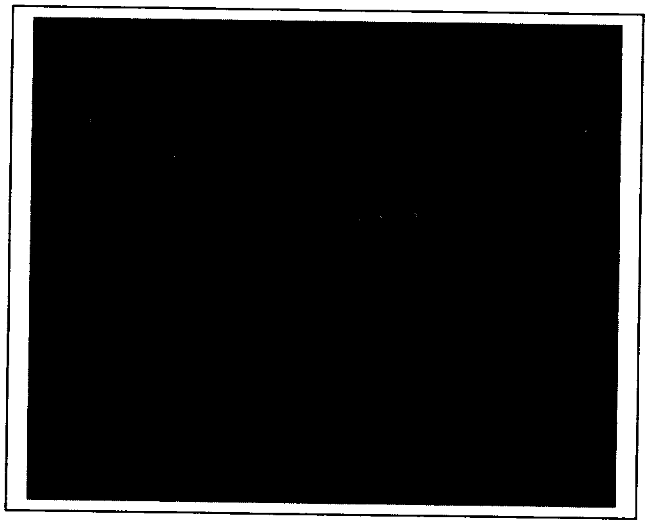


Figure 17a – multiple 4 data lines line at 10 MHz, 10 m/s

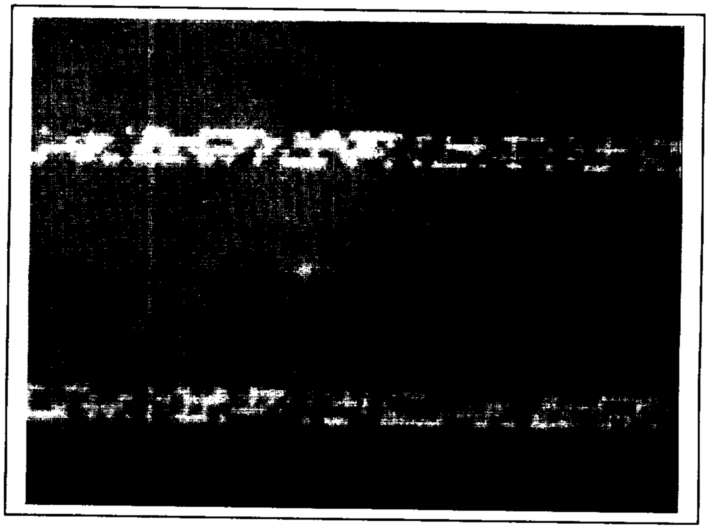


Figure 17b – Magnified version of Figure 16a

Optical resolution simulation is validated by the experimental data as is shown in Figures 18a, b respectively. Optical resolution is defined as the ratio of the amplitudes of the highest spatial frequency signal to the maximum signal amplitude. An optical resolution of 50% was simulated for a perfect system. However, given that the real world is not as perfect and optical aberrations do exist, which degrade the read/write focused spot quality, a measured optical resolution of 40% would be reasonable. Experimental results show an optical resolution of greater than 40%. The upper trace in Figure 18b is the residual track error signal.

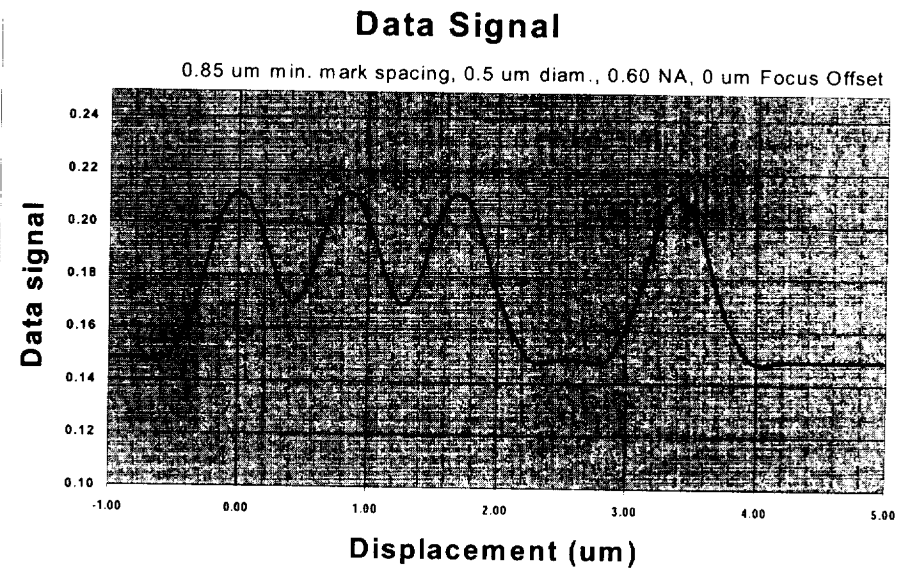


Figure 18a – optical resolution simulation

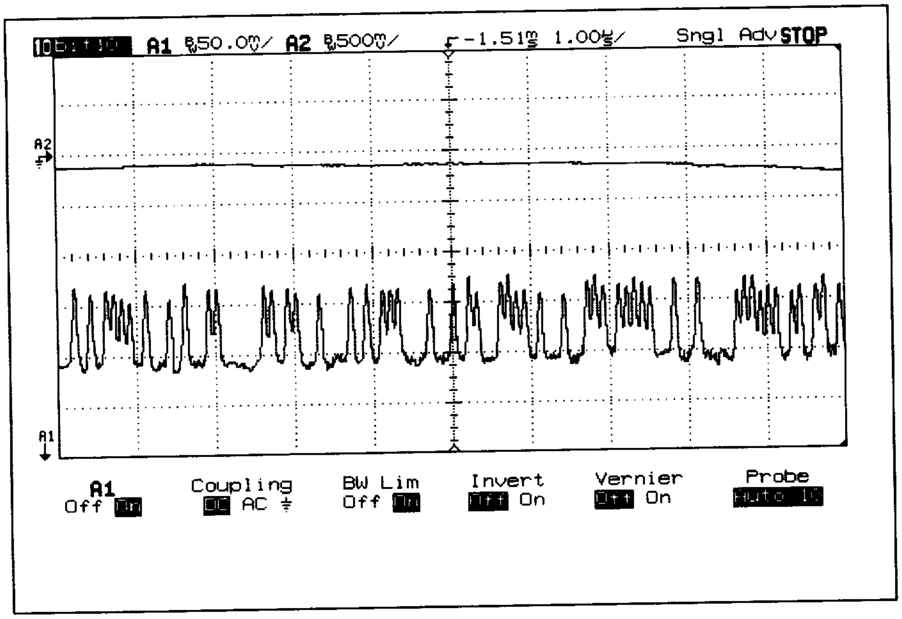


Figure 18b – optical resolution measurement 10 MHz, 10 m/s

Signal to noise ratio (SNR) of the data is shown in Figure 19. A single frequency data pattern at 10 MHz was written on the tape at 10 m/s. Subsequently that data was read back and fed to a signal analyzer. The 10 MHz data peak stands 40 dBs above its base line.

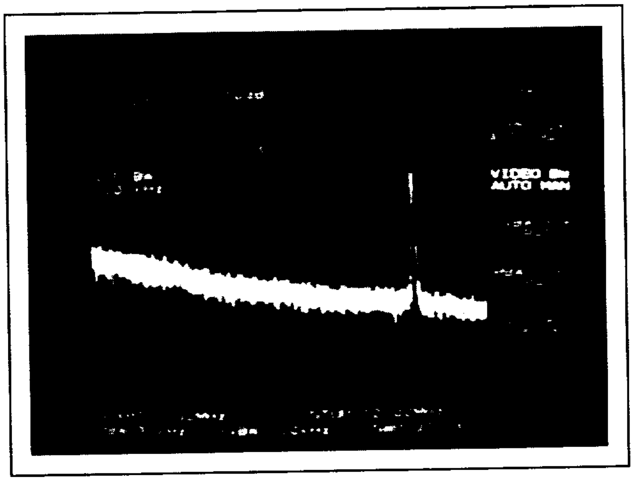


Figure 19 – SNR Measurement of 10 MHz Data

Data track-to-track crosstalk was also both simulated and measured. Simulation result showed that under the worst case situation, the T/T crosstalk is better than 40 dBs down. Preliminary experimental data based on 0.8 um track spacing show similar results.

Figure 20 shows two adjacent data channels one of which is reading data and the other one is on blank tape region. No measurable crosstalk amplitude is noticed. To show that these two channels are indeed adjacent, a region of the tape with a common surface defect with a similar amplitude effect on both channels was chosen. In this way, the defect appears in both adjacent channels and is almost equal in amplitude. However, the data marks do not show up in the adjacent channel indicating lack any noticeable track to track crosstalk.

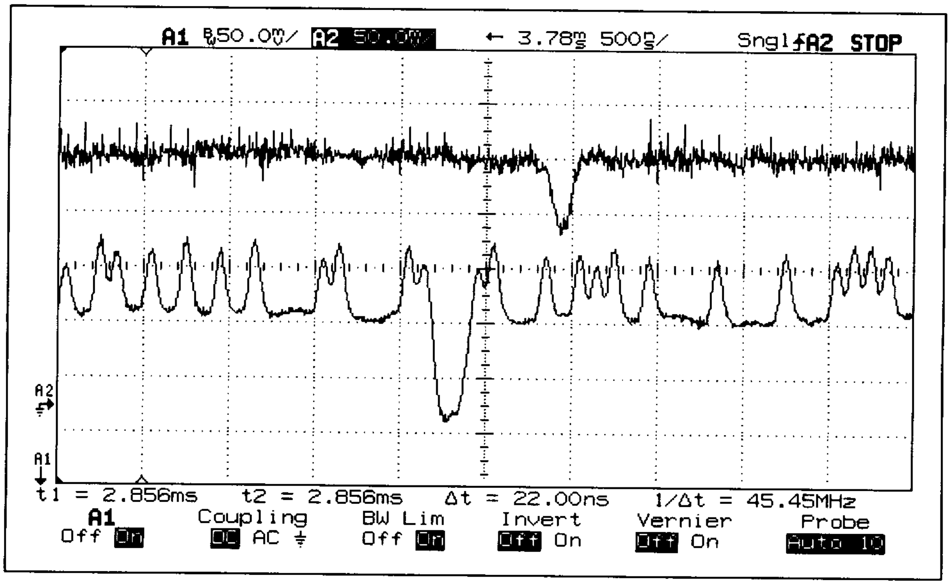


Figure 20 – Track-to-Track Data Crosstalk

Related Work

Other attempts have been made in using a laser diode array to generate the multiple beams to read and write. However, issues of thermal crosstalk, wavelength variation, and beam parameters differences among the individual diodes within the array make the design extremely expensive to manufacture especially when a large number of channels is required for a fast data transfer rate.

Future Work

As media with improved surface quality becomes available, more experiments will be done to gain a better understanding about effects of tape shuttling on data SNR and BER. Furthermore, the capability to record and read back with an increased number of beams will be tested.

Conclusions

In conclusion, we have shown an optical tape drive design capable of storing a nominal 1 TB of user capacity in a 3480-style cartridge with a user data transfer rate of 25 MB/s. We further showed the novel idea of using a single beam and a hologram combination to generate a multiplicity of recording channels. This approach significantly enhances reliability of the tape drive in contrast to using laser diode arrays. This design is extensible to higher data transfer rates with a minimum hardware modification by increasing number of channels. The optical tape drive design approach is very realistic and viable since not only is there no mechanical contact of the tape with either the transport or the optical head, but also the optical tape itself has a lifetime of longer than 30 years. The recorded information will not be damaged by environmental temperature changes or by a strong magnetic field.

References

[1] Tyan et-al. *Kodak Phase Change Media for Optical Tape Applications.* (NASA Conference Publication 3198, Vol. II, September 1992)

A Portable Tape Driver Architecture

Curtis Anderson
TurboLinux
2000 Sierra Point Parkway, 4th Floor
Brisbane CA 94005
canderson@turbolinux.com
tel +1 650 244-7777
fax +1 650 244-7766

Abstract

This paper describes a new architecture for device drivers for tape drives attached to UNIX-like systems. The design goals are presented, some current architectures are measured against the requirements, and a new architecture is described along with its advantages and disadvantages. This architecture was developed, and this paper was written, while the author was at Silicon Graphics. The work continues at SGI for IRIX, SGI's variant of UNIX, and for Linux. The resulting Linux software will be licensed as Open Source

1 Introduction

In order to evaluate the architecture presented later, and the design decisions that went into it, the environment and assumptions that existed during the design must be presented as well.

1.1 What Do Tape Drivers Do?

The fundamental purpose of a tape driver is fairly simple; it provides access to the tape drive hardware for use by application programs. Beyond that basic purpose, however, there are secondary characteristics that are controlled by the environment that the driver is running in and by application program expectations.

For example, tape drivers for a UNIX-like environment will attempt to both insulate the application from some of the device specific details of controlling the drive hardware and to homogenize the operational interfaces for different types of drives. One example of the latter is the need to have all drive types end up on the same side of a file mark when reading through a tape, in spite of what the drive's firmware does naturally. A critical result of this is the application having the ability to predict where the tape is going to physically end up after each operation, i.e.: before the tape mark or after it.

Other secondary characteristics include the failure modes of the driver and the failure domain. The failure modes describe how the driver can fail; e.g.: hang, crash, returns an error, silent failure, etc. The failure domain describes how much impact the failure modes have; e.g.: does the current operation fail, the application hang, or the system crash on each type of failure. Some combinations are clearly more desirable than others.

1.2 Traditional Tape Driver Architectures

The normal architecture for a tape driver in a UNIX-like system is an event driven state machine built into the operating system kernel. For most UNIX-like systems building it into the kernel is required in order for the driver to have access to the hardware control registers for the connection to the drive. That connection will usually be a SCSI bus but FibreChannel attached tape drives are now starting to appear on the market.

The driver must be able to support multiple devices simultaneously. As a result of the interrupt-driven nature of modern I/O busses, it must also be able to coordinate an asynchronous thread of execution with a synchronous one for all such devices simultaneously. In a multiprocessor system it must even be able to protect itself against race conditions between the processors, in both normal code and interrupt level code. The interrupt level code can have impacts on the rest of the kernel's ability to respond to real-time events, and the kernel's scheduler can have an impact on the driver's ability to keep the drive streaming.

The kernel of a UNIX-like system is usually a fairly hostile place to program. The interfaces to supporting code in the rest of the kernel are complex and the dependencies are delicate, and those interfaces and dependencies can change at every release of the kernel. In addition, the debugging tools are usually very primitive.

The driver may also attempt to support in a single monolithic driver source file many, if not all, of the devices the system vendor wants to be able to control. This may lead to either table-driven code or inscrutably complex run-time checking of the drive type. In either case there is the risk that any change to the code requires careful attention and regression testing to ensure no breakage in support for any of the supported devices.

The opposite structure, a separate driver source file for each tape drive make/model to be supported, is used by some system vendors. This option avoids some of the pitfalls of the monolithic driver, but it leads to a risk of different semantics from one tape make/model to the next, and still has a requirement to modify and retest all the supported drivers if the interface to a kernel support routine changes.

The sum of all these forces acting on the design of tape drivers has brought us to a point where a tape driver is a complex and delicate piece of code where any bug includes the risk of impacting the entire computer system.

2 Architectural Requirements

In order to decide what, if anything, is wrong with a traditionally structured tape driver, we need a list of those characteristics that we believe are important to the architecture of a tape driver. Debating which characteristics belong in this list, which do not, and their order of importance is in itself an interesting topic. Here is the list of requirements used for this work, shown in their order of importance:

2.1 Failures Must Be Contained

Failures in a tape driver must be isolated to that one drive, i.e.: it cannot be allowed to crash the system. Limited forms of service interruption are acceptable; for example a bug might affect the use of one device.

2.2 All Drivers Must Provide The Same Operational Semantics

Any tape driver must provide application programs with the same operational semantics as all other drivers. For example, they must all end up on the same side of the file mark when reading a tape.

An application should be able to reliably predict the behavior of the tape drive no matter who wrote the driver for it. An application should also be able to reliably predict the behavior of the tape drive, with some caveats, no matter which drive type it is. The former ensures that all DLT7000's operate the same, while the latter ensures that (to the extent that it is physically possible) a simplistic application doesn't need to know if it running against an AIT-2 or a DLT7000.

2.3 Drivers Must Be Portable To Multiple Operating System Platforms

Portability of an application that uses tapes to a new operating system can be greatly hindered by a difference in tape access semantics. The best way to avoid such differences is to use the same driver on all of them. Any new driver for a given make/model of tape drive must be source code portable to multiple kernels.

2.4 Distributed Development Of Drivers

The model used in the PC marketplace of bundling the driver software with the drive hardware is a good one. The people who make the drive are the ones in the best position to be able to make that drive perform correctly and reliably. This implies that it must be possible for the drive vendor to be able to build a driver for an operating system without reference to proprietary information from the platform vendor. It must be possible for multiple organizations to develop drivers for different devices in parallel.

2.5 High Performance

Any new tape driver architecture cannot sacrifice the performance of the drive or impose a significantly higher CPU load than a traditional architecture.

2.6 Isolating Support Of A Device From Other Devices

A monolithic tape driver implementing all supported tape drives will become unwieldy as the number of tape drive make/models being supported grows. Regression testing each driver change against all supported devices quickly becomes the dominating factor in the cost of adding support for a new drive. Using separate driver source files for each supported make/model of tape drive avoids this pitfall.

2.7 Differing Levels Of Investment For Each Drive Type

It must be possible in a tape driver architecture for one driver implementer to provide the basic set of operations and error recovery mechanisms while another provides that plus additional error recovery and/or additional device dependent operations. Any new architecture cannot raise the minimum requirements for implementing a tape driver too high, nor can it disallow extensions to take advantage of drive-specific features. Note that a driver providing additional operations risks requirement 2.2 unless it is a pure superset.

3 Evaluating The Traditional Architectures Against The Goals

Before going to the trouble of designing a new tape driver architecture, we need to decide if the above mentioned "traditional" approaches to driver structure are lacking significantly enough to warrant the effort.

The two approaches we've been discussing are both fully inside the kernel so they both will impact the entire computing system if they encounter a severe bug. There is no inherent difference between these two models in terms of their portability across operating systems (they are not) or in terms of their performance.

To successfully create a high performance, highly reliable driver in the hostile environment of an operating system kernel, the implementer must have a great deal of detailed knowledge about the particular kernel they are targeting and be able to use some fairly primitive debugging tools. Both of those requirements imply that writing a driver requires a talented operating systems engineer. It is desirable to eliminate those requirements in favor of allowing an engineer with less specialized experience perform the task.

3.1 Monolithic Drivers

A monolithic driver implementing all supported tape drives easily provides common operational semantics across all drive make/models. It will become unwieldy as the number of tape drive make/models being supported grows, however. Regression testing each driver change against all supported devices quickly becomes the dominating factor in the cost of adding support for a new drive. Changes to the kernel support interfaces that the driver uses also necessitate a full regression test against all supported devices.

A monolithic driver has great difficulty isolating one drive type from another, and is in practice only modifiable by one person at a time, in a serial fashion of implementation then testing. It is also difficult to provide extended error recovery for one drive type while isolating that recovery code from the other drive types.

The monolithic model has significant problems meeting the requirements set out above.

3.2 Separate Driver Source Files

Using separate driver source files for each make/model of drive has a significant risk of allowing variances in the semantics provided by one drive versus another. There is no structural help in the architecture or development model to ensure this, it is just a matter of all the programmers knowing that they have to do the same thing.

Using separate source files for each make/model of drive is quite good at isolating one drive type from another and it lends itself quite well to being worked on by more than one person at a time. Adding extended error recovery code to the driver for one particular type of drive is straightforward and has no impact on the other drive types.

It has the disadvantage of locking the system vendor into providing a static set of support interfaces in their kernel for the tape drivers to use. Those interfaces can become inefficient and/or difficult to implement as the system vendor makes changes in the structure of the operating system kernel.

Overall, the separate driver source files model is better than the monolithic driver model, but it still has the critical problem of failure containment as well as some significant application portability risks.

4 New Architecture

The architecture being proposed is composed of a document, two main software components, and a well-defined interface between those components:

- *Tape Access Semantics Document*: specifies the behavior that an application can expect from the tape driver, e.g. which side of the file mark the tape ends up on after a read operation.

- *Tape Support Driver (TSD)*: a piece of code that lives inside the operating system kernel and is uniquely optimized for that kernel but is common to all tape drives supported by that operating system.

- *Personality Daemon*: a piece of user level code that is uniquely optimized for a given make/model of tape drive but is common to all operating systems that support that drive.

- *Personality Interface*: the interface between Tape Support Drivers and Personality Daemons. This is the piece that allows portability.

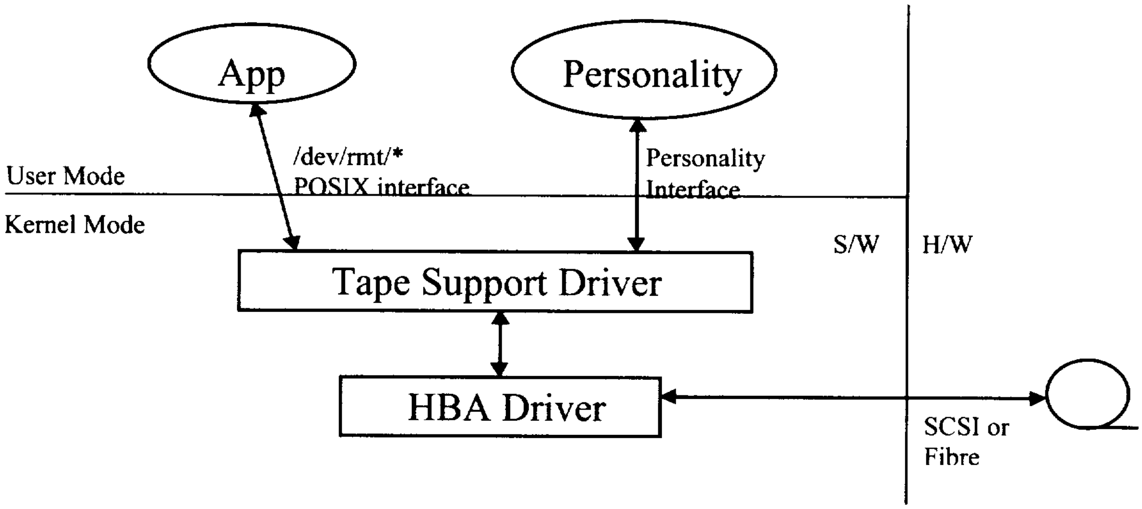


Figure 1. Structure of the new tape driver architecture.

This new architecture will be implemented on IRIX, SGI's variant of UNIX, and on Linux. The source code for the Linux port will be distributed under Open Source licensing terms. It is one of the key characteristics of this architecture that a site be able to modify or fix an existing Personality Daemon to meet their needs without the involvement of the operating systems vendor.

4.1 Tape Access Semantics Document

One of a tape driver's most important characteristics is the application's ability to predict what the tape drive is physically going to do for each operation the application asks the driver to perform. For example, which side of a tape mark will the drive end up on after a read command runs into a tape mark.

Given the requirement for independently written Personality Daemons, there must be a document that accurately and completely describes the semantics that can be relied upon by application developers and that therefore must be provided by Personality Daemon developers.

In practice, a conformance test suite must also be written. It will need to exercise all of the operations defined in the semantics document in order to verify that a given Personality Daemon correctly implements those operations. The test must check the handling of tape marks, end-of-data, end-of-tape, beginning-of-tape, file-space-forward, short reads, long reads, etc.

The semantics that the document describes, and that the Personality Daemons implement, is not part of this paper. This new architecture is independent of the particulars of the tape access semantics being implemented. In fact, in the Future Work section of this document we talk about the possibility of there being different documents describing different sets of semantics, each matching a de facto industry norm. Examples include Solaris, IRIX, AIX, and HP/UX. A site could have several Personality Daemons available for each drive, one for each common set of tape access semantics. The use and management of multiple Personality Daemons per drive is, again, not part of this paper.

4.2 Tape Support Driver

In order to gain access to the SCSI or FibreChannel controller, and as a practical requirement of getting high performance, we have defined a component inside the kernel. We call that piece the *Tape Support Driver (TSD)*. It is unique to each operating system kernel but is common to all drives supported by that kernel.

The TSD is basically just a data pump. It is highly optimized kernel code that has all the usual dependencies, interrupt level code, multiprocessor locking, etc., and is probably written by a senior operating systems engineer. The implementation will be unique to each operating system and will take advantage of all the optimizations that are available in that environment. The TSD supports the read() and write() system calls from the application as well as the newly defined *Personality Interface*. The TSD does not do anything other than read/write and the Personality Interface, it depends on a Personality Daemon for support of all other operations and for error handling.

There is an exception to the policy that the TSD does no handling of errors. A read() operation where the data transferred is less than the data in the tape block will result in an

Illegal Length Indicator error being reported by the drive. This would appear to be an error to the TSD and would normally be bounced up to the Personality Daemon for processing. This is not considered an error under some circumstances, and we cannot afford to involve the Personality Daemon on every read() operation if we want to maintain streaming, so the architecture allows the TSD to have some parameters and for the Personality Daemon to control them. The number of parameters must be kept to a minimum in order to keep the TSD simple, but they will undoubtedly have to exist.

4.2.1 External Interfaces To The Tape Support Driver

The TSD must support two external interfaces. The first interface is the POSIX compliant tape-access interface that an application uses to control the tape; e.g.: /dev/rmt/tps0d4nrv. The second is the Personality Interface to a Personality Daemon.

Some operating systems layer disk and tape drivers on top of drivers for the specific Host Bus Adapter (HBA) cards, and some provide more formal interfaces to kernel support routines that a driver can make use of. Both of those are examples of interfaces that are not visible in this architecture, they are implementation dependent.

The Personality Interface for a drive is only available to a Personality Daemon, and only one Personality Daemon can be associated with a given physical drive at a time.

4.2.2 Error Injection In The Tape Support Driver

Murphy assures us that the error recovery code in a Personality Daemon will not work correctly unless it has been tested. Making a real drive fail in exactly the required way at the required time in order to test that code is difficult at best. Therefore, it is desirable to be able to inject errors into the operation stream from software. Software based error injection may not be quite the same as if a physical drive were actually failing, but the increased flexibility of testing and code coverage would more than make up for the difference in the nuances.

The error injection is probably best done by a thin layer below the TSD itself. It could watch the sequence of operations as the TSD and Personality Daemon interact with the drive. When the desired position, time sequence, or pattern was found, the layer would return an error rather than the real answer. The patterns, sequences, and resulting error codes should be programmable by a user-level utility program; the level of programmability of the layer would undoubtedly be increased over time.

This layer will be needed in the long run, but is not required for initial support of the new tape driver architecture.

4.3 Personality Daemons

The second major component of the new architecture is called a *Personality Daemon*. It is unique to a given make/model of tape drive but is common to all operating systems.

The Personality Daemon is a piece of user level code that makes use of the Personality Interface to talk to the Tape Support Driver in the kernel. It provides operational control and semantics for that tape drive. We take it as a base assumption that performance of the drive is less important when recovering from an error or processing a special request from the application (for example, writing a file mark), than when it is doing reads or writes.

There will be a unique implementation of a Personality Daemon for each make and model of tape drive. Making a change to one Personality Daemon, or writing a new one, cannot introduce bugs into another Personality or change its semantics. This allows multiple people or organizations to develop Personality Daemons simultaneously. It also eliminates the need to regression test a Personality until it actually changes.

Having separate Personality Daemons for each drive type means that writing a Personality Daemon for a drive that is fairly self-sufficient is easier that writing one for a drive that requires more hand-holding. The flip side of that coin is also true, that it is possible to invest in a Personality Daemon for one drive more than for another, gaining better error logging or recovery or other operational advantages beyond simply pushing data to and from the drive.

The biggest drawback to having separate Personality Daemons is that it is more difficult to ensure that they are all supporting the same semantics (e.g.: does the tape end up before or after the file mark) for the application program. A conformance test suite will be required in order to test Personality Daemons. Fortunately, the set of operations and their expected outcomes has already been clearly defined by the *Tape Access Semantics Document* and as part of the definition of the *Personality Interface*.

4.3.1 Personality Daemon Failure Containment

A single Personality Daemon is the unit of failure in this architecture. Since only the most primitive of device support is inside the operating system kernel, and the complexity and potential for bugs comes mostly from error processing and recovery code, most failures should be contained to a single user-level process. That process can be restarted if it fails or hangs, thereby regaining correct operational control of the drive without impacting the system as a whole. Only the application that was accessing the drive would be impacted.

4.3.2 Personality Daemon Responsibilities

A Personality Daemon is responsible for processing both control commands from the application and for handling errors generated by the drive.

When an application issues any control operation on the drive like a rewind or a seek-to-end-of-data, any operation other than a read or a write in fact, the TSD will simply forward that request to the Personality Daemon. The daemon is then responsible for building the correct SCSI command block(s) for the particular drive it is controlling and using the Personality Interface to send those commands to the drive. In this way it interacts with the drive to accomplish the operation that the application wanted. This sounds very indirect, but this mechanism allows for Personality Daemons to work around variations in the native behavior of one drive type versus another in order to accomplish the goal. In short, this is how we can homogenize the native behavior of drives into the standard semantic model that applications want to depend on. When the daemon has finished its processing of the control operation, it tells the TSD to resume the application with a successful return code.

The Tape Support Driver will also pass virtually any error condition reported by the drive up to the Personality Daemon for processing. This allows the daemon to do device-dependent error diagnosis and recovery operations. The daemon will again use the SCSI pass-through capability of the Personality Interface to interact with the drive before returning control to the application.

4.3.3 Personality Daemon Programming Model

There will be one running instance of a Personality Daemon for each physical drive attached to a system. This allows the programming model inside a Personality Daemon to be single-threaded and fully synchronous. The Personality Daemon will wait on interactions with the Tape Support Driver through the only interface it has to support, the Personality Interface, which is a straightforward event-response style interface. The Personality Daemon does not need to worry about coordinating interrupt level code with non-interrupt level code, multiprocessor locking issues, or asynchronous operations. The reduction in code complexity is one of the significant advantages of the new architecture.

The fact that there is one Personality Daemon per physical drive also implies that the failure scenarios only involve one drive at a time and do not impact the rest of the system. Barring a bug in the Tape Support Driver or a case where the TSD does not adequately protect itself from bad input from the Personality Daemon, the rest of the system will not be impacted if a Personality Daemon crashes or hangs. In fact, the Personality Daemon can simply be killed and restarted to recover from a hang.

The single-threaded, synchronous, user level process programming model allows the use of powerful debuggers and a much more elaborate testing environment. Adding this to the reduced complexity of the code should result in a much higher level of reliability for the driver overall.

4.4 Personality Interface

The *Personality Interface* is common to all Tape Support Drivers and Personality Daemons and defines the relationship between them.

The Personality Interface is used by the Tape Support Driver to tell the Personality Daemon about actions requested by the application, for example: rewind, file-space-forward, and write-a-file-mark. It is also used by the TSD to tell the Personality Daemon about exceptions generated by the tape drive, for example: file mark found, early warning for end-of-tape, read failure, etc. The TSD expects the Personality Daemon to handle those conditions as it sees fit.

The Personality Interface is used by the Personality Daemon to receive notification from the TSD of either application-requested actions or tape drive generated errors. It is also used to directly interact with the tape drive via a mechanism to send/receive arbitrary SCSI commands, commonly called a pass-through driver, and to control the error indications going back to the application.

The Personality Interface is synchronous in both directions. When the application requests a rewind, for example, the TSD will store the details of who is doing what and will use the Personality Interface to wake up the Personality Daemon. Note that the application will be blocked in the TSD at this point. The Personality Daemon will use the Personality Interface to query the TSD for status. It will then generate the SCSI "rewind media" command and will call the Personality Interface in order to send it down to the TSD for execution. When that has finished and returned to the Personality Daemon, it will call back into the TSD with a command telling the TSD to resume the application with a specific return code.

As of the writing of this paper in December of 1999, SGI is now in the implementation phase of the project. The particulars of the Personality Interface will not be finalized until a few Personality Daemons have been written and the Personality Interface's generality has been verified, but listed below is the structure of the interface as it is currently defined.

4.4.1 Interface Initialization

When a Personality Daemon first starts up it needs to set basic parameters for use by the TSD. It must also ensure that the TSD and itself are both using the same version of the Personality Interface and that the drive to be controlled is of the correct type for the Personality Daemon. The *TSD_INIT* ioctl() is used to initialize the interface.

Some of the parameters that are configured when the interface is initialized include: basic device timeouts, whether the drive supports reads followed by writes without an intervening tape positioning command (or writes followed by reads), whether some device-reported error conditions can be ignored by the TSD (e.g.: short length reads), and the list of ioctl() operations that the Personality Daemon supports.

4.4.2 Sleeping And The Use Of Signals

The Personality Daemon will sleep during the time that it is not actively servicing the TSD, waiting for a signal to arrive. The TSD will send the Personality Daemon a *SIGUSR1* signal when it needs help with something.

Once the Personality Daemon has been broken out of its sleep, it will use the *TSD_QUERY* ioctl() to determine the basic situation and will then use the ioctl() operations defined in the following sections to interact with the drive and the TSD.

4.4.3 Type Of Service Required

The TSD_QUERY ioctl() returns the reason for the latest signal from the TSD and the details behind that signal. The possible reasons include an application doing an open(), close(), or ioctl() system call, an error reported by the drive, and a read or write operation on the drive. The structure returned by the TSD_QUERY ioctl() includes all of the fields required to give the details appropriate to all of the reasons for the Personality Daemon being involved. Combining the various fields into one structure was seen as better than defining a set of query operations, one for each type of service that the TSD requires of the Personality Daemon.

For all operations, the structure includes the process ID of the application. For open()'s, the structure also shows the flags associated with the open() call (rewind-on-close, density, compression, etc). For close()'s, no additional fields are required. For ioctl()'s, the structure contains the ioctl() command code and the ioctl() argument if it is not a pointer to a data structure (see the *TSD_COPYIN* request). For errors reported by the drive, the structure contains the SCSI sense code information, the requested I/O size, and the residual un-transferred byte count. For I/O operations, the structure includes the type of operation and the transfer counts. The structure also contains some statistical counts such as total bytes read/written, total read/write operations performed, current block number on tape, etc.

4.4.4 Application Request Processing

The application will be suspended and the Personality Daemon woken up on all open(), close(), and ioctl() operations and on some read() or write() operations.

The Personality Daemon needs to be involved every time an application opens a tape drive so that it can ensure the drive is ready for the application, validate access modes, etc. The Personality Daemon needs to be involved on every close operation as well so that it can clear and/or set status flags appropriately and to issue the rewind operation for the rewind-on-close semantics that some applications depend on. All control operations that the application performs (e.g.: rewind, write file-mark, etc) will come into the kernel via the ioctl() system call.

When an application issues an ioctl() operation on a tape, it provides the operating system kernel with an operation code and a pointer to a memory buffer. The size and contents of that memory buffer are operation dependent. We do not want to provide direct access from the Personality Daemon into the application's address space, so the contents of the memory buffer must be copied into a buffer inside the TSD. The TSD can then make the contents of that kernel buffer available to the Personality Daemon. The Personality Daemon needs to tell the TSD at initialization time which operation codes it supports, the associated amount of data to be copied in to or out of the kernel for that operation, and whether super-user privileges are requires to perform that operation..

The Personality Daemon will make use of two ioctl() operations when processing ioctl() requests from the application: *TSD_COPYIN* and *TSD_COPYOUT.* TSD_COPYIN returns from the TSD the ioctl() operation code and associated data that the application passed to the TSD. TSD_COPYOUT sends to the TSD the bytes to be copied out to the application as the results of the application's ioctl() operation.

The return code for the ioctl() call the application made comes via a separate ioctl() operation, *TSD_RESUME*, that the Personality Daemon uses. It implies that the TSD should unblock the application and allow it to continue processing.

4.4.5 Support For Device-Dependent Ioctl() Operations

We do not want to artificially limit the ioctl() operations that a Personality Daemon can support. There are many drives that provide unique features that an application might want to use if it was willing to include drive-type-dependent code. The additional operation codes will simply be listed as part of the table of supported ioctl() codes and buffer sizes that is passed into the TSD by the TSD_INIT ioctl().

4.4.6 Drive Error Processing

The TSD_QUERY ioctl() returns to the Personality Daemon essentially all of the information that is needed to start processing the error from the drive. The Personality Daemon will do an initial diagnosis of the problem based on the SCSI sense code information that came back from the last TSD interaction with the drive. It may use the SCSI pass-through support described below to interact with the drive, doing additional error analysis and/or error recovery operations.

When the Personality Daemon has finished all of the processing that it wants to do for the reported error, in addition to its ability to return an error to the application, an option to the TSD_RESUME ioctl() allows the Personality Daemon to ask the TSD to retry the original operation.

4.4.7 I/O Notification Processing

The Personality Daemon needs the ability to tell the TSD to involve it just prior to, or just after, the next read or write SCSI command is issued to the drive.

For example, if the application has opened the drive in fixed-block mode and not set the block size to be used, the tape driver should use the block size that the tape was written with for all subsequent read operations. In order to determine what block size to use, the Personality Daemon will need to get involved just prior to sending the first SCSI read command to the drive.

Some tape drives will not report that the cartridge has physically been marked read-only until some time after the first write operation when the data in the on-drive buffer is actually flushed to the tape. The Personality Daemon for such a drive will arrange to get involved after the first write to a cartridge completes. It can issue a command to force the on-drive buffer to tape, thereby checking the read-only status of the cartridge at a point where the Personality Daemon can still return an error code for application's first write operation indicating that the cartridge is in fact read-only.

This capability can also be used by the Personality Daemon to govern all access to the tape by the application if necessary. After some types of serious I/O errors, further reads or writes to the drive must be disallowed. The Personality Daemon can intercept all I/O operations before they happen and return an error to the application.

The TSD_RESUME ioctl() will include flags that tell the TSD to involve the Personality Daemon just prior to, or just after, the next read or write SCSI command.

4.4.8 SCSI Passthrough Support

When the Personality Daemon wants to send a SCSI command to the drive, it needs to provide to the TSD the following:

- The bytes comprising SCSI command to be sent.
- A pointer to a data buffer used for output to the drive or for input from the drive.
- Flags, including whether the drive will expect to transfer data to or from the host.
- The maximum number of seconds to wait for the command to complete.
- A pointer to a buffer for the SCSI sense code information if the command fails.

If the command was successful, the Personality Daemon can expect that the TSD has filled the data buffer with the results from the drive and has returned the number of valid bytes in that buffer. If the command was not successful, the Personality Daemon can expect that the TSD has filled in the status reported by the HBA, the status reported by the drive (if any), and the SCSI sense code information (if any). The HBA status will include errors such as "parity error on the bus" which will render the other status information invalid.

The *TSD_SEND* ioctl() is the operation used by the Personality Daemon to send SCSI commands to the drive. With one exception, that ioctl() will not return to the Personality Daemon until the SCSI command has either successfully completed, the command has failed and error status has been obtained, or the command has timed out.

In order to support operations such as the ability of an application to continue processing while a tape rewind is in progress, the flags field of the TSD_SEND ioctl() will tell the TSD whether to wait for the SCSI command to finish or to return to the Personality Daemon immediately. The Personality Daemon can then use the TSD_RESUME ioctl() to allow the application to continue processing.

If an application is resumed while a long-running operation is in progress, the Personality Daemon is responsible for managing the application's access to the drive. For example, it can use the flags on the TSD_RESUME ioctl() to intercept all I/O operations from the application before they are sent to the drive, then use the TSD_SEND ioctl() to verify that the drive has finished the long-running operation before retrying the application's I/O operation. An alternative approach relies upon the fact that while the drive is busy rewinding, it will report a "busy" status. All I/O operations that the application issues will generate an "error" that will then involve the Personality Daemon. In either case of the Personality Daemon gaining control, it should probably just sleep for a while and then retry the operation.

4.4.9 Block Size Control

The Personality Daemon needs to be able to control the size of the read and write operations that the TSD is passing from the application to the drive. The *TSD_BLOCKSIZE* ioctl() tells the TSD the minimum and maximum block sizes that the drive can accept, and if the drive is operating in fixed-block mode, the current block size to use. This information may change during an application's use of a drive as a result of the application asking to change the block size being used for fixed-block mode access.

4.4.10 Stopping In-Progress Operations

The Personality Daemon needs the ability to abort a long-running operation that might be in progress on the drive. The *TSD_ABORT* ioctl() asks the TSD to do just that. Under certain circumstances, the Personality Daemon needs to be able to get control of the drive again after issuing long-running operations such as a rewind.

4.4.11 Personality Interface Summary

SIGUSR1 – Signal when the TSD needs help from the Personality Daemon
TSD_INIT – Initialize the Personality Interface
TSD_QUERY – Show the reason the Personality Daemon needs to get involved
TSD_COPYIN – Copy the application's ioctl() info into the Personality Daemon
TSD_COPYOUT – Copy the Personality Daemon's response to the application's ioctl()
TSD_SEND – Pass a SCSI command through the TSD to the drive
TSD_RESUME – Resume the application or retry the operation that failed
TSD_BLOCKSIZE – Set the allowable block sizes for read() and write() calls
TSD_ABORT – Ask the TSD to abort any in-progress operations with the drive

4.5 Example: Processing A Drive Exception

A critical design issue in the new architecture is how to handle errors reported by the drive. The tools that the Personality Daemon has available to it to handle drive errors have already been described, but walking through the sequence of events in a representative example would be illustrative.

Assume that the application is reading data from the drive and the drive runs into a media defect that has obliterated some of the data.

1. For initial state we assume that the Personality Daemon is sleeping waiting for a Personality Interface signal that the Tape Support Daemon (TSD) needs help. We also assume that the application has issued a read() system call and is blocked waiting for the results.

2. The TSD gets the read request from the application and issues a "read" SCSI command to the drive. The drive encounters a problem and responds to the host with a "check condition" (a SCSI message indicating a problem with the command).

3. The TSD then uses a "request sense" SCSI command to get more information from the drive on what type of error happened and uses the Personality Interface to signal the Personality Daemon that something needs attention.

4. The Personality Daemon uses the Personality Interface to get the sense code information that the drive returned as well as the current status of the application and any other context information it needs such as block counts, operation counts, etc.

5. Based on the sense code information, the Personality Daemon decides to run some diagnostics on the drive. It builds a SCSI command block for the command it wants to send to the drive and calls into the TSD. The Personality Daemon is then blocked waiting for the call to return from the TSD.

6. The TSD sends the SCSI command block to the drive and either collects any resulting output or uses a "request sense" command to collect any error (sense) codes. It allows the call from the Personality Daemon to return with whatever it has collected.

7. The Personality Daemon analyses the results from the TSD and decides that it should log the error, fail the operation, and return an error to the application. The error is logged via the normal SYSLOG facility from the Personality Daemon.

8. The Personality Daemon calls into the TSD asking for the application to be resumed with an "EIO" error code being the return value from the read() system call.

9. The call returns from the TSD into the Personality Daemon and it goes back to sleep waiting for the next signal from the TSD that something needs to be done.

Common variations on the above sequence would include the Personality Daemon issuing more diagnostic and/or error recovery SCSI commands to the drive, doing more detailed error logging or using different modes of notification (e.g. pager or email), interacting with any system management framework that might be desirable, and possibly retrying the operation that failed.

5 Future Work

It is possible to write a range of Personality Daemons for a given operating system providing different legacy-based semantics for the same drive. Since a Personality Daemon can be stopped and another one started for a given drive, it is possible for a Media Management System such as IEEE 1244 to offer different sets of semantics to an application and let the application choose at run-time which it wants to use.

6 Conclusions

Under this new architecture, the operating system vendor's job is to write a Tape Support Driver that can control their HBA, interact gracefully with the rest of their kernel, and implement the Personality Interface. Having done so, then they benefit from the available Personality Daemons. The drive vendor's job (or whoever provides a Personality Daemon) is to accurately control the drive and to conform to the Personality Interface. Having done so, then their drive will be supported on a wide range of operating systems.

Optical Recording and Recordable DVD Overview

Koichi Sadashige
Sadashige Associates
15 Amherst Rd, Voorhees NJ 08043-4901
Phone: +1-856-767-2644, FAX: +1-856-767-1462
e-mail: mm306@msn.com

Optical Recording

The expression *optical recording* is often used loosely by both engineers and marketing executives. Nearly all recording systems, where a focused laser beam is used in either writing or reading, are normally referred to as optical recording.

There are four basic types of optical recording, shown in Fig 1. All four are currently in use and although they are all applicable to both disc and tape, only disc products are available at the moment.

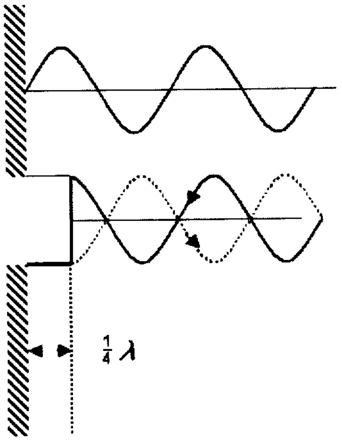
1. Change of physical dimension: Data recording changes the dimension, generally the thickness, of the media. The 1's are thicker than the 0's, or *vice versa.* Removing the material, or replicating a master recording by injection molding, which produces a medium with thickness variations representing the recorded data, can make the recording. CD-Audio, CD-ROM, DVD-Video and DVD-ROM media are volume produced through the injection molding process. The material removal recording, used since at least the beginning of recorded history, may become fashionable again in the future, using nanoscale technology.

2. Magnetic Recording: By raising the temperature of the medium to close to its Curie point, a weak magnetic field can be used to reverse the existing polarity of the bit cell on a track. The recorded pit size and shape are defined by the diameter and the on-time duration of the laser beam used as the heat source. This is the basic form of Magneto-Optical (MO) recording.

3. State Transition: Certain alloys of elements from the group VI of the periodic table can be caused to transition between amorphous and crystalline states by controlled heating and cooling. These alloys are stable in both states at room temperature. This technique is often referred to as Phase Change (PC) recording. On the medium, which is in the polycrystalline state, recording is accomplished by rapidly heating the material with a laser and letting it cool quickly to the amorphous state. As with MO recording, the PC process is completely reversible. PC recording is the basis for DVD-RAM, and the proposed newer format, "DVD-Video Recording".

4. Polymer Dye Burn-In: In this irreversible process, data is burned onto the medium whose surface contains a polymer dye. CD-R, and DVD-R belong to this recording category.

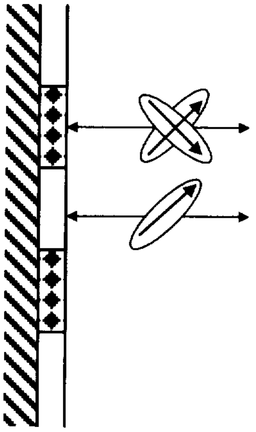
Optical Recording Characteristics



- Recording Type: Physical Height Change (Pit or Land)
- Recording Method: Injection molding
- Reading Method: Optical Path interference
- Products: CD, DVD, CD-ROM

Figure 1a. Optical Recording Characteristics: Land and Pit recording

Optical Recording Characteristics 2



- Recording Type: Magnetization Reversal
- Recording Method: Spot temperature elevation near Curie point
- Reading Method: Kerr Angle differentiation
- Products: MO (Magneto-optical)

Figure 1b. Optical Recording Characteristics: Magneto-optical recording

Optical Recording Characteristics 3 & 4

Recording Type	Recording Method	Reading Method	Products
3. State Change	3. Pit temperature elevation	3. Reflectivity differentiation	3. PC (phase change) DVD-RAM
4. Polymer Dye characteristic change	4. Pit temperature elevation	4. Reflectivity differentiation	4. CD-R DVD-R

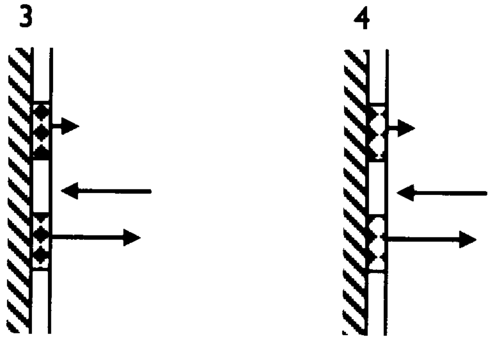


Figure 1c. Optical Recording Characteristics: Polymer Dye and Phase Change

Data Recovery

In technologies 1, 3 and 4, the reflectivity difference between recorded and unrecorded areas is detected and used to generate an electronic signal.

In CD-Audio, CD-ROM, DVD-video and DVD-ROM, which are replicated by injection molding, the effective elevation difference between recorded (pit) and unrecorded (land) areas is $\lambda/4$, leading to near cancellation of light reflected from the recording area; here λ is the laser wavelength.

In PC media, the polycrystalline state has a higher reflectivity than the amorphous state.

On polymer dye media, the burned-in spots have lower reflectivity than the unrecorded areas.

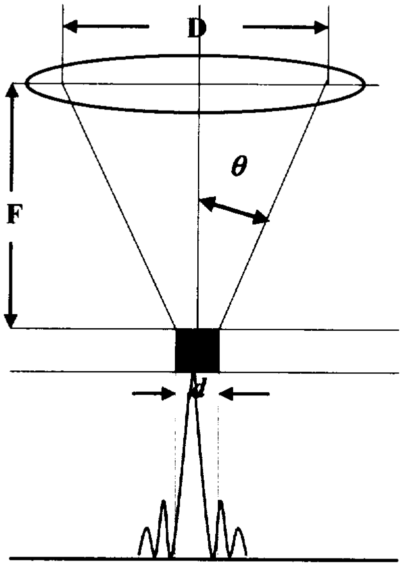
When reflected off a magnetized surface, the plane of polarization of a polarized beam is rotated with respect to the incoming beam, a phenomenon known as the *Kerr effect*, discovered in 1877 by John Kerr. In MO recording, this small rotation is used to generate the read signal.

Factors determining Optical Recording Density

The diameter of the focused laser beam is the determinant of areal density except in the case of injection-molded media. The diffraction-limited spot diameter is given by the expression

$$d = C\lambda/NA$$

Where NA is the numerical aperture of the lens and the proportionality constant depends on the factors illustrated in Fig 2.



f number = F/D

Numerical aperture, NA = *sin θ*

$$NA = \frac{D/2}{(F^2 + D^2/4)^{1/2}}$$

Spot size = *d*

$d = \sigma\lambda/NA$

Where λ = wavelength of laser

σ = factor determined by beam energy distribution

Figure 2. Factors determining spot diameter and their relationship

The numerical aperture is the sine of the solid angle as viewed from the focal point of the lens to the lens's maximum radius. The larger the NA, the faster the lens, as opposed to the commonly used lens speed expression of *f*-number. A smaller *f*-number indicates a faster lens. Theoretically NA cannot exceed unity, and the *f*-number cannot be smaller than 0.5.

For an optical recording drive to be practical -- physically small, reasonably priced, and requiring no external support equipment -- the laser must be a semiconductor type. Since the early 1980's, diode lasers operating in the near infrared wavelength, 900 to 800 μm -- have been available.

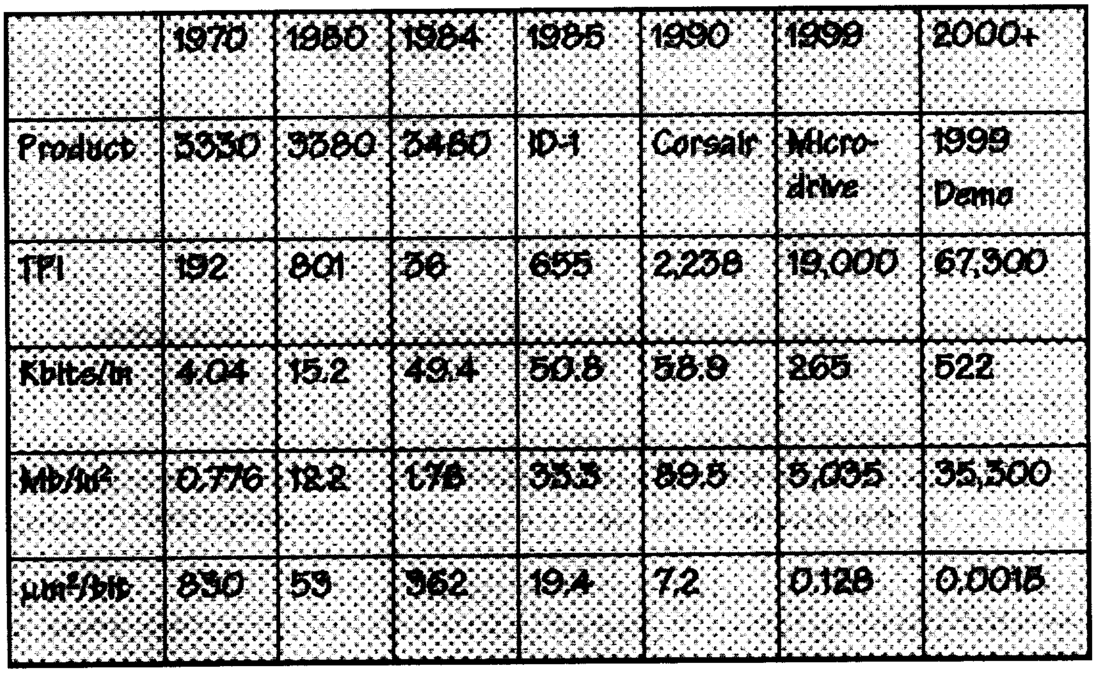
A single group lens with NA of 0.4 to 0.5 (corresponding to *f*-numbers of 1.2 to 0.9) can be manufactured inexpensively.

Based on these numbers, *i.e.*, the wavelength of 800 ~ 900 μm and NA of 0.4 to 0.5, the spot size comes out to be 1 to 0.8 μm.

The track pitch in optical recording is twice the spot diameter, and this almost completely eliminates inter-track interference. Further, by choosing the spot diameter as the minimum mark length, a density of 1.5 to 2 μm^2 can be achieved. This was believed to be achievable in the early 80's during the period leading up to the development of what came to be known as Compact Disc Technology.

Comparison with Magnetic Recording

If one studies the evolution of hard disk drives in the 70's and 80's, an interesting fact emerges. Fig. 3 shows the areal density of representative hard disc drives during this period. In the early 1980's, when optical recording was just moving from an engineering concept to the state of marketable product development, the HDD was operating at a density of 10 to 20 Mb/in^2, or 30 to 60 μm^2/bit. At that time, these figures were a factor of 20 to 30 smaller than what could be realized optically with available components.



	1970	1980	1984	1985	1990	1999	2000+
Product	3330	3380	3480	ID-1	Corsair	Micro-drive	1999 Demo
TPI	192	801	38	655	2,238	19,000	67,300
Kbits/in	4.04	15.2	49.4	50.8	58.9	265	522
Mb/in^2	0.776	12.2	1.76	33.3	89.5	5,035	35,300
µm^2/bit	830	53	362	19.4	7.2	0.128	0.0018

Figure 3. Growth of Magnetic Recording Areal Density

This explains why optical recording was, and still is, considered synonymous with high-density recording.

Tape recording was not much better than the disc recording. The first generation digital video recording format, D-1, entering the marketplace in 1981, had a 20 μm^2/bit. For purposes of comparison, the Fig 3 also shows the recording density of IBM-3480, the first cartridge tape data recording system, 362 μm^2/bit.

Details of the CD-Audio disc, as it finally entered volume production in early 1980's, are shown in Fig.4 CD-ROM has essentially the same characteristics, and it is produced by the same process, *viz.,* injection molding. All CD derivatives operate at the same recording density.

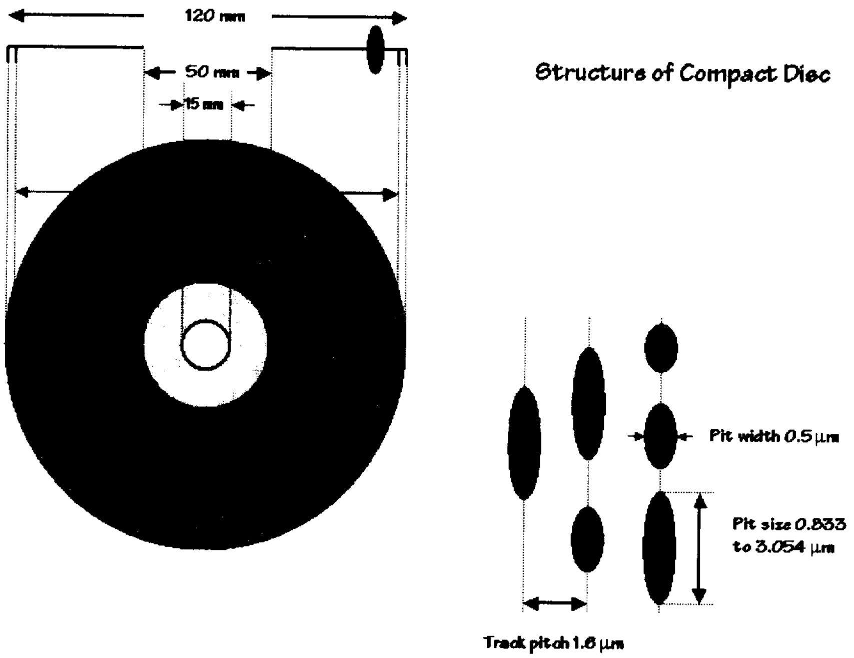


Figure 4. Structure of Compact Disc

The Road to DVD

The CD and its derivative products were the first volume-produced digital recording systems with an inexpensive storage medium. Its data holding capacity of 680 MB was a remarkable achievement when compared to the then standard data storage medium of IBM-3480 cartridge, with a capacity of 200 MB. The volume of the 3480 cartridge, 300 cm^3, is more than twice that of a CD in its protective plastic case.

New applications for both non-recordable mass replicated formats as well as recordable formats based on the CD technology emerged in 1980's and early 1990's.

The most notable one was the Video CD, which stores MPEG-1 compressed video of one-hour duration.

Unlike magnetic recording, where incremental performance improvements are practical and thus being introduced by manufacturers on frequent intervals, the CD has maintained its fundamental physical and optical characteristics intact for a considerable period of time.

During the intervening years, however, components, technology, and signal processing techniques, all directly applicable to optical recording, have made significant progress.

For the next generation Optical Disc Recording format, two technical consortia proposed remarkably similar systems. The two systems, however, contained minor, yet completely incompatible, technical details.

After fierce competition and much argument, the two groups agreed to join forces and develop a unified, single technical approach for the highly versatile, video, audio, and data compatible optical disc format of very high data holding capacity.

DVD, the digital versatile disc product line was born in December 1995.

DVD is a family of products, and specifications for new product configurations are constantly being developed and also being updated.

The first volume-produced product, the "DVD-Video", has approximately seven times the data capacity of the CD, or 4.7 GB per surface.

This capacity, 4.7 GB, has since become the yardstick against which the capability of all subsequent DVD products are measured. Recordable DVD formats, DVD-R, DVR-RW, and DVD-RAM, which did not meet this target initially, are now working toward the 4.7 GB per surface capacity objective.

DVD-Video incorporated following technical innovations to achieve the seven-fold capacity increase possible. Details of DVD-Video disc is shown in Fig. 5.

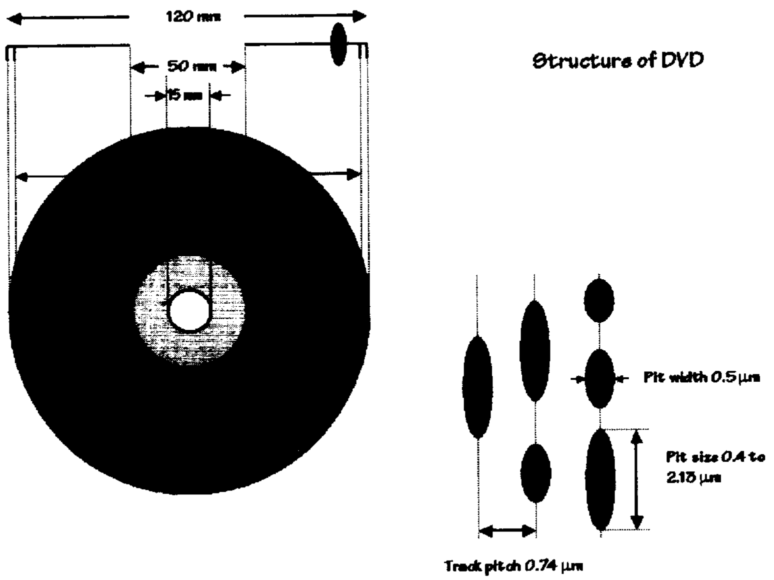


Figure 5. DVD-Video Disc

1. Laser wavelength shortened from 780 nm to 650/635 nm
2. Objective lens numerical aperture widened from 0.45 to 0.60

The effects of these two improvements were the reduction of track pitch from 1.6 to 0.74 μm, and shortening of the minimum mark length from 0.833 to 0.40 μm, or an areal density increase of 4.5.

3. The channel code was changed from 8 – 14 modulation (EFM) to 8 – 16 modulation, making a modest but important gain in the linear track density of 7%. (Even though the CD modulation code is EFM, three *merge* bits were used with each channel symbol, effectively making it an 8-17 code)

4. The Error Correction Code was improved. While it had become more powerful in correcting longer burst errors, its overhead was reduced from 30% to 15%.

In addition, DVD has incorporated a double layer data storage technology. Both recorded layers face the front side of the disc, making it possible to read the data from the same side, without flipping the disc. Since the signal from the bottom layer must be read through the front layer, the front recording reflective layer is semi-transparent.
Because of these added limitations imposed on the data-holding layers, the two-layer, front-readable DVD-Video has a combined capacity of 8.5 GB.

Phase Change Recording

Like the CD family of products, DVD also has a number of recordable formats. The first recordable DVD, DVD-R, is a higher density version of CD-R, whose basic characteristics have been discussed earlier.

Perhaps the most important DVD product for the data storage application is the DVD-RAM, which operates like a magnetic tape recorder, with some additional desirable features.

DVD-RAM is based on the Phase Change (PC) recording principle.

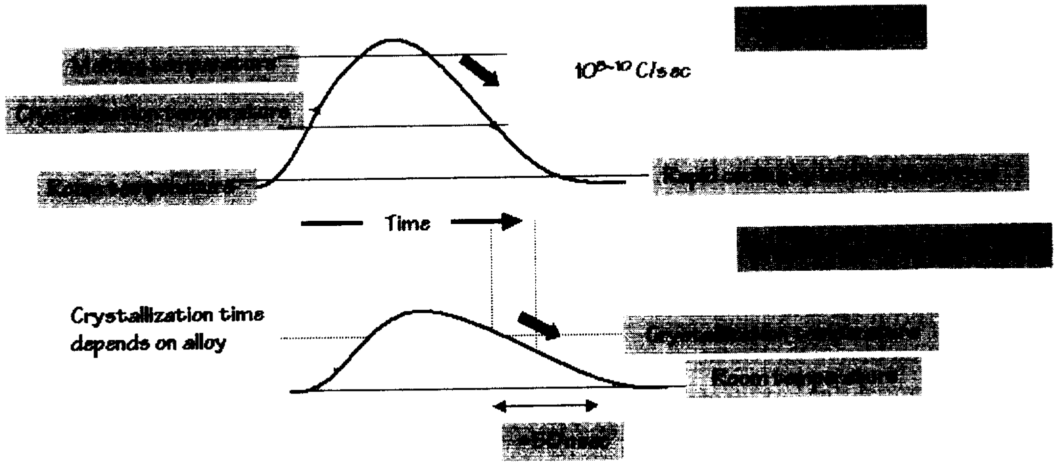
Alloys composed of such metals as Tellurium, Selenium, Antimony, Tin, Germanium and Silver have characteristics of transitioning between amorphous and polycrystalline states when subjected to temperature cycles.

Some alloys are more stable than others in each state. Rapidly heating the alloy to its melting point, and quickly cooling it can change it from the polycrystalline state to the amorphous state.

To revert to the polycrystalline state, the amorphous state alloy is heated to a temperature just above its crystallization point, then allowed to cool naturally to crystallize itself. The phase change processes are shown in Fig.6. The alloy has a higher level of light reflectivity when it is in crystalline state than when it is in amorphous state. This is the basic operation of phase change recording. The change in the light reflectivity is detected as the recorded data output.

Recording Process (Phase Change)

- Erased state ------ Crystalline phase
- Recorded state ------ Amorphous phase



Data rate limited by crystallization time characteristic of the alloy

- Current status –3 MB/sec
- Near term objective –10 MB/sec

Figure 6. Recording Process in Phase Change (PC) Media

An alloy of Te and Sb is preferred both for its environmental stability and large reflectivity change between states (typically <5% reflectivity in the amorphous state, and >20% in the crystalline state). Some other materials, such as Germanium, Indium, and Silver are often added to the basic alloy to improve its optical characteristics.

The structure of a typical phase change recording disc is shown in Fig.7. Thickness of the recording layer is ~20 nanometer, and its thickness significantly influences its lateral heat conductivity, which, in turn, controls the achievable data recording rate.

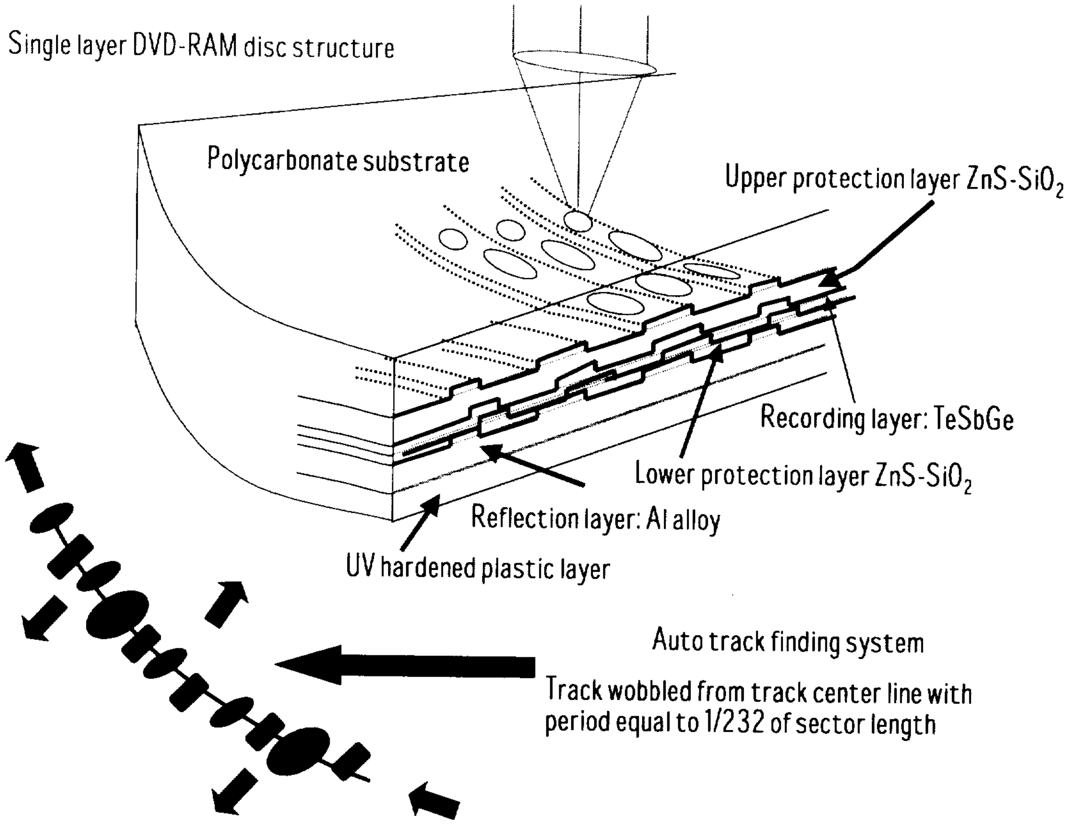

Single layer DVD-RAM disc structure
Polycarbonate substrate
Upper protection layer ZnS-SiO₂
Recording layer: TeSbGe
Lower protection layer ZnS-SiO₂
Reflection layer: Al alloy
UV hardened plastic layer
Auto track finding system
Track wobbled from track center line with period equal to 1/232 of sector length

Figure 7. Structure of single-layer Phase Change recordable DVD

The recording layer is sandwiched in between dielectric layers for protection. The protection layer material, such as Zinc Sulfide-Silicon Dioxide, $ZnS-SiO_2$, has a significantly higher melting temperature, and it stays in amorphous state during the entire recording process. The dielectric layers not only protect the recording layer during heating cycles to maintain its thickness constant, but they also improve the reflectivity change between states. This function is sometimes referred to as *optical tuning*.

Optimization of the disc construction, including the alloy composition and thickness of each layer, is an art of compromise.

For rapid heating during the initial record cycle, the record layer should hold all the heat within, but on the transition to amorphous state, the actual recording, quick heat dissipation is required. Slower rate of cooling could form crystals within the amorphous region, which is highly undesirable.

The process of returning to crystalline phase of non-recorded state is even more complex because the crystallization process for the material requires a defined rate of cooling, expressed as 10X degrees per second.

Lateral flow of the molten record layer material degrades the integrity of recorded information. Excessive pressure from the protection layers could cause this undesirable effect. Small amount of high melting temperature material in the record layer, which acts as an agent for lateral flow prevention, is a technique used by some of the media producers.

The structure shown in Fig.7 also illustrates the concept of land-groove recording, which offers two-for-one track pitch reduction. The land-groove recording does not have the inter track guard band for reduction/elimination of the data cross talks between tracks.

By making the elevation difference between the land and the groove to be exactly ¼ wavelength of the read laser wavelength, however, the signals from adjacent tracks picked up by the skirt energy of the read beam cancel by themselves. This technique is just as effective as the guard-band-less, azimuth recording used in high density magnetic tape recording.

DVD RAM

The DVD-RAM is a recording format intended for all data storage applications, including imagery, video, and numerical data. The first product, with 2.6 GB per surface capacity, was introduced in 1998. The 2.6 GB disc surface, as shown in Fig.8, is divided into two areas. In the central, non-recordable area, format and product information is stored. The outer area, where data is to be written, operates as a phase-change recording disc. The recordable area is divided into 24 concentric zones, zone-0 through zone 23. Each zone consists of 944 land tracks and 944 groove tracks.

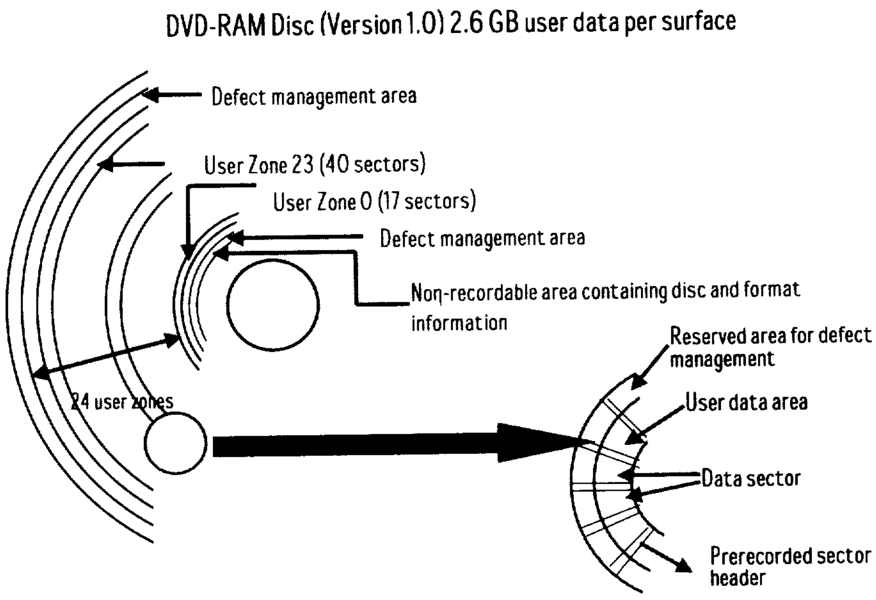

DVD-RAM Disc (Version 1.0) 2.6 GB user data per surface
Defect management area
User Zone 23 (40 sectors)
User Zone 0 (17 sectors)
Defect management area
Non-recordable area containing disc and format information
24 user zones
Reserved area for defect management
User data area
Data sector
Prerecorded sector header

Figure 8. DVD-RAM version 1.0

To maintain near constant data recording density throughout the disc surface, more data is recorded in outer zones than in the inner zones. This is accomplished by recording varying number of data sectors - - each sector has 2 KB user data - - in each zone. The inner most zone holds 17 sectors. One sector is added to each zone as the zone moves outward, and the outer most zone holds 40 sectors. This technique is referred to as ZCLV, Zone Constant Linear Velocity recording. The head-to-medium velocity is essentially constant at 6.0 meters/sec.

Each zone contains areas normally used for data storage and reserved areas to be used as replacement for defective user blocks. Defect area replacement is done on a sector basis, and it is automatically accomplished by the Defect Sector Management, DSM, operation.

The operational instructions, and data related to defect area replacement, are stored in the inner- and outer-most portion of the disc, as shown in Fig. 8.

In addition to the land-and-groove recording, DVD-RAM employs a technique which significantly improves the in-track density - mark edge recording. In conventional optical recording, as shown in Fig. 9 A, 1 is recorded as a mark, or pit, on the medium.

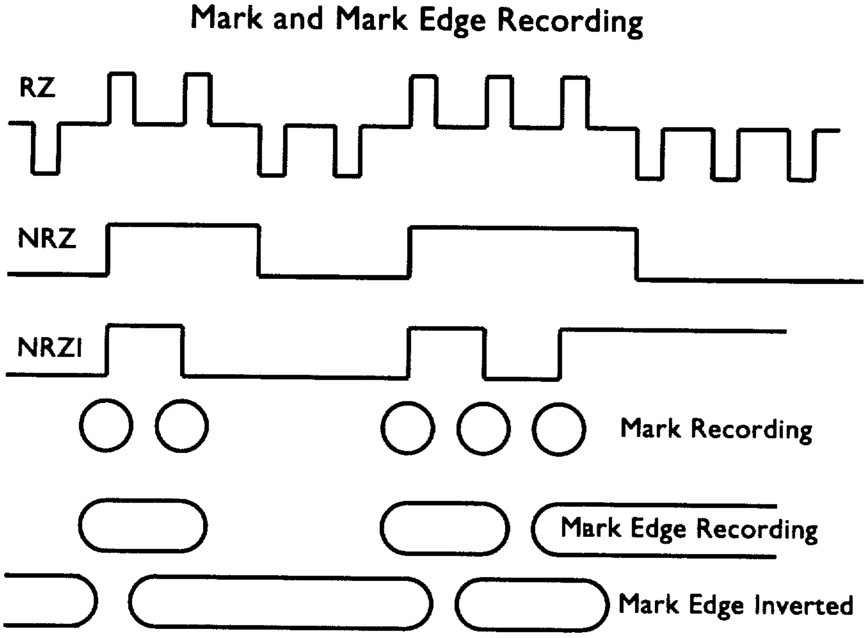


Figure 9. Mark edge recording

In a consecutive run of 1's, the successive recording of pits tends to raise the local medium temperature higher, and unless carefully controlled, the pits recorded get successively larger. This phenomenon, coupled with the fact that pits are closely placed to begin with, makes the minimum bit spacing longer than what is dictated by the actual bit size itself.

Mark edge recording offers an attractive solution for this problem. In mark edge recording, the consecutive 1's are recorded as an elongated pit, each edge corresponding to the start and end of the consecutive 1's. This technique makes it unnecessary to record a short pit, enabling the system to record a significantly higher amount of data within the given physical media length.

The 2.6 GB DVD-RAM media is available in Single Side version as well as in Double Sided version. Unlike the DVD-Video Dual layer disc, Double sided DVD-RAM must be turned over to write and read both sides by a writer/reader drive. Both layers within a DVD-Video dual layer disc are readable by a standard DVD player without reinserting the disc.

Specifications for the higher capacity version DVD-RAM, with per surface capacity of 4.7 GB, have been finalized.

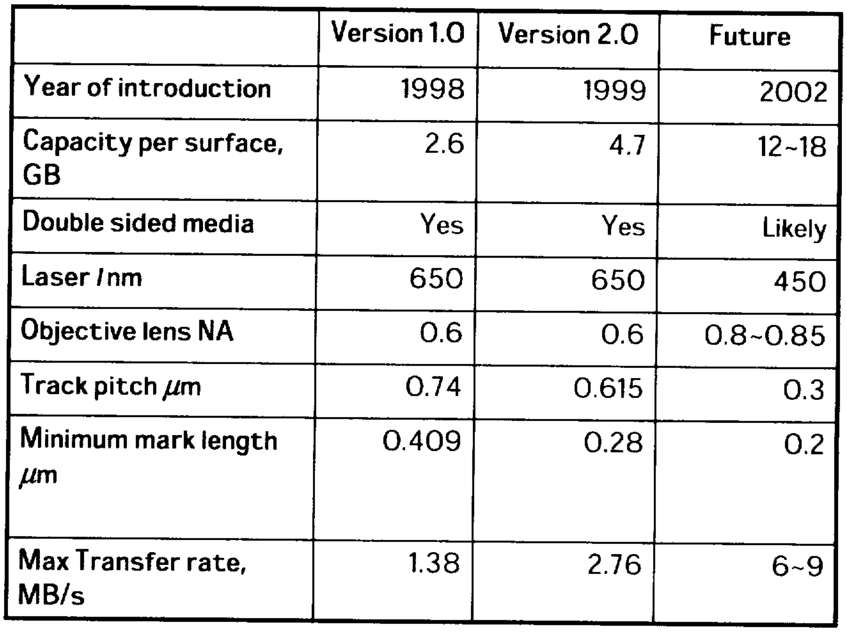
Version 2.0 of the specifications for the 4.7 GB disc, state the following characteristics:

Track Pitch = 0.615 μm, Minimum mark Length = 0.28 μm
No. of Zones = 35
Number of Sectors in innermost zone = 25
Number of Sectors in outermost zone = 59

The 4.7 GB capacity disc is expected to be available in both single sided (4.7 GB) and double sided (9.4 GB) versions.

The data transfer rate for the high capacity version is twice that of the 2.6 GB version, at 2.76 MB/sec. This is the sustainable record rate of the system. As in the case for CD-ROM, a DVD-RAM disc can be read at a much faster rate than its record rate.

Specifications for the recordable DVD's are shown in Fig. 10.



	Version 1.0	Version 2.0	Future
Year of introduction	1998	1999	2002
Capacity per surface, GB	2.6	4.7	12~18
Double sided media	Yes	Yes	Likely
Laser / nm	650	650	450
Objective lens NA	0.6	0.6	0.8~0.85
Track pitch μm	0.74	0.615	0.3
Minimum mark length μm	0.409	0.28	0.2
Max Transfer rate, MB/s	1.38	2.76	6~9

Figure 10. DVD-RAM specifications

DVD Capability Expansion Possibilities

DVD proponents faced the age-old dilemma of designers: when to stop enegineering development and finalize the specifications. The technology was on a steep ascending curve then, and the disc data holding capacity was growing daily

Areal Density Improvement

Today, they were facing exactly the same problem. The blue/Violet light diode laser in 400 nanometer wavelength region is no longer a laboratory curiosity. While the price is quite steep, a manufacturer is delivering a 405 nanometer laser based on Gallium Nitride Crystal with 10 mw CW capability, and guaranteeing 10,000 hours operation under room temperature. A score of other semiconductor manufacturers are all developing a laser of similar performance and capability. The major supplier of current CD and DVD laser established a two year project to complete the development of their short wavelength laser.

Another component requiring up-grades to increase the areal density is the objective lens. In this area also, experimental lenses approaching the theoretical limit of unity, in the region of 0.8 to 0.9, have been produced.

Combining a blue/violet laser and a super fast objective lens, an areal density increase of 3 to 5 is practical, making the per surface data capacity to be 15 to 25 GB. A number of 15 to 25 GB experimental disc systems have been demonstrated within the last two years.

Multiple layer Recording

While it is practical to read a dual-layered, embossed pattern DVD-Video disc from its front side, writing to and reading data from a multi layered disc from its front side creates an entirely new set of problems.

Taking a double layer disc as an example, the first and the closest to the front surface record layer must be transparent enough so that the record and play back beams for the second layer can penetrate it without undue attenuation. In case of the playback beam for the second layer, it must pass through the first layer twice.

Also, if the reflective layer for the first recording layer is retained, it must be at least semi-transparent.

Therefore, the multi-layer recording disc must have an entirely different layering structure. A possible structure of dual layer, phase change recording disc is shown in Fig. 12.

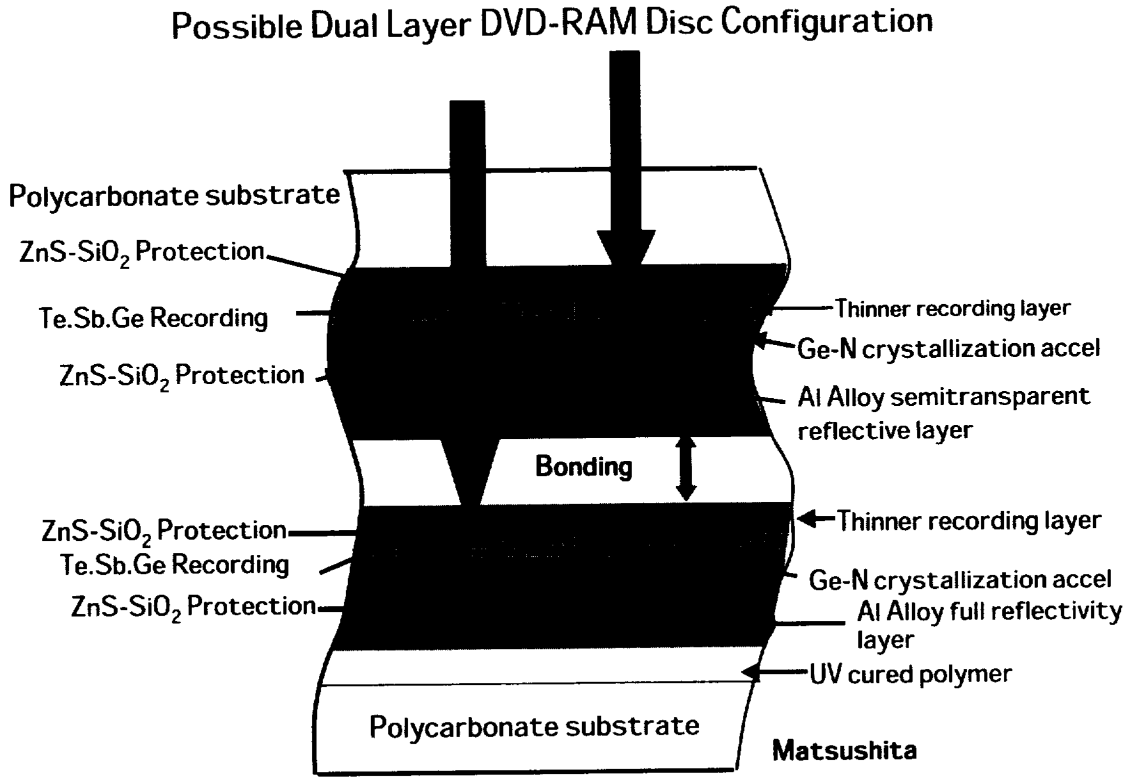


Figure 12. Possible Dual Layer DVD-RAM

Faster Recording Rate

One of the weakness of the phase change recording is its relatively slow recording rate. In version 2.0 of DVD-RAM, it has just reached about 3 MB/sec.

An anticipated application of DVD-RAM is the broadcast video recording camera, the professional camcorder equipment. While 3 MB/sec may be just sufficient for the standard definition television recording, it is entirely inadequate for HDTV applications, where 6 to 12 MB/sec recording rate is required.

As mentioned earlier, the recording rate is essentially governed by the crystallization rate of the recording layer material. In addition to thickness variation and optimization of composition, studies are being conducted to accelerate crystallization by placing an *augmentation* layer next to the recording layer. On the reading side, all DVD discs are already readable at 10 MB/sec, and within three years, technologies will be available to read the disc at rates as fast as 40 MB/sec.

Summary

The entire DVD infrastructure is being driven by the large commercial business. Continuing technology advances are assured because of the rapidly growing markets demanding higher performance and feature-packed products.

DVD-RAM is an attractive data storage technology, offering a high data capacity per unit volume, with expected storage life of 60 years or longer. It could potentially displace magnetic tape in certain applications.

Magneto-Optical recording, while making steady progress in data storage density, is used in the data storage area only, lacking the support enjoyed by DVD from the consumer industry. It can record faster than DVD, and is certain to maintain its place in the overall data storage infrastructure.

Write-once technology, represented by CD-R and DVD-R, is developing a unique position in both private life and business environs. It is a replacement for paper-based notebooks, log books, ledgers, and a convenient and low cost storage for music and pictures.

EOSDIS: Archive and Distribution Systems in the Year 2000

Jeanne Behnke
Goddard Space Flight Center - Code 423
Greenbelt, MD 20771
jeanne.behnke@gsfc.nasa.gov
tel: +301-614-5326
Alla Lake
Lockheed Martin Corp
1616 McCormick Drive, Upper Marlboro, MD 20774
alake@eos.hitc.com
tel: +1-301-925-0626
fax: +1-301-925-0651

Abstract

Earth Science Enterprise (ESE) is a long-term NASA research mission to study the processes leading to global climate change. The Earth Observing System (EOS) is a NASA campaign of satellite observatories that are a major component of ESE. The EOS Data and Information System (EOSDIS) is another component of ESE that will provide the Earth science community with easy, affordable, and reliable access to Earth science data. EOSDIS is a distributed system, with major facilities at seven Distributed Active Archive Centers (DAACs) located throughout the United States. The EOSDIS software architecture is being designed to receive, process, and archive several terabytes of science data on a daily basis. Thousands of science users and perhaps several hundred thousands of non-science users are expected to access the system. The first major set of data to be archived in the EOSDIS is from Landsat-7. Another EOS satellite, Terra, was launched on December 18, 1999. With the Terra launch, the EOSDIS will be required to support approximately one terabyte of data into and out of the archives per day. Since EOS is a multi-mission program, including the launch of more satellites and many other missions, the role of the archive systems becomes larger and more critical. In 1995, at the fourth convening of NASA Mass Storage Systems and Technologies Conference, the development plans for the EOSDIS information system and archive were described [1]. Five years later, many changes have occurred in the effort to field an operational system. It is interesting to reflect on some of the changes driving the archive technology and system development for EOSDIS. This paper principally describes the Data Server subsystem including how the other subsystems access the archive, the nature of the data repository, and the mass-storage I/O management. The paper reviews the system architecture (both hardware and software) of the basic components of the archive. It discusses the operations concept, code development, and testing phase of the system. Finally, it describes the future plans for the archive.

Introduction

Earth Science Enterprise (ESE) is a long-term NASA research mission to study the processes leading to global climate change. The Earth Observing System (EOS) is a NASA campaign of satellite observatories that are a major component of ESE. The EOS Data and Information System (EOSDIS) is another component of ESE that will provide the Earth science community with easy, affordable, and reliable access to Earth science data. EOSDIS is a distributed system, with major facilities at data centers located throughout the United States.

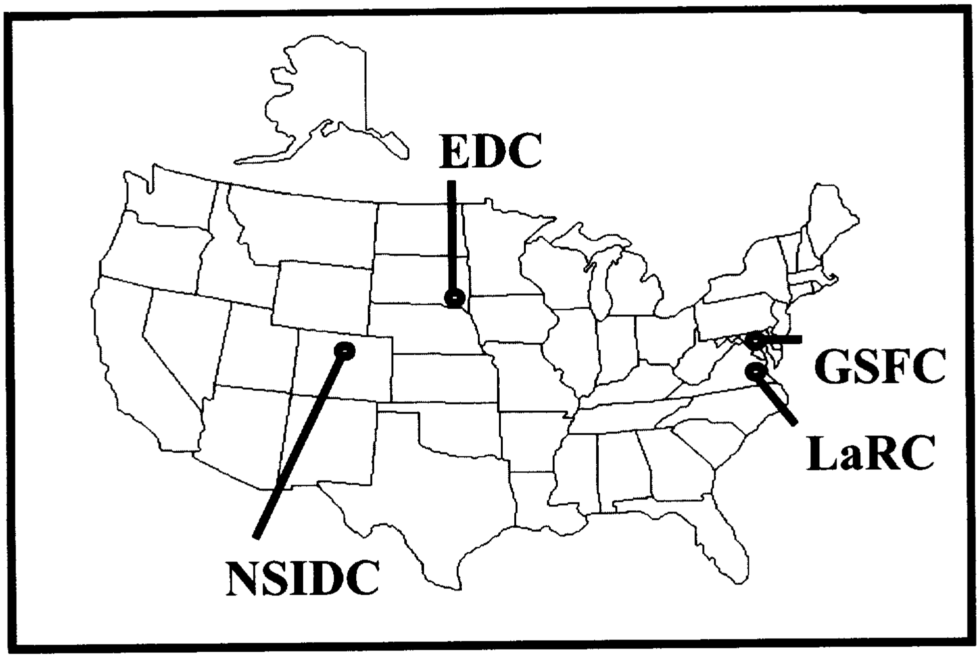


Figure 1. Locations of EOSDIS Distributed Active Archive Centers

In this paper, we describe the archive and distribution operations at four Distributed Active Archive Centers (DAACs). These DAACs are located at Goddard Space Flight Center (Greenbelt, MD), Langley Research Center (Hampton, VA), EROS Data Center (Sioux Falls, SD), and the National Snow and Ice Data Center (Boulder, CO). The EOSDIS software architecture is being designed to receive, process, and archive several terabytes of science data on a daily basis. Thousands of science users and perhaps several hundred thousands of non-science users are expected to access the system.

The first major set of data to be archived in the EOSDIS is from Landsat-7. Landsat-7, an Earth imaging satellite launched on April 15, 1999, provides repetitive, synoptic coverage of continental surfaces; spectral bands in the visible, near-infrared, short-wave, and thermal infrared regions of the electromagnetic spectrum; average spatial resolution of 30 meters (98-feet); and absolute radiometric calibration (http://landsat.gsfc.nasa.gov).

Following Landsat-7, the Terra satellite (formerly known as EOS AM-1) was launched on December 18, 1999. Terra is uniquely designed for "comprehensive" Earth observations and scientific analysis, covering science priorities as land cover change and global productivity, seasonal-to-interannual climate predictions, natural hazards, long-term climate variability, and atmospheric ozone (http://terra.nasa.gov). With the Terra launch, the EOSDIS will be required to support approximately one terabyte of data into and out of the archives per day. The next big mission is the Aqua satellite to be launched in December 2000 (http://aqua.gsfc.nasa.gov). Moreover, EOS is a multi-mission program that includes several more Earth study campaigns and satellites through the year 2011. Given this extended time frame, the role of the archive systems becomes larger and more critical. In 1995, at the fourth convening of NASA Mass Storage Systems and Technologies Conference, the development plans for the EOSDIS information system and archive were described [1]. Five years later, many changes have occurred in the effort to field an operational system. It is interesting to reflect on some of the changes driving the archive technology and system development for EOSDIS.

The focus of this paper is the description of the Science Data Processing System (SDPS) segment of EOSDIS, with particular attention to the ingest, archive and distribution processes and components. The SDPS system will be required to manage, store, retrieve, and process more than a terabyte of data per day at its data centers. As Table 1 illustrates, the projected capacity required by the project during 2000 is quite formidable. Across the data centers, the expectation is to archive on the order of 1.5 TB per day and 16,500 granules. A granule is the smallest package of data made available by EOSDIS. A granule can contain 1 or many files. Another important distinction is made between a full dataset and a "browse" dataset. Browse datasets can be thought of as small examples of the full resolution data. They are used by scientists to quickly determine whether a particular dataset is useful without having to look at its entire contents.

Data Center	Archive Volumes GB/Day	# of granules per day	Archive Volumes In TB per year	# of Granules cumulative per year	Distribution via Network GB/day	Distribution via tape GB/day
EDC	522	6886	190	2,513,390	194	159
GSFC	688	5545	251	2,023,925	226	226
LaRC	312	2945	114	1,074,925	102	102
NSIDC	22	1083	8	395,295	6	6
Total	**1544**	**16459**	**563**	**6,007,535**	**528**	**493**

Table 1. Projected capacity through the end of 2000

In addition to designing and providing a comprehensive data retrieval and processing system, the SDPS is tasked to be a flexible, scaleable and reliable system. The architecture should be capable of supporting:

- new data types with minimal software modifications
- new data centers that will not require new code and software agreements
- standard interfaces (HDF-EOS) enabling coordinated data analysis
- data access from a wide variety of users (e.g., kindergarten teachers, as well as college professors)
- technological advances and the infusion of new COTS products and techniques (e.g., new file storage management systems)
- inevitable change and new additions

To meet the challenge of the SDPS, the EOSDIS Core System (ECS) was designed under contract to NASA by the Raytheon Systems Company/Landover MD. Lockheed Martin Corporation designed the archival component of the system encompassing Ingest, Storage Management, and Distribution, under a subcontract to the Raytheon Systems. It is an enormous development effort for the entire ECS comprises 75 COTS packages, about 1 million lines of code and the efforts of approximately 220 developers. In this paper, we describe how the archive and distribution systems work for the ECS.

The ECS system is designed at a central development location and then distributed and installed at the various DAAC sites. Each of the DAACs has a different area of science emphasis and the system to be deployed is adapted to that need. For example, not all DAACs will have the same archive size requirements. However, the software system works the same way at all DAAC sites. The science datasets supported by this SDPS also vary in size and type. The SDPS is composed of six major subsystems shown in Figure 2, ECS Context Diagram.

1. INGEST subsystem - receives data from external and internal sources and submits them for storage into the archive
2. DATASERVER subsystem - archives and distributes data
3. PLANNING subsystem - develops plans for producing data products (level 0 to level 1)
4. DATA PROCESSING subsystem - manages, queues and executes processes for the generation of data products
5. INTEROPERABILITY subsystem - provides the software infrastructure for the communications between clients and servers in the system
6. DATA MANAGEMENT subsystem - supports the location, search, and access of data and services.

This paper will principally describe the Data Server subsystem including the description of how the other subsystems access the archive, the design of the data repository, the mass-storage I/O management, and archive operations. The paper will review the system architecture (both hardware and software) of the basic components of the archive

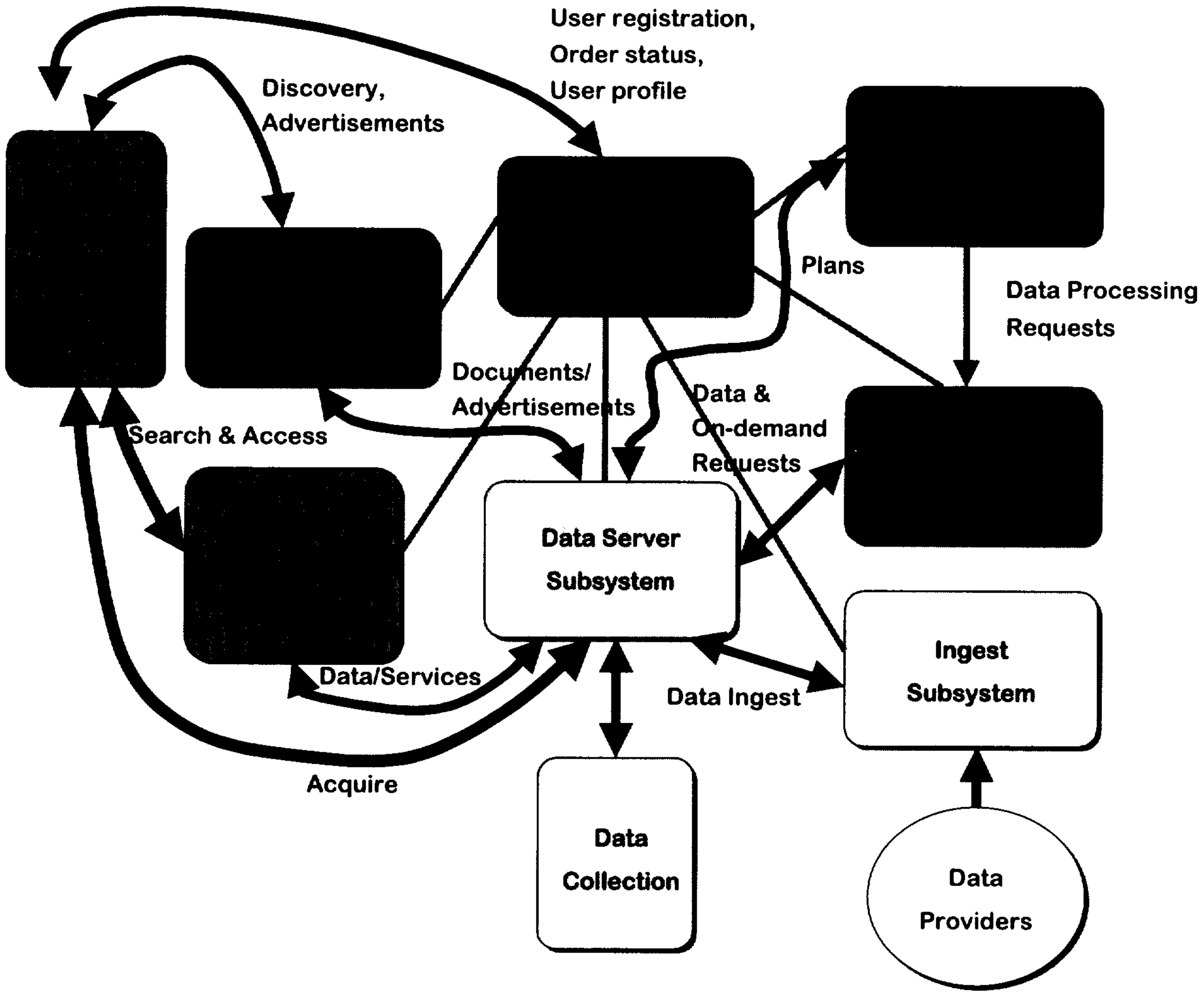


Figure 2. ECS context diagram

It will discuss the operations concept, code development, and testing phase of the system. Finally, it will describe the future plans for the archive.

Data Server Subsystem

The Data Server Subsystem (DSS) provides the data storage and management functions including archiving EOS data, managing and searching the archive, and resource staging. It stores, searches, retrieves and distributes EOS data. The DSS interfaces with virtually all ECS subsystems and components. It is composed of several internal subsystems and constitutes the largest software and hardware segment of the entire ECS. The subsystems internal to DSS include the Storage Management Configuration Item (CI), Data Distribution CI, and the Science Data Server CI.

As with all major systems in ECS, the DSS was written in C++ using an object-oriented software methodology. The DSS uses the Distributed Computing Environment (DCE) for its infrastructure and ClearCase to manage the software configuration. Many other commercial packages are used to develop, build and operate the system. The DSS contains 252,000 lines of custom code. Ingest, which is a separate subsystem designed to load the archive and enter the metadata into the inventory tables, is composed of 83,000 lines of code. The system is extensively tested prior to being fielded. The ECS was originally developed on small workstations that didn't adequately emulate the hardware being fielded at the DAACS. During the course of the five years, it became clear that the development team would require a complete archive system that duplicates the configuration of the archive systems at the larger DAACs in order to develop and test the software systems effectively. This archive system, called the Performance Verification Center, was created to not only field new versions of the software but also to provide the ability to troubleshoot and tune the system to maximize performance.

The Science Data Server CI subsystem in the DSS provides the entire ECS system with a catalog of data holdings organized by Earth Science Data Types (ESDT). The ESDT includes not only the data type definitions but also service functions that can be performed on that specific data. The Science Data Server manages and provides user access to data collections through its catalog of metadata, principally using the Sybase database management system. When another subsystem (for example, the Data Processing Subsystem) requests data from the archive, the request is sent to the Science Data Server subsystem. Science Data Server then initiates a request to the Storage Management subsystem, to allocate magnetic disk space for staging of that data and a request to the Data Distribution subsystem to stage and distribute the data appropriately to the requestor. The Data Distribution subsystem requests the data from the Storage Management subsystem. The Storage Management subsystem initiates the acquisition of the data from the physical storage in a robotic silo and stages the data in the appropriate disk space that it manages.

The Storage Management CI stores, manages, and retrieves data files on behalf of the other subsystems. It also manages all the magnetic disk space within the DSS. An archive software server is used to manage requests from the other subsystems to store or retrieve the archive data. A staging disk server is used to manage the files in the magnetic disk storage area and a pull monitor server is used to manage the files that are in the 'user pull' area. The magnetic disk is also used as a Storage Management disk cache, which is managed by a staging monitor server. The 'user pull' disk space is disk space allocated for users to FTP their requested data from the EOSDIS. A resource manager server has also been developed to manage the peripheral devices available at each data center. These include 8mm tape drives in stackers, D3 tape drives at EDC, and, in the near future, CD-ROM drives and DLT tape drives in stackers.

The Data Distribution CI formats and distributes data to users, either electronically or on physical media (e.g. 8mm tapes). It directs the Storage Management subsystem to place data in the desired location. The request could be to place the data on magnetic disk for another subsystem to retrieve it, to copy the data to tape, or to push the data via FTP to the user's workstation. This CI sends distribution notifications as the action is completed. The data distribution server provides control and coordination for data distribution through request processing. More on the request processing design can be found in the poster paper by J. Crawford presented at the Eighth NASA Mass Storage Systems and Technologies Conference. [3]

Data Repository

The Data Repository component includes the nearline storage system, cache magnetic disks, and servers required for storing and retrieving the EOS data, see Figure 3, Hardware Architecture Diagram. The architecture is specialized to insure high availability [4], high-speed access to the data in the nearline system. Because the amount of data to be stored in the EOS system is so large, the system had to be designed to use high storage density at low cost and, therefore, is a tape-based archive. In 1995, ECS was planning to purchase AML multi-media robotics from the EMASS corporation. As the requirements for the project evolved, the decision was made in 1996 to have a multi-vendor solution, with the EMASS Corporation supplying the AML multi-media robotics for the small file size Browse collection and Storage Tek Powderhorn silos for the large data archive. AML robotics are attractive, based on their support for drives and media from different vendors. After several months of evaluation, it was decided that it would be a cost benefit to trade the four AMLs for Storage Tek Powderhorns, which is our current configuration.

Today, the nearline storage system used by EOS is based on StorageTek Powderhorn silos as the hardware base and the AMASS (Archival Management And Storage System), a product of ADIC. StorageTek silos have been installed at the data centers: three silos at EROS Data Center (EDC), two at Langley Research Center (LaRC), one at NSIDC and four at Goddard Space Flight Center (GSFC). Each silo holds up to 6000 cartridge tapes,

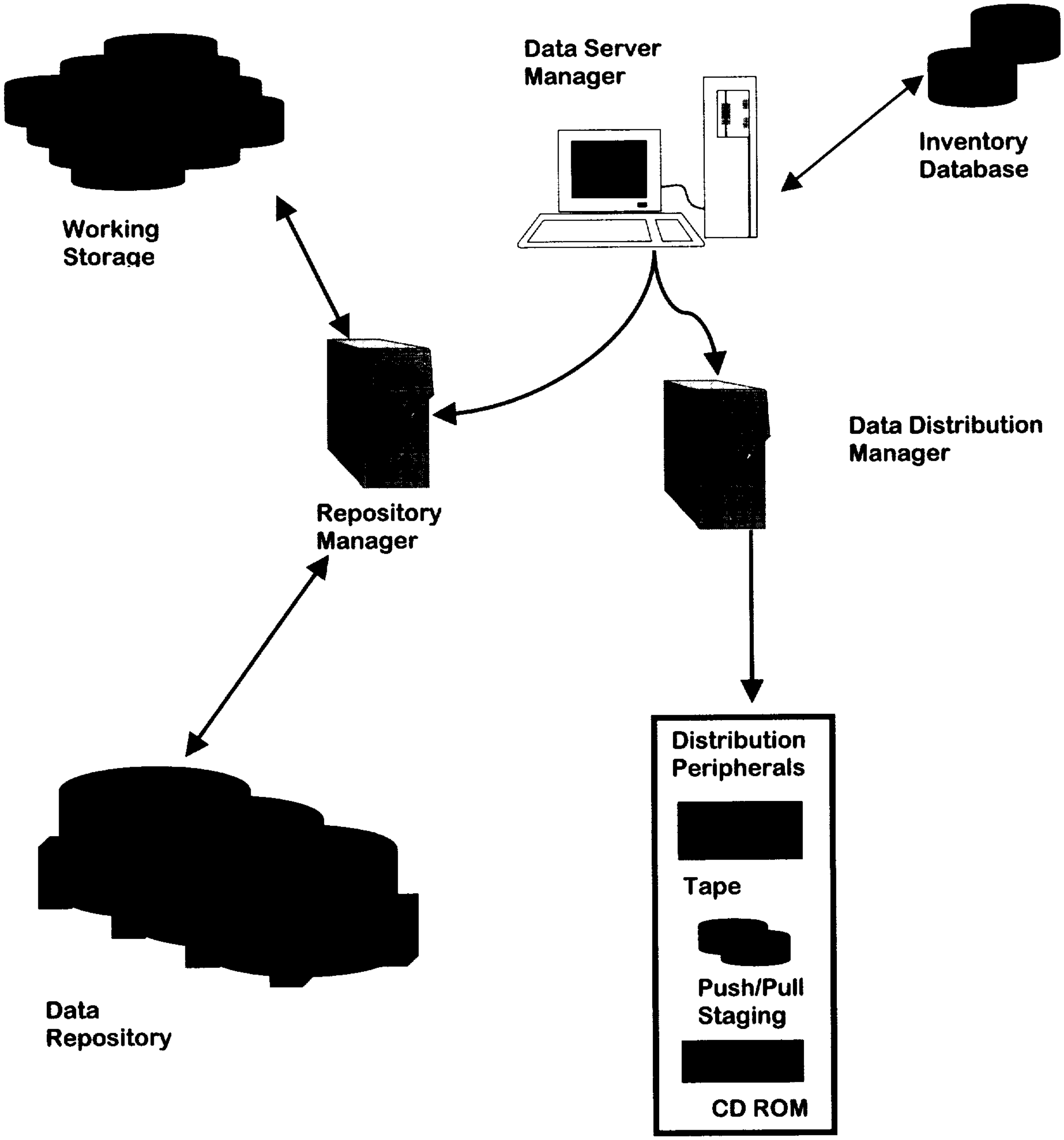


Figure 3. ECS Hardware Architecture Diagram

in our case 50GB D3 tapes, for an aggregate of approximately 300 TB in a silo. The actual number of media in a silo is derated by the number of attached tape drive enclosures and by a Plexiglas observation window, if used. On average, each EOSDIS silo filled exclusively with D3 media stores 270 TB of data without compression. Some data, however, is compressible at the drive. The Landsat collection is one example where realized data compression approximates 2:1. A silo containing Landsat data at the EDC will, when filled to capacity, store between 500 and 600 TB.

The number of tape drives attached to each of the silos varies depending on the data throughput requirements of an archival site. The silos at the larger sites, such as EDC and GSFC, run with eight D3 drives in each of the archive silos. A smaller site, such as NSIDC, has 3 D3 drives in its data silo. The drives are rated at a maximum sustained throughput of 11 MB/sec, but the observed effective rate with compression is near 16 MB/sec. The number of storage silos at a data center is also determined by the data center size, i.e. the cumulative size of the data holdings in storage. The physical storage of data is managed by the AMASS file storage management system.

Both the hardware and software system design supports the growth of ECS. NASA has planned to grow the system to support the archiving of future missions with additional hardware over the ECS program lifetime. For example, NASA purchased an additional silo for each of the larger data centers last summer. The silo was easily incorporated into the architecture. Another ECS design modification was to place the Browse Data Collection on STK 9840 tapes and house these tapes in the STK Powderhorn silos. That design was driven by both a very large accumulation (up to 30 TB at GSFC) of Browse data and the relatively small file size of each Browse file - in the 1 MB range. Since even at the larger sites the Browse collection will fill only part of the silo, the Browse silos may be used in a multi-media mode, outfitted with both 9840 and higher tape capacity drives and media. Just like D3, 9840 is a fast streaming drive, rated at 12 MB/sec maximum sustained throughput rate and capable of effective data rate of 17 MB/sec with compression. Unlike the Helical Scan D3, 9840 is a linear tape drive – more suitable for a start and stop operation mode associated with smaller data files.

Silicon Graphics (SGI) workstations were chosen as the platforms for managing the data repository. The current configuration assigns a single SGI Challenge host per STK silo in order to sustain the required data rates to and from the tape drives through the attached buffer RAID. Significant effort was expended in tuning the SCSI attached RAID to produce the desired effective data rates of 120 MB/sec per RAID subsystem [2]. Over the summer, ECS will migratethe existing archive servers to SGI Origin platforms. Although the combination of Origin servers and fibre attached RAID afford much faster data rates, the same 1 server per 1 silo ratio will be preserved initially to allow for redundancy.

Mass-Storage I/O Management

The greatest challenge for the DSS is the management of the massive I/O (multiple terabytes per day) between the archive and the ECS components requesting data actions. It must handle continuous requests from

1. The Data Processing subsystem for files needed for processing Terra or Landsat data and for storing products once they are created;

2. The Ingest subsystem to store data from data providers external to ECS;
3. The client subsystems for data ordered by users from the GUI front-end;
4. The ECS subscription service to send data to users when it gets stored in the database.

The physical storage of ECS data is managed via a "commercial off the shelf" (COTS) AMASS files storage management system by ADIC. The AMASS system runs on the SGI hosts and operates the STK hardware silos. The control of the robotic mechanism of the silo (loading and unloading of the tapes) is via the STK Automated Cartridge System Library Software (ACSLS) running on a SPARC5 SUN workstation. AMASS addresses the ACSLS through a network connection. The ACSLS controls the robot directly via an RS232 line.

AMASS is a direct access file system as opposed to a Hierarchical Storage Management (HSM) product. Its cache area serves as a write-through buffer. The size of AMASS cache is set independently at each data center and each particular server. It is determined on the basis of the expected storage and retrieval profile associated with the data types handled by the server. Predominant file size, many small files or many large files for example, plays a role in choosing the exact configuration. Although AMASS supports both FTP and NFS access to the archived data, NFS is used solely by the ECS system. Unfortunately, AMASS uses an internal database that is singlethreaded in the implemented version. The performance constraints that it places on the overall system are mitigated by using custom-code manipulation of AMASS and the hardware. AMASS uses an internal database for tracking file allocations to tape. This internal database is journaled. The location of tapes in the silo slots is tracked by an Oracle database in the ACSLS. To enhance performance, AMASS allows creation of specialized volume groups of tapes in the silos. In our case, these are created for particular EOSDIS data types.

Archive Operations

With the launch of Terra, archive operations for EOSDIS are now fully established at all the data centers. Many of the data centers are operational on a 24 x 7 basis, however, the ECS system has been designed with a 'lights out' approach. To be able to maintain the required ingest and distribution rates, the software is designed to be highly autonomous. The only area of operations requiring direct human involvement is hard media distribution, where the distribution media must be loaded and unloaded and packaged for shipment to the user.

A Systems Monitoring Center has been built at GSFC to monitor each of the data centers as well as to provide some special, centralized functions. Locally at each data center, all areas of operation are closely monitored, especially process and log monitoring. The operations system administrators and system engineers are automatically paged if specific events or error conditions are encountered. For example, a severe error in the archive systems will trigger an event 'page call' to the engineers. This enables staffing to be minimal even at the largest data centers during off-hours. Routine full system backup of the software is performed using DLT tape. The AMASS database (ORACLE) in the archives is backed up at some data centers as frequently as every hour. Backup of the actual data is determined at each data center and is dependent on the data type. It can be done as simply as setting aside an additional volume group in a secondary silo for a backup copy. One of the most important concerns in operating of the mass storage system for ECS was the ability to recover from tape errors. A recovery procedure that is combination of automated scripts, custom code, operator actions and vendor actions has been designed and tested.

Each data center supports three modes in its current system environment. There are two test modes and one operational mode. The routine work of the data center is performed in the operational mode. The two test modes are used for testing and installing software patches/releases and COTS patches/releases. For the ECS system, it was important to fully test the archive systems as much as possible prior to becoming operational. We tested both the hardware and software for functionality as well as performance. Many problems and features of the systems were discovered during the test phase. The archive vendor was notified and fixes were supplied. In some cases, ECS compensated with custom code. Testing the archives required almost two full-time engineers in advance of being operational and will continue to require test engineers during the operational EOS lifetime. Each data center also requires support from very experienced archive engineers.

Conclusions

The focus of this paper is the Data Server subsystem, the archive component of the EOSDIS Core System (ECS). It comprises the data repository, the mass storage I/O management and the software configuration items needed to manage, store, retrieve, and process massive amounts of EOSDIS data on a daily basis. The Data Server Subsystem design is constrained in many areas by the schedule pressures, cost considerations, realities of implementing large and very complex system, and sheer technical limitations existing at the time of initial design. Even so, the initial performance results are satisfactory to meeting the data flow and operating requirements. In order to continue to meet the requirements of the future missions and the anticipated growth in user demands, the system must continue to evolve. The ECS design allows for such evolution in areas of both the custom implementation and replacement of outdated COTS technologies by their successors. Several evolutionary steps, such as robotic technology replacement and migration to the new SGI server model, have already been undertaken or are in the process of being undertaken. In the foreseeable future, the hardware architecture is limited to tape archives, although a plan for regular migration to alternative tape media is being considered. Some changes may be anticipated by evolving storage area networks, file storage management software and disk technology. The very design alterations that took place during the course of the project represent the systems capacity to evolve. We have described the archive and distribution systems for the EOSDIS Core System (ECS) for 2000 in this paper, however there are many other components to the system. Please feel free to contact the authors for further information.

References

[1] Caulk, P.M., "The Design of a Petabyte Archive and Distribution System for the NASA ECS Project", Fourth NASA Goddard Conference on Mass Storage Systems and Technologies, College Park, Maryland, March 1995.

[2] Lake, A., "Performance Tuning of a High Capacity/High Performance Archive for the Earth Observing Systems Project", Sixth Goddard Conference on mass Storage Systems and Technologies, College Park, Maryland, March 1998.

[3] Crawford, J.M., "A Scalable Architecture for Maximizing Concurrency", Eighth NASA Goddard Space Flight Center Conference on Mass Storage Systems and Technologies, College Park, Maryland, March 2000.

[4]Lake, A., Crawford, J., Simanowith R., Koenig, B., "Fault Tolerant Design in the Earth Orbiting Systems Archive", Eighth NASA Goddard Space Flight Center Conference on Mass Storage Systems and Technologies, College Park, Maryland, March 2000.

Mass Storage System Upgrades at the NASA Center for Computational Sciences

Adina Tarshish
NASA Center for Computational Sciences, Code 931
NASA/Goddard Space Flight Center
Greenbelt, MD 20771
Adina.Tarshish@gsfc.nasa.gov
Tel +1-301-286-6592
Fax +1-301-286-1634

Ellen Salmon
NASA Center for Computational Sciences, Code 931
NASA/Goddard Space Flight Center
Greenbelt, MD 20771
Ellen.Salmon@gsfc.nasa.gov
Tel +1-301-286-7705
Fax +1-301-286-1634

Medora Macie
NASA Center for Computational Sciences, Code 931
NASA/Goddard Space Flight Center
Greenbelt, MD 20771
Medora.Macie@gsfc.nasa.gov
Tel +1-301-286-3812
Fax +1-301-286-1634

Marty Saletta
NASA Center for Computational Sciences, Code 931 (Raytheon)
NASA/Goddard Space Flight Center
Greenbelt, MD 20771
Marty.Saletta@gsfc.nasa.gov
Tel +1-301-286-9810
Fax +1-301-286-1634

Abstract

The NASA Center for Computational Sciences (NCCS) provides supercomputing and mass storage services to over 1200 Earth and space scientists. During the past two years, the mass storage system at the NCCS went through a great deal of changes both major and minor. Tape drives, silo control software, and the mass storage software itself were upgraded, and the mass storage platform was upgraded twice. Some of these upgrades were aimed at achieving year-2000 compliance, while others were simply upgrades to newer and better technologies. In this paper we will describe these upgrades.

1 Introduction

UniTree first arrived at the NCCS in July of 1992, when it was installed on a Convex C3240. At the time it was attached to 8 3480 tape drives in 2 StorageTek silos, 110 GB of disk, and its main client was a Cray Y-MP. Our UniTree system now runs on a Sun E10000 and is connected to 56 tape drives in 7 silos and an IBM 3494 robotic library, as well as 4 freestanding Timberline drives. It has a disk cache of 1.5 TB, and its main clients are SGI/Cray J932se machines. Figure 1 below shows our current configuration.

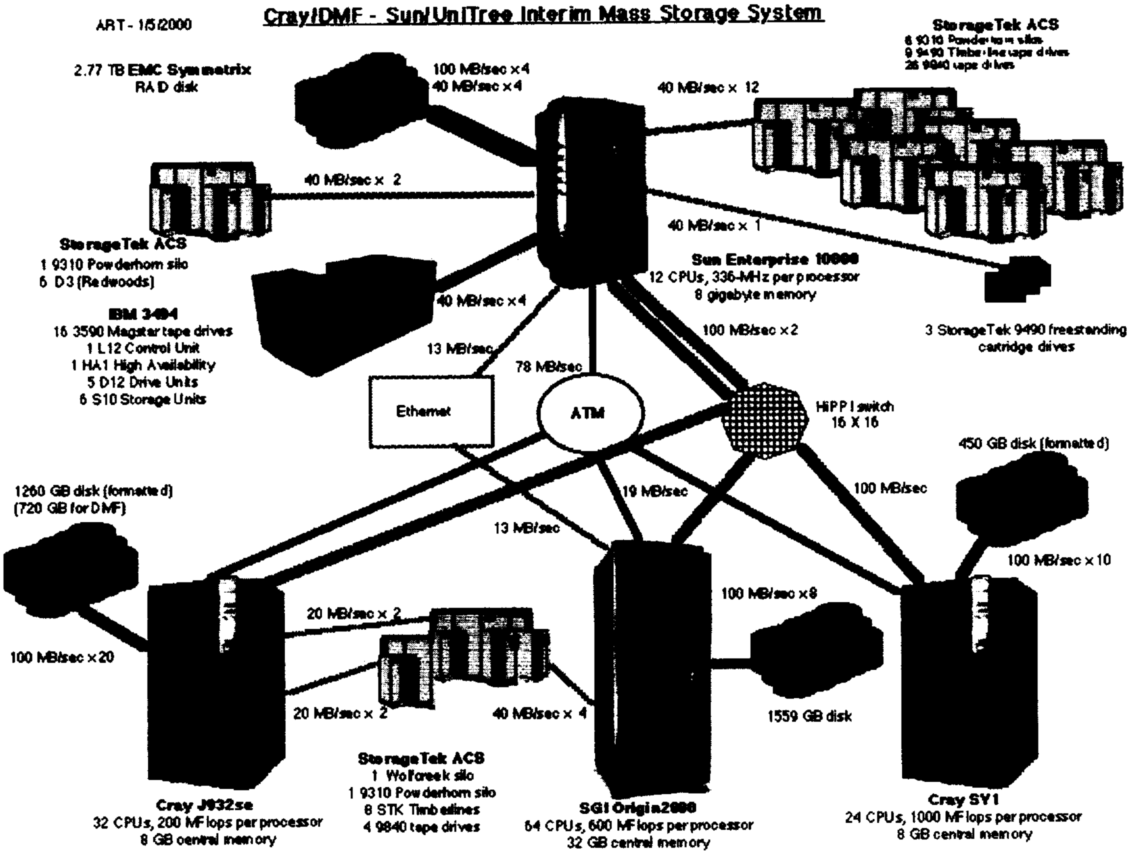


Figure 1. NCCS supercomputing/mass storage configuration

As of November 1, 1999 there were nearly 69 TB of unique data under UniTree's control, plus 35.6 TB of duplicated data stored in a remote facility. Figure 2 shows the breakdown of data by category as of that date.

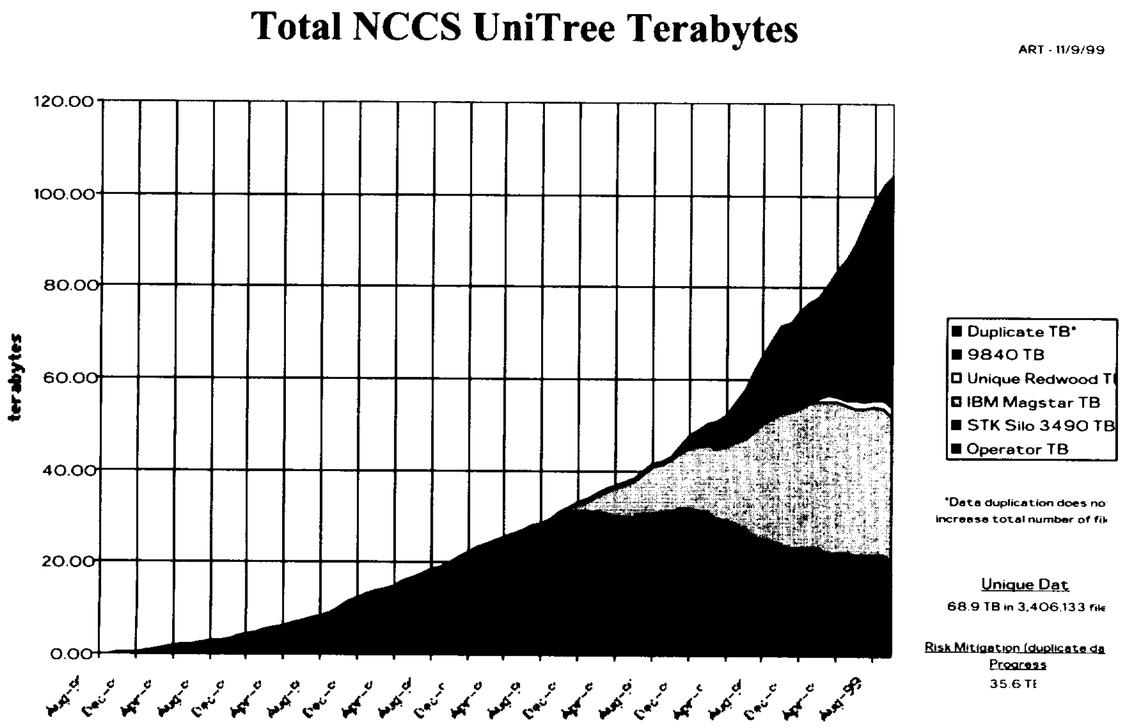


Figure 2. breakdown of data under NCCS UniTree control as of 11/1/99

NCCS users have done as much as 300 GB of network traffic in a single peak day. Figure 3 shows the weekly network traffic to and from UniTree for the past two years.

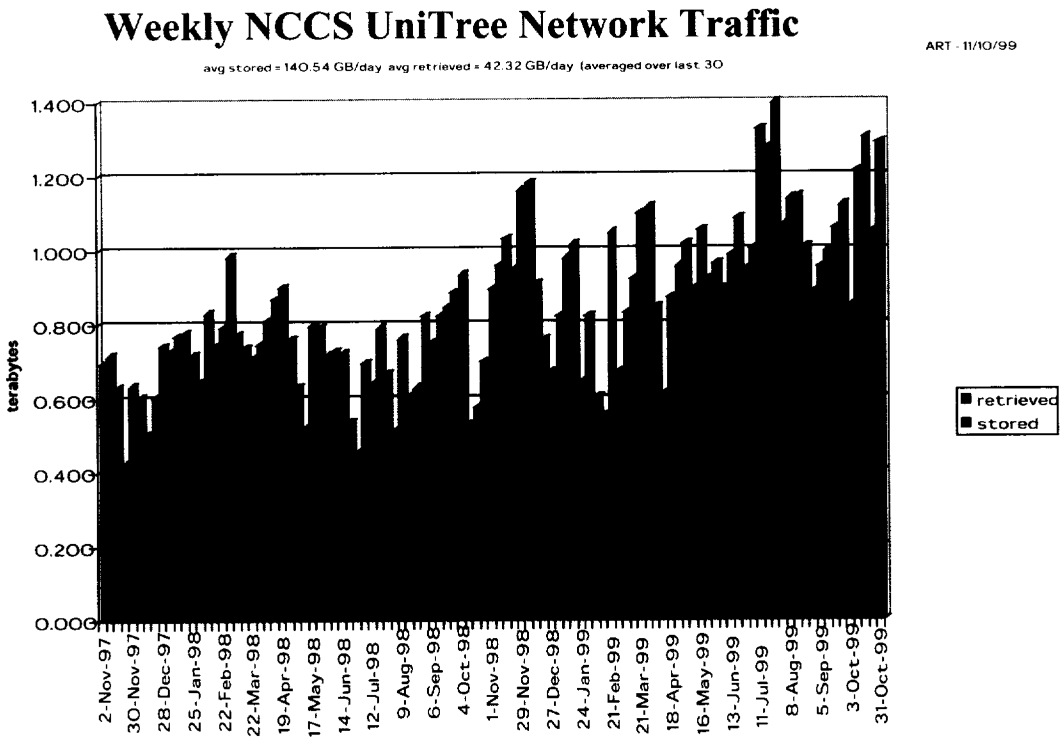


Figure 3. Weekly NCCS UniTree network traffic

The retrieval pattern of the NCCS user community is shown in figure 4 below.

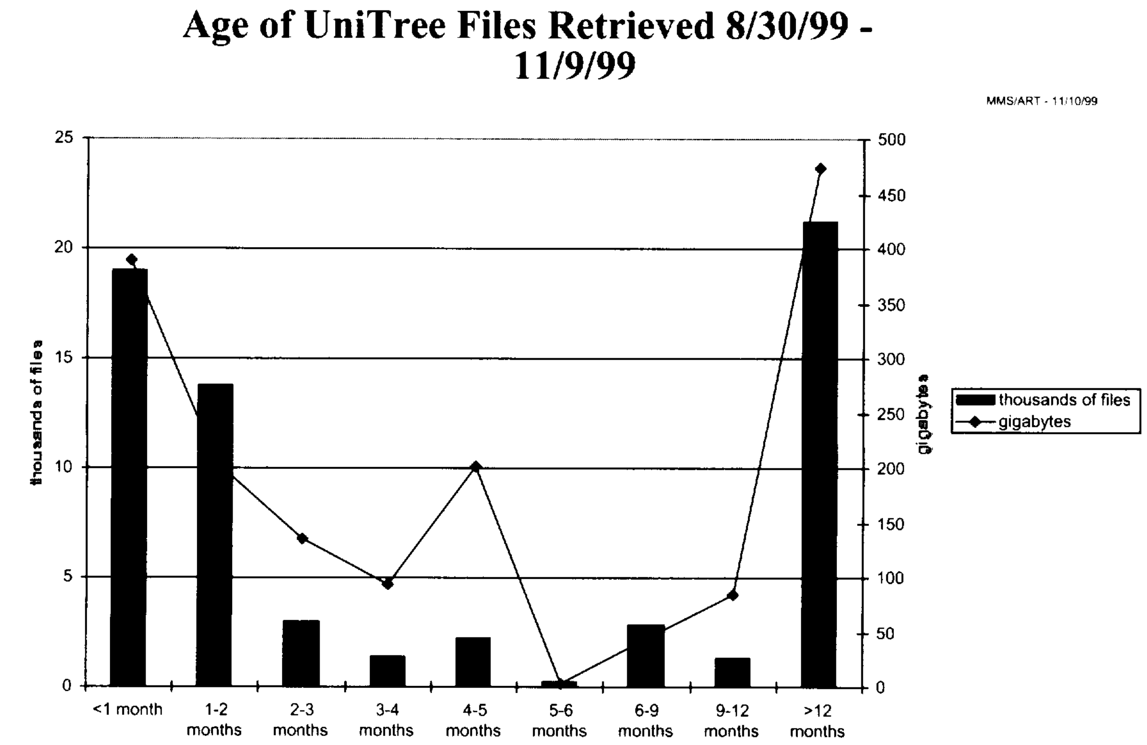


Figure 4. Age of UniTree files retrieved over a 2-month period

The early history of the NCCS UniTree+ mass storage system can be found in the proceedings of the third, fourth, and fifth Goddard conferences on mass storage [1,2,3].

2 Platform/software upgrade

Since September of 1993, the NCCS had run HP's UniTree+ mass storage software on an HP/Convex C3830 machine. By the end of 1997, however, we had decided to survey the market, seeking a newer system that might be less expensive to maintain. In the early spring of 1998 HP informed us that the C3830 was not year-2000 compliant and would not be supported past 9/30/99. Support for UniTree+ was being dropped as well, and they recommended that we convert to UniTree Software, Inc.'s (UTSI) UniTree and run it on a V-class HP machine. With the short amount of lead time we were given to find something compliant, we decided to look for an interim year-2000 solution that would be able to read our UniTree+-written tapes outright, without requiring UniTree+ to remain running as a "middleman". At the time we found only two possible software candidates: LSC's SAM-FS, and UTSI UniTree.

SAM-FS ran only on a Sun platform. UTSI UniTree was supported on HP, Sun, DEC, and SGI machines. Using provisions of NASA's SEWP contract, the four hardware vendors provided us loaner machines with which to "test drive" UTSI UniTree and (for Sun) SAM-FS. By July of 1998 we had four test platforms installed: a Sun Ultra E6000 with A3000 and A5000 RAID disk arrays, an SGI Origin2000 with a Clariion RAID fibre disk array, an HP V2250 with an EMC fibre disk array, and a DEC Alpha 4000 with a StorageWorks RAID Ultra SCSI disk array. Our intention was to test SAM-FS on the Sun E6000 and to test UTSI UniTree on all four. However, the deadline for making the final decision was mid-August. The massive effort of learning and testing two new mass storage systems and four separate flavors of Unix while simultaneously supporting the production UniTree+ mass storage system ultimately proved overwhelming for the limited staff we had. Something needed to be taken out of the equation. We therefore decided that the interim solution would be UTSI UniTree for common-sense reasons: it involved the shortest learning curve, was available on several platforms, and was successfully running at DKRZ, a site which regularly experienced more than three times our network traffic load.

With hardware platform and disk array decisions still to be made, we threw ourselves into configuring the machines and disk for optimum I/O performance. Generally, the disk vendors recommended creating a small number of large disk stripes while UniTree support recommended splitting disk arrays into as many small luns, as possible. In our I/O tests several of the disk arrays had problems handling simultaneous reads and writes to the same lun; writes would wait for the reads to finish before getting started. We concluded that this must be a configuration issue within the machine or the RAID array itself and did not allow it to affect our decision. We ultimately decided to purchase EMC disk and StorageTek Clariion disk because they were reasonably priced and able to connect to various platform types, preventing us from being locked into a particular vendor.

An important component of our testing was network-related. Our largest UniTree clients have always been the Cray systems, to which the UniTree platform has long been connected via a HiPPI switch. The switch being used in production was a Netstar with only copper interfaces, while the test machines required fiber interfaces. We had a new Gigalabs HiPPI switch waiting in the wings to be tested, so a fiber "blade" was ordered for that switch as well as a HiPPI modem for the Netstar switch. Complications arose when certain ftp retrieves over HiPPI hung; this was eventually traced to the HiPPI modem. Transfers over HiPPI that did not hang were nevertheless much slower than expected. ODS, the manufacturer of the faulty modem, decided to give us a loaner HiPPI switch, confident that we would see much greater HiPPI performance than we had in the past.

Meanwhile, our deadline for making our decision arrived, and we were forced to use the data we had at that point. Network performance was obviously inconclusive. However, Sun was the most cost-effective solution, and we were comforted by the fact that it was UTSI's primary UniTree port and that DKRZ was running UniTree on a Sun with far heavier loads than we expected at the time. We decided to purchase a Sun Ultra E6500 as our interim year-2000-compliant mass storage platform. At the same time we purchased 1.3 TB of EMC disk, 900 GB of StorageTek Clariion disk, 22 StorageTek 9840 tape drives, and 4 freestanding Timberline tape drives. Since the 9840 tapes drives were not yet available, we were to receive Timberlines to use temporarily in their place. The upgrade to 9840 tape drives will be discussed in the next section.

The new Sun E6500 arrived, and we promptly rolled up our sleeves and got to work. For our 24 IBM Magstar tape drives, we obtained both the Sun Magstar driver as well as the IBM Magstar driver. Testing quickly showed that with the Sun Magstar driver we were unable to append to previously-written tapes, making our decision to go with the IBM driver a simple one. UTSI, meanwhile, was busily developing an interface for UniTree to communicate with the IBM 3494 robotic library. When this was completed, and we had tinkered with a test system long enough to feel reasonably comfortable with how it worked, we converted the existing UniTree+ 3.0 test system we had worked with on the HP/Convex C3830. This was an excellent exercise and brought to light many issues we would have to deal with when conversion of the production UniTree+ system would occur. Among the most important was the discovery that tape format "A" written under versions of UniTree+ prior to 3.0 was actually different than the same format "A" written under UniTree+ 3.0. This discovery surprised us greatly. In the course of our attempt to upgrade to UniTree+ 3.0 months before, we had converted to and reverted from version 3.0 several times, until the major problems were fixed, and we had never seen 2.0 show any difficulty reading 3.0-formatted tapes, nor did we see 3.0 show any difficulty reading 2.0-formatted tapes. UTSI UniTree, however, needed separate tape types defined for the separate tape formats. Mark Saake of UTSI, who had made this startling discovery, proceeded to analyze all of our production tapes. He then listed for us those that had been written since the upgrade to version 3.0 as well as those that were being written at conversion and reversion times, invariably written partly in one format and partly in the other. He advised us to repack those mixed-format tapes before conversion to UTSI UniTree, advice we carefully followed.

On Wednesday January 27, 1999, we halted user activity on the UniTree+ system running on the C3830 and allowed migration to complete. We copied the databases to theSun E6500, and Mark began the conversion process. The Sun assumed the IP address of the C3830. Tape drives were uncabled from the C3830 and connected to the Sun. By about 10 PM that evening, users were permitted access. Mark remained logged into our new system through the entire night, and virtually no problems were seen. The UniTree we began running that evening was version 1.9.1 into which was backported several 2.x features such as Y2K support, support for up to 64 tape drives, and support for raw disk cache devices greater than 2 GB. Additionally, it had 3494 robotic support, which had not been previously available with UTSI UniTree.

3 Tape Drive Upgrades

9840s were a technology we had been awaiting a long time. Nearly half our UniTree files were under 1 MB in size, effectively ruling out helical drives for anything but duplicate copies. However, we were very interested in denser linear technologies. Back when StorageTek's densest linear technology was 3490E tape, storing something over 1 GB a cartridge, we had made the decision to purchase an IBM 3494 tape library with 8 Magstar drives. At 10 GB per cartridge uncompressed and 9 MB/s transfer rate, this technology served us very well. When it was available, we purchased additional Magstar tape drives in C12 cabinets and installed them in two of our StorageTek silos, which increased our silo data capacity tenfold. By the time StorageTek 9840s became available, we had 16 IBM Magstar drives in the silos and another 8 Magstar drives in the IBM 3494 robotic library, and we had been using Magstars to store all new data coming into UniTree for nearly two years with great success.

9840s had a capacity of 20 GB uncompressed, and were therefore very desirable. However, their arrival would mean the silo Magstars would have to go. The Library Management Unit microcode level that was required for the 9840s disallowed Magstar cartridges with "J" letters on them. Without "J"s on cleaning cartridges, we could not have automatic cleaning enabled in the silos, because our silos were mixed-media – they had Timberlines as well as Magstars. Disabling automatic cleaning was not an option in silos mounting hundreds of tapes a day. Since 9840s and Magstars could therefore not coexist in the same Automated Cartridge System, we had 2 choices: either to move the Magstars out of the silos entirely, or to create a separate ACS with back-level LMU microcode for the Magstar drives and install the 9840s in the other ACS. The first choice was a much better one, especially since we had a 3494 robotic library that was already home to 8 Magstar drives. Our next step was to purchase additional drive cabinets for the 3494, with the intention of moving the 16 silo Magstars into them. Sometime after this 3494 upgrade took place, however, IBM informed us that in a SCSI-connected robotic library we could only fit 16 Magstar drives total. They offered to take back the extra 8 silo Magstars, as well as the 4 silo cabinets in which they had been housed. In return, they would provide certain items that were of interest to us.

In preparation for the move of 8 silo Magstars into the 3494 robotic library, the silo Magstar tapes had been gradually transferred to the 3494. A few weeks after the UniTree conversion, the Magstar tape drives were moved out of the silos. This paved the way for the upgrades required for STK 9840 support. First the Library Management Unit microcode was upgraded, then the hands of the robots themselves, then the Automated Cartridge System Library Software that manages tape access. Finally, the 9840s themselves were installed and tested. By the end of March all new data being stored into UniTree was being written to 9840 tapes.

The following month, IBM announced their E1A product, a 256-track Magstar drive that could write double the original density to the same Magstar media. At the time of this writing, 7 out of our 16 Magstar tape drives have been upgraded from 128-track to 256-track, and data on older 3490-type media is being rewritten onto Magstars with these new drives.

4 Further upgrades

The Sun E6500 we had purchased was fully adequate for supporting our current load. However, we were told that some of our users expected their storage needs to increase considerably. To support this increase, UniTree would require more disk; however, the E6500 had almost no room for additional peripherals. Ultimately, we replaced the E6500 with an E10000 as the UniTree server. With nearly 5 times the I/O bandwidth, 4 additional CPUs, 11 additional SCSI adaptor slots, and room for 4 additional system boards, the E10000 gave us the ability to scale up to whatever the near term might require. We are successfully running UniTree on the Sun E10000 today.

At the time of the original upgrade to UTSI UniTree, 2.0 had just been released, but we were advised to upgrade to the relatively stable 1.9.1 instead, with support for y2k, 64 tape drives, and raw disk partitions greater than 2 GB backported into our version. UniTree 2.1, however, offered considerable performance increases as well as the ability to use up to 256 disk cache partitions, where 1.9.1 allowed for only 110. Our first attempt at upgrading to 2.1 uncovered some bugs which were quickly fixed. Our second attempt shortly thereafter uncovered a Solaris bug which UTSI was eventually able to work around. On Monday August 30 we successfully upgraded to 2.1 and never looked back.

In the past year, some of our users have informed us that their data requirements are expected to increase drastically over the next several years. To accommodate this anticipated increase we decided to purchase more disk for UniTree. After testing several brands and reviewing offers from several vendors, we purchased an additional 4 TB of 10,000 RPM disk from EMC. Adding this new disk to UniTree's cache will require the repartitioning of the EMC disk we are currently using, since UniTree at present can use only 256 disk cache partitions.

5 Future Work

Within the next several months (as of this writing) 4.5 TB of High Performance Computing data, currently stored under DMF on a Cray T3E, are expected to be moved into UniTree using a DMF FTP Media Specific Process (MSP). This will allow current T3E DMF users to continue retrieving their files transparently from DMF during and after the move. The MSP defaults to one flat UniTree directory per DMF user, which is causing us some concern. There will be some very large UniTree directories created in this fashion, one of which is expected to contain over 200,000 files. Testing of this situation and how it might affect UniTree performance for other users is ongoing, and namesrvr tuning advice is being solicited from UniTree support. The silo on which these DMF files have been stored, along with its 4 Redwood drives, will be attached to the existing remote silo to add to UniTree's duplicate-copy capacity.

Also ongoing is the effort to duplicate existing UniTree data in our remote silo. Duplication of new data coming into UniTree has been automatic since November 1997. As of this writing 33.2 TB of older UniTree data remains to be duplicated. A UniTree 2.1 facility allows an existing file to be marked "dirty" so that it is rewritten to tape and a second copy generated at the same time. This will have the added benefit of rewriting data on older 3490-type media to new 20 GB media.

In the long term, we plan to do a more thorough survey of the HSM market, the survey we would have done had Y2K not become a pressing deadline. Many new HSM products have come to market within the past few years, and we are interested in keeping abreast of these new developments.

References

[1] A. Tarshish and E. Salmon, "The Growth of the UniTree Mass Storage System at the NASA Center for Computational Sciences," *Proceedings of the Third Goddard Conference on Mass Storage Systems and Technologies*, (1993) 179-185.

[2] A. Tarshish and E. Salmon, "The Growth of the UniTree Mass Storage System at the NASA Center for Computational Sciences: Some Lessons Learned," *Proceedings of the Fourth Goddard Conference on Mass Storage Systems and Technologies*, (1995) 345-357.

[3] E. Salmon, "Storage and Network Bandwidth Requirements Through the Year 2000 for the NASA Center for Computational Sciences," *Proceedings of the Fifth Goddard Conference on Mass Storage Systems and Technologies*, (1996) 273-286.

Data Volume Proliferation in the 21st Century
The Challenges Faced by the NOAA National Data Centers (NNDC)

John Jensen
National Climatic Data Center (NCDC)

John Kinsfather
National Geophysical Data Center (NGDC)

Parmesh Dwivedi
National Oceanographic Data Center (NODC)

National Oceanic and Atmospheric Administration (NOAA)
National Environmental Satellite and Information Services (NESDIS)

National Climatic Data Center (NCDC)
Federal Building
151 Patton Avenue
Asheville, NC 28801-5001
jjensen@ncdc.noaa.gov
+1-828-271-4680
+1-828-271-4876

National Geophysical Data Center (NGDC)
325 Broadway
Boulder, CO 80303-3328
jkinsfather@ngdc.noaa.gov
+1-303-497-6404
+1-303-497-6513

National Oceanographic Data Center
1315 East West Highway, Room 4113
Silver Spring, MD 20910
pdwivedi@nodc.noaa.gov
+1-301-713-3284
+1-301-713-3301

ABSTRACT

Data Volume Proliferation in the 21st Century
The Challenges Faced by the NOAA National Data Centers (NNDC)

This paper describes the challenges facing the three NOAA National Data Centers (NNDC), as well as the Information Technology Storage Area Network Systems (SANS) and Telecommunications industries in successfully meeting one component of the NOAA mission, Long Term Stewardship (Archiving and Access) of environmental data in the 21st Century.

The collective holdings of digital data for the three data centers are approaching 800TeraBytes (TB), and steadily growing by several hundred terabytes of new data each year. By the year 2001, it is anticipated that the NNDC will be ingesting and processing over two hundred TeraBytes (TB) of new data each year, while managing and providing access to over one PetaByte (PB) of data and information. There is an immediate need to explore what Information Technologies (IT) and Data Handling Techniques will be available to successfully meet the challenges regarding ingest, processing, convenient access, and efficient mass storage of very large volumes of environmental data.

Early in the last decade of the Century/Millennium, it was clear that developments in Information Technologies (IT) would place powerful tools in the hands of most everyone, which could process and display large volumes of data at a relatively small cost. By 1995, NOAA developed a strategic vision and embarked on a program to build a dynamic and responsive IT architecture that would provide worldwide electronic access to the enormous volumes of climatic, geophysical, and oceanographic data under the long term stewardship of the three NOAA National Data Centers (NNDC).

Electronic access and mass storage issues required examination from a total enterprise systems perspective taking into account connectivity between the data collection platforms, the processing and storage facilities, and users anywhere in the world. In 1996, the three centers, under the leadership of the National Environmental Satellite and Information Services (NESDIS) office, embarked on an initial five-year plan to address these issues and meet the challenges of the 21st Century. This plan is referred to as the NOAA Virtual Data System (NVDS) Initiative. At that time, the cumulative digital holdings were about 300TB, and it was projected that by 2000 the total volume of new data would increase to over one hundred terabytes per year. By 1998, the annual volume of new data exceeded the projected Year 2000 figure. The volume of new data per year for 2001 is currently projected to be over two hundred terabytes per year. If history is any indicator, this projection is conservative.

How will NOAA successfully achieve a solution for the end-to-end stewardship of the proliferation of data volume and variety that will be a reality in the very near future?

Data Volume Proliferation in the 21st Century
The Challenges Faced by the NOAA National Data Centers (NNDC)

Introduction

Climatic, Geophysical, and Oceanographic observations, data, and information are growing at a rate exceeding any projections five years ago. Today, the NOAA National Data Centers (NNDC) provide stewardship (storage and access) to a data volume that increases each year by more than the equivalent all the data managed by the centers over the past 100 years! At the same time, the phenomenal successes of the World Wide Web and Internet have resulted in the worldwide diversification and expansion of the user community by over two orders of magnitude in just the past five years. The data volume curve turns sharply upward in the first decade of the 21st Century. These growth projections are probably conservative. The central issue for the NOAA, in particular the three data centers, is how to respond to the challenges presented by the unprecedented proliferation of environmental observations and data. This paper will describe the NOAA National Data Centers mission, current capabilities, and vision for data management and access. It seeks to educate decision-makers and the Information Technology (IT) industry about the critical and urgent requirement for solutions.

Description of NOAA National Data Centers (NNDC)

The three geographically dispersed NOAA National Data Centers (Figure 1) are designated as official sites for the long-term stewardship of climatic, geophysical, and oceanographic data. The mission of these data centers is to preserve and provide access to these data. The data centers have existed in one form and place or another for many decades. Data are collected by a variety of observing systems and types of instruments, both in-situ earth-based and remote satellite-based. Satellite data represents the overwhelming majority of digital data. These observing systems are operated by different line offices within the National Oceanic and Atmospheric Administration (NOAA), as well as many other agencies, such as the U.S. Geological Survey (USGS), U.S. Department of Agriculture (USDA), Federal Aviation Administration (FAA), the Department of Defense (DoD), and others. Data is also obtained from observing systems operated in foreign countries through bilateral and multi-lateral agreements and other arrangements negotiated through the auspices of United Nations sponsored agencies, such as the World Meteorological Organization (WMO) and the International Council of Scientific Unions (ICSU). Each of the data centers is also a designated World Data Center (WDC) for their respective data types. This further enhances the exchange and availability of worldwide data.

The National Climatic Data Center (NCDC), Asheville, NC, is responsible for the Nation's climate data. Currently, the NCDC manages and provides access to about 220 million paper records (equating to about 38,878 miles – five times around the world), 125,130 rolls of 35 mm and 16mm film (equal to the distance between Washington DC and Los Angles, 2,340 miles), 1.2 million pages of microfilm (equal to the distance between Washington DC and Philadelphia, 114 miles), and about 800 TeraBytes (TB) of digitally stored data on about 600,000 3480, 3590, and 8mm Exabyte magnetic tape cartridges (would fill a million CD ROMs which stacked vertically would be the height of five Empire State Buildings). The 3480 and 8mm tapes are located in Asheville, NC and at the University of Wisconsin (GOES satellite data only). The NCDC has a robotic storage system. In most cases, the observing networks are reasonably well defined and managed, and data collection, formats, and reporting are coordinated. Figure 2 summarizes digital data growth for the period 1989-1999.

The National Geophysical Data Center (NGDC), Boulder, CO, is responsible for a wide variety of specific data categories, such as Solar, Tectonic/Earthquake, Volcanic, Magnetic, Gravitational, and other geophysical specific areas of study. The distinctive division of data into different classes of physical sciences makes the NGDC more unique. NGDC data are stored on paper, microfilm, and digitally on magnetic tapes. Data collection and reporting comes from a much smaller (spatial and temporal) network of data unique observing systems operated by a variety of agencies, both U.S. and foreign. Currently, NGDC manages 400 different data sets and about 15TB of digital data on 3480 and 8mm Exabyte tapes. Satellite data from the Defense Meteorological Satellite Program (DMSP) make up the majority of the digital data holdings. Figure 3 summarizes digital data growth for the period 1988 to 1999.

The National Oceanographic Data Center (NODC), Silver Spring, MD, is responsible for oceanographic data. NODC faces the most difficult challenge relative to the acquisition of oceanographic data and perhaps also the largest variation in how the data is collected and recorded. There are few coordinated and well-defined oceanographic observing networks. In general, oceanographic data are collected by many different individual organizations, such as universities, federal agencies {DoD, NOAA, Mines and Minerals Services (MMS), Environmental Protection Agency (EPA), U.S. Coast Guard (USGS), U.S. Corps of Engineers (COE), etc.}, coastal state agencies, and commercial companies, in particular oil and gas. These data are often not reported for months or years later due to the desire to keep the data secure until research papers and other reports are ready for publication. In other cases the information may be classified and never shared, particularly from oil and gas companies. The data format and methods of recording and reporting the data varies greatly from paper to unique digital recorders. There is a tremendous volume of data that has been collected but gathering the data and formatting it for placement into a digital data base are significant issues for NODC operations. Nearly 75% of the current total volume of digital data, 1.5 TB, managed by the NODC is satellite data. Figure 4 summarizes digital data growth for the period 1990 to 1999.

Current Capabilities

Early in this decade, it was clear that developments in Information Technologies (IT) would place powerful tools in the hands of most everyone, which could process and display large volumes of data at a relatively small cost. By 1995, NOAA embarked on a program to develop a dynamic and responsive IT architecture that would permit worldwide electronic access to the enormous volumes of climatic, geophysical, and oceanographic data.

It was obvious that electronic access and mass storage issues needed to be addressed from an enterprise level perspective for data Ingest, Storage, and Access, to include a complete review of connectivity between the data collection and recording sites, the data centers, and the user anywhere in the world. In 1996, the three Centers, under the leadership of the National Environmental Satellite and Information Services (NESDIS), embarked on a five-year plan referred to as the NOAA Virtual Data System (NVDS) Initiative. Significant capital investment has been made to the three centers' IT infrastructure. Much more needs to be done to sustain the progress made in the first four years of the initiative, if the NOAA is to successfully meet the challenges in the first decade of this new millennium. The NNDC goal is to provide easy, convenient, and timely access to large volumes of the Nation's environmental data and information.

The first two areas addressed were the storage and access components of the enterprise system. A common IT architecture and "look" to the customer are essential elements of the design. Flexibility within the strategic IT architecture plan is paramount in order to capitalize on new IT developments. Acquiring a hierarchical mass storage system that provided direct electronic access to data was a priority. Currently, only the NCDC has a mass storage robotics system, IBM 3494, with HPSS software resident on one of the IBM Scalable PowerParallel (SP2) System nodes. The concept of an On-Line Data Store consisting of a web farm with large storage disk devices has been implemented. The Oracle RDBMS was selected to support the On-Line Store. All three data centers are linked to each other. A single easy to use access and comprehensive ordering system permits browsing, viewing, ordering, and transfer of data while on-line. Populating the On-Line Store System with on-line digital data will continue indefinitely. The ultimate goal is many tens of terabytes placed in an on-line status and the balance of holdings near on-line. Multiple paths to access and browse the on-line data have been integrated into the design to service the sophisticated and the uninitiated users. The NNDC Climate Data On-Line (NNDC CDO) System is now operational and can be accessed by going to: www.nndc.noaa.gov.

Currently, there are several dozens of data sets, products, and other information, about 120 GigaBytes (GB), available through the NNDC CDO System. Direct electronic access to on-line and near on-line data (robotic system) and timely transfer of large volumes of data are now at the fingertips of a worldwide clientele. The net result for the customers will be exceptional service, while the data centers will be able to respond to increasing volumes of new data and greater customer demands in a cost effective manner.

The Data Explosion Challenge

Table 1 and the accompanying graph, Figure 5, provide an overview of the projected volumes of data that are expected to be delivered to the data centers during the first decade of the 21st Century. It is quite apparent that the satellite data will be the largest contributor to unprecedented data growth. The NEXt generation RADar (NEXRAD) network is now in place and this volume should remain relatively constant. However, NEXRAD data does constitute a fair level of total digital volume. The vast majority of the new data volume will be directed to the National Climatic Data Center. All new data must be delivered digitally and immediately placed into the current mass storage system.

In order to efficiently manage and access these large volumes, the centers must be able to create the requisite inventories, place the data immediately into a robotic style mass storage system, and retrieve the data in a relatively straightforward and timely manner.

Another significant challenge is the migration the digital data currently on the 600,000+ off-line 3480 and 8mm tapes into a robotic mass storage system. A multi-year data modernization effort is addressing some of the non-digital data (paper, microfilm, and microfiche). Some of the paper and microfilm records are being optically scanned and the images placed on CD ROM. Eventually these files will need to be place into a robotic system or some other system that permits direct and convenient electronic access and transfer. Some of the data on these records are being manually keyed into digital data bases. In addition to these activities, significant portions of the current and future digital data are periodically "reprocessed" when new mathematical algorithms are developed which can improve the utility and value of earlier data. Therefore, as the volume of data grows, the level of reprocessing efforts will grow in kind.

The centers typically adhere to National Archives and Records Administration (NARA) policies and guidelines. Long term stewardship and access to these data and information must be guaranteed for future generations of customers. If long term stewardship in perpetuity is required, then all these accumulated data and information will have to be migrated to new systems, perhaps as often as every five years, due to rapid changes in the information technology environment.

Meeting the Challenge

Four key areas that must be addressed by the NOAA and the commercial IT marketplace in order to meet the demands of the 21st Century: 1) Network Centric Planning, 2) Highly Automated Mass Storage, 3) Large Volume Storage Media and Data Compression (particularly for satellite data), and 4) Rapid and Convenient Data Access.

Network Centric Planning will contribute to one of the goals of modernization, the digital delivery of data and information from the observing systems to the centers and ultimately to the customers. There have been a number of successes resulting from the practical application of concepts associated with the Network Centric Operations and utilizing available communications and in-place IT systems and capabilities. Over the past few years, the data centers in cooperation with other offices within NESDIS, the National Weather Service (NWS), and the Department of Defense have transitioned from paper forms, diskettes, and magnetic tapes to staging data on a daily basis and using ftp transfer procedures. Examples include digital transfer of DMSP data to NGDC, as well as data transfers from NWS activities to NCDC to include Marine, Surface Hourly (ASOS, AWOS), and global CLIMAT observations. In some cases, these data are posted daily to the web pages for access by a worldwide clientele. The results have been more data received and processed, improved up front quality control, and data available earlier to customers, i.e. hours and days as compared to days, weeks, and months.

There are several areas, which require a focused effort to complete the transition for other established observation networks. These include the 8,000 paper forms and 2,400 punched paper tapes from the COOPerative Observation Network, the 3590 magnetic tapes used to deliver GOES and POES satellite data, and the 8mm and optical disks used to deliver the NEXRAD data. Interactive Voice Recognition (IVR) and Natural Language (NL) can, for a small initial investment, replace the 8,000 COOP paper forms while substantially reducing annual operating costs. Data would be available via the web on a daily basis rather than the current schedule, 45-60 days after the end of a data month. Similarly, a digital recorder can replace the punched paper tapes. Beginning in early CY 2000, all the POES satellite data will be digitally transferred on a daily basis from Suitland, MD directly into the robotic system at Asheville, NC. Daily digital data transfer of GOES satellite data between the University of Wisconsin and the NCDC will begin when the cost-benefit ratio between telecommunications and 3590 tapes becomes more attractive. NESDIS and NWS are currently examining a Network Centric approach to transmitting radar level II data via telecommunications from each radar site to a central data collection facility. This facility will assemble packets of radar data and information, which will then be digitally transmitted to the NCDC on a daily basis. Radar level III data will be transmitted nationwide via the NWS AWIPS/NOAAPort satellite communications system. In the case of the GOES and NEXRAD, the life cycle cost of handling tapes warrants replacing tapes with near real time digital data transfers as soon as practical.

Table 1 reveals that ingest, inventory, storage, and access tasks for the emerging satellite systems (METOP, NPP/NPOES, EOS, and growth in DMSP and GOES) present a far different challenge. Based on past performance, it is reasonable to expect telecommunications bandwidth and associated costs will permit direct data transfer to the robotic mass storage systems. However, the ability to inventory, transfer, and access these data using highly automated mass storage devices and techniques require considerable planning and IT advances. The NCDC procured the IBM 3494 system in 1994-95 and upgraded from UNITREE to HPSS in 1998-99. The NCDC also installed the first nodes of the SP2 system in 1998 and currently use two nodes to manage file transfer activities into and out of the IBM 3494. Many of the performance, staging, inventory, and retrieval issues discussed during the March 1999 Mass Storage Conference in San Diego remain serious concerns for the data centers.

A Highly Automated Mass Storage System provides a solution to another goal of modernization, efficient long-term stewardship of enormous volumes of data and information. These new systems must reduce the cost per Terabyte of storage, provide reasonable write/read throughput, and large data storage with a small physical "foot print." This presents a bigger challenge than telecommunications capabilities and costs. In the short and near term, next 1-3 years, the NCDC will install additional SP2 nodes and upgrading the current 100Mbps to a higher performance local area network (LAN) will be required by FY 2001. New mass storage systems must provide Large Volume Storage Media and Data Compression features. In FY 00, the NCDC must begin the upgrade to 3590E drives and utilize the new 3590Extended (40GB) tapes. However, even though the IBM 3494 is scalable and will have improved capacity 3590E drives and tapes, the IBM 3494 is not envisioned as the system to manage the high volume satellite data. Therefore, a higher capacity, higher throughput performance mass storage system must be identified and procured about the year 2003-2004. It is envisioned that the existing IBM 3494 and the new higher capacity (less cost per TB and smaller footprint) system will be utilized concurrently to perhaps the year 2010. Use of straightforward data compression techniques must be a consideration in the design of any future mass storage system (magnetic or optical tapes, disk, other). Hardware data compression is now in use on a number of the different types of data, but not satellite data. This will help in the short term to control the growth of the current IBM mass storage system. "No Loss" data compression techniques for satellite data need to be available very soon.

Neither of the other two data centers have a robotic mass storage system to support their current or future needs. The NGDC has the most immediate requirement for a modest automated mass storage system to manage (new and migrate existing) the DMSP and other digital data currently stored off-line on 8mm Exabyte (about 6,800) and 3480 (about 5,200) tapes, respectively. Similarly, the NODC will need a modest mass storage system.

Another goal of modernization to be achieved by 2010 is to have all digital data resident on automated mass storage systems. Only then can these systems and associated software be utilized to migrate in a highly automated fashion the data to the next generation mass storage system beyond 2010. If this goal is not achieved, it may be impossible from a cost and perhaps hardware availability point of view to "rescue" the data (mostly satellite and NEXRAD) stored on off-line tapes from the last half of the 20th Century. Perhaps magnetic tapes will no longer be the preferred, cost effective, media for storage and access in the second decade.

Rapid and Convenient Access to data and information by a worldwide clientele is required if NOAA and the data centers are to succeed in their information services mission. Progress in this area may prove to be the greatest challenge. Electronic commerce developments in the marketplace are providing many of the solutions to customer servicing. The operational implementation of the NNDC Climate Data On-Line (NNDC CDO) System is the first step toward providing rapid and direct access to data and information. The NNDC Home Page and the NOAA Server Home Page provide access, browse, and data transfer capabilities to data under the stewardship of the NOAA. These are interconnected and can pass a customer along to one of the three data centers or other locations based on the type of data being requested. However, placing data into web based systems with real on-line disk access and near on-line mass storage systems only minimally satisfies user requirements. Many of the NOAA clients will require electronic services that allow users to discover, subset, retrieve, overlay, visualize, and analyze data from different data bases and formats. This presents a challenge for the data centers and the NOAA because historically data has been managed primarily on the basis of the characteristics of the observing system. Future success will require the creation and adoption of industry standards and conventions to allow heterogeneous systems and data bases to communicate with one another. Secondly, it will require automated staging of communications and computational resources to execute required applications on the data. Data discovery and retrieval using an information system environment must to be done without the user having to understand the particulars of each individual system. Transaction processing must be implemented that enables an essentially "hands-off" operation requiring little or no human handling or transport of data and where appropriate the system will allow users to pay for data or services through credit and debit cards or automated billing.

Good News

There is an understanding within the NOAA and other agencies that cooperation and collaboration are critical to meeting mission goals and objectives. There are considerable discussions and interest regarding the concept of Network Centric Planning. More than any time in the past, line offices within the NOAA, particularly the NWS and NESDIS, are meeting to discuss the requirements for data transmission over the AWIPS and NOAAPort system, as well as the future directions for NEXRAD data transmission and storage and modernizing the Cooperative Observing Network. These discussions must also take into account communication needs to activities and data bases widely dispersed at many other of the NOAA sites operated by the Fisheries, Ocean Services, and Office of Atmospheric Research line offices, both on land and at sea. The NOAA High Performance Computing and Communications (HPCC) Program Office is leading the way in bringing these different offices together and defining a Network Centric approach to data access and delivery.

Most recently, NOAA and NASA representatives have been meeting to design a Long Term Archiving and Servicing Plan for the Earth Observing System (EOS) to include joint archiving and servicing activities. Similar planning is underway for the METOP and NPP/NPOES programs. The intent is to build on the experiences and lessons learned from the current digital POES transfer project and apply these to the GOES, METOP, and then NPOES which are similar in nature, just a larger daily volume.

The National Environmental Data Archive and Access System (NEDAAS) is being promoted as the successor to the NVDS Initiative. NEDAAS will build on the progress made under NVDS and other information systems modernization efforts within the NOAA, as well as other agencies. NEDAAS is the next step in providing the NOAA the means and capabilities to meet the data management tasks in the first decade of this millennium. A key component will be the ability to access, merge, and visually display multidimensional data and information. The vision is a suite of information services linking observing systems and customers to many locations, such as the NOAA National Data Centers, State and Regional Climate Centers, NOAA Laboratories (i.e. National Hurricane Center, Severe Storms Laboratory, Pacific Marine Environmental Laboratory, Coastal Science Center, etc.), and other university and research facilities. The goal is to provide rapid access to new and historical data and information either on-line or near on-line through the use of the Next Generation telecommunications and other information technologies. NOAA will be a partner in defining and capitalizing on new technologies and capabilities.

Conclusions

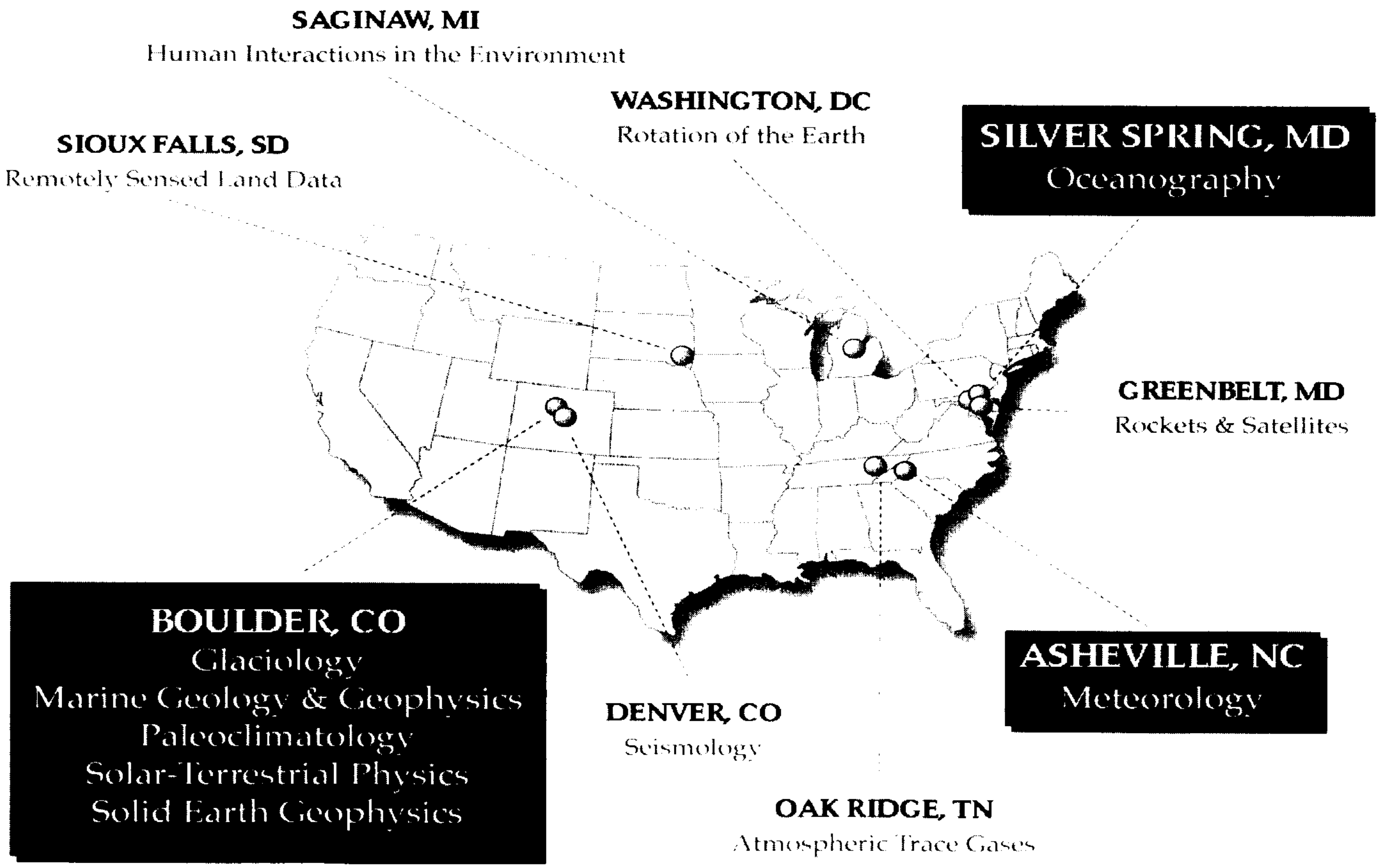
The NOAA is at a critical juncture with respect to its information services mission. Preserving the Nation's environmental records now means managing a data volume that increases in one year by the equivalent all the data NOAA has managed over the past 100 years! At the same time, the World Wide Web and the Internet have fuelled the explosion of users by over two orders of magnitude in just the past five years. Today's worldwide clientele demand rapid and convenient access to data and information. The confluence of information management technologies with observation technologies must provide the capabilities to respond effectively to the massive array of new data and satisfy a far larger, more demanding, more sophisticated, and more diverse user community. The NOAA faces a unique and multi-facet task. First, it must be able to process and store very large volumes of new data. It must provide customers rapid access to data and information, which means getting data from observing networks in near real time, rapid processing, and placing large volumes of data on-line. The enormous volume of digital data stored off-line must be migrated to a near on-line mass storage system. Computational systems capable of rapidly reprocessing large data arrays must be available if there is to be any value to aging data and information. Finally, data and information recorded on non-digital media (paper and film) require migration to a digital format, if only optical images for preservation and digital access. Incremental investment in technology designed to integrate new technology advances with the existing IT infrastructure requires strategic planning and an associated supporting budget process.

It is essential that the NOAA clearly articulate to industry and government leaders the magnitude of the challenge. The capacity and complexity of a future integrated mass storage, data processing and reprocessing, and data access system seem overwhelming. However, 30 years ago these same issues appeared to be considerable when the first satellites were being launched.

The question then: How to manage tens or several hundreds of MegaBytes?

The question today and tomorrow: How to manage PetaBytes and then YottaBytes?

WORLD DATA CENTER-A LOCATIONS



SAGINAW, MI
Human Interactions in the Environment

SIOUX FALLS, SD
Remotely Sensed Land Data

WASHINGTON, DC
Rotation of the Earth

SILVER SPRING, MD
Oceanography

GREENBELT, MD
Rockets & Satellites

BOULDER, CO
Glaciology
Marine Geology & Geophysics
Paleoclimatology
Solar-Terrestrial Physics
Solid Earth Geophysics

DENVER, CO
Seismology

OAK RIDGE, TN
Atmospheric Trace Gases

ASHEVILLE, NC
Meteorology

Figure 1

Quantity of Digital Data in the NCDC Archive

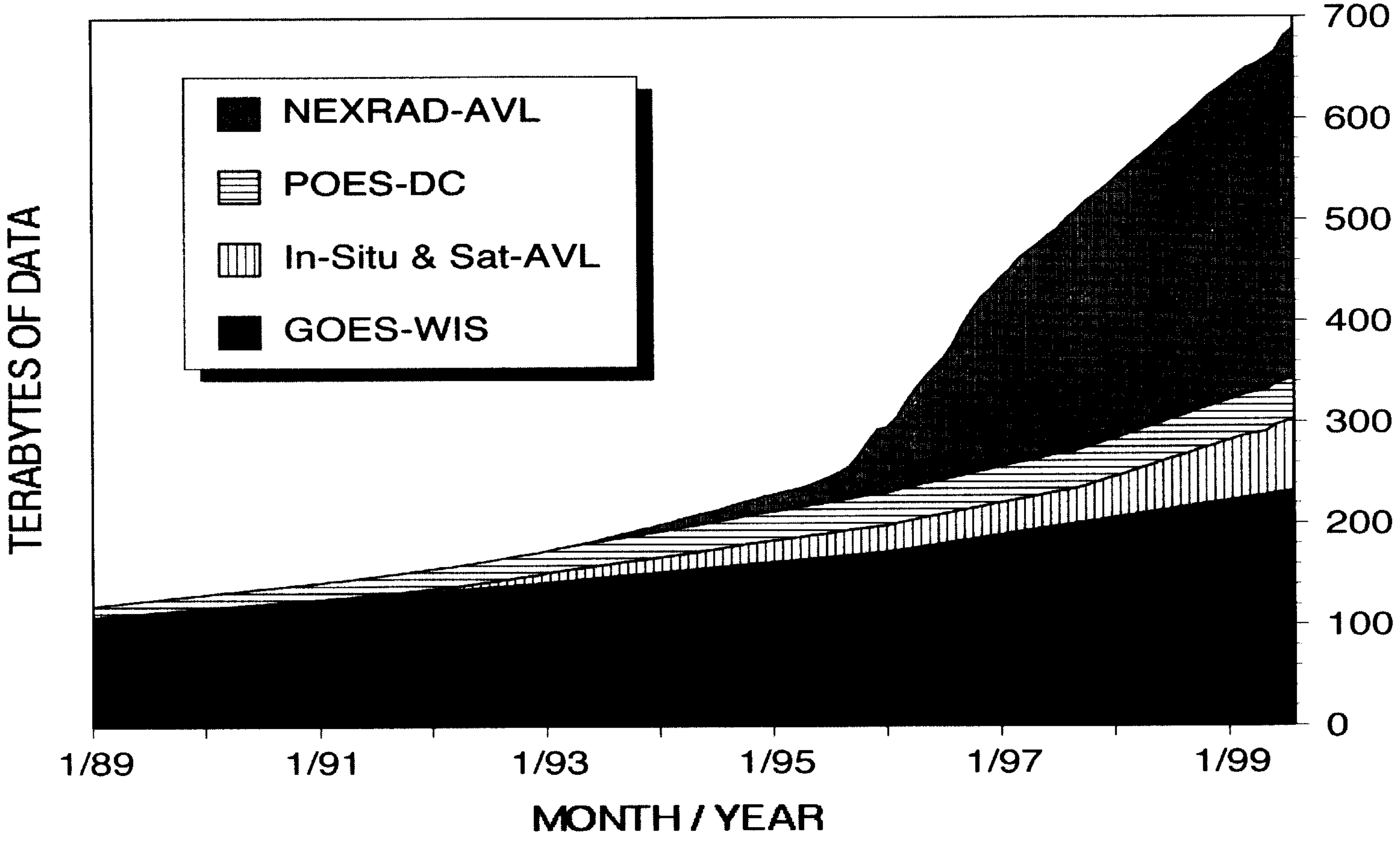


Figure 2

Quantity of Digital Data in the NGDC Archive

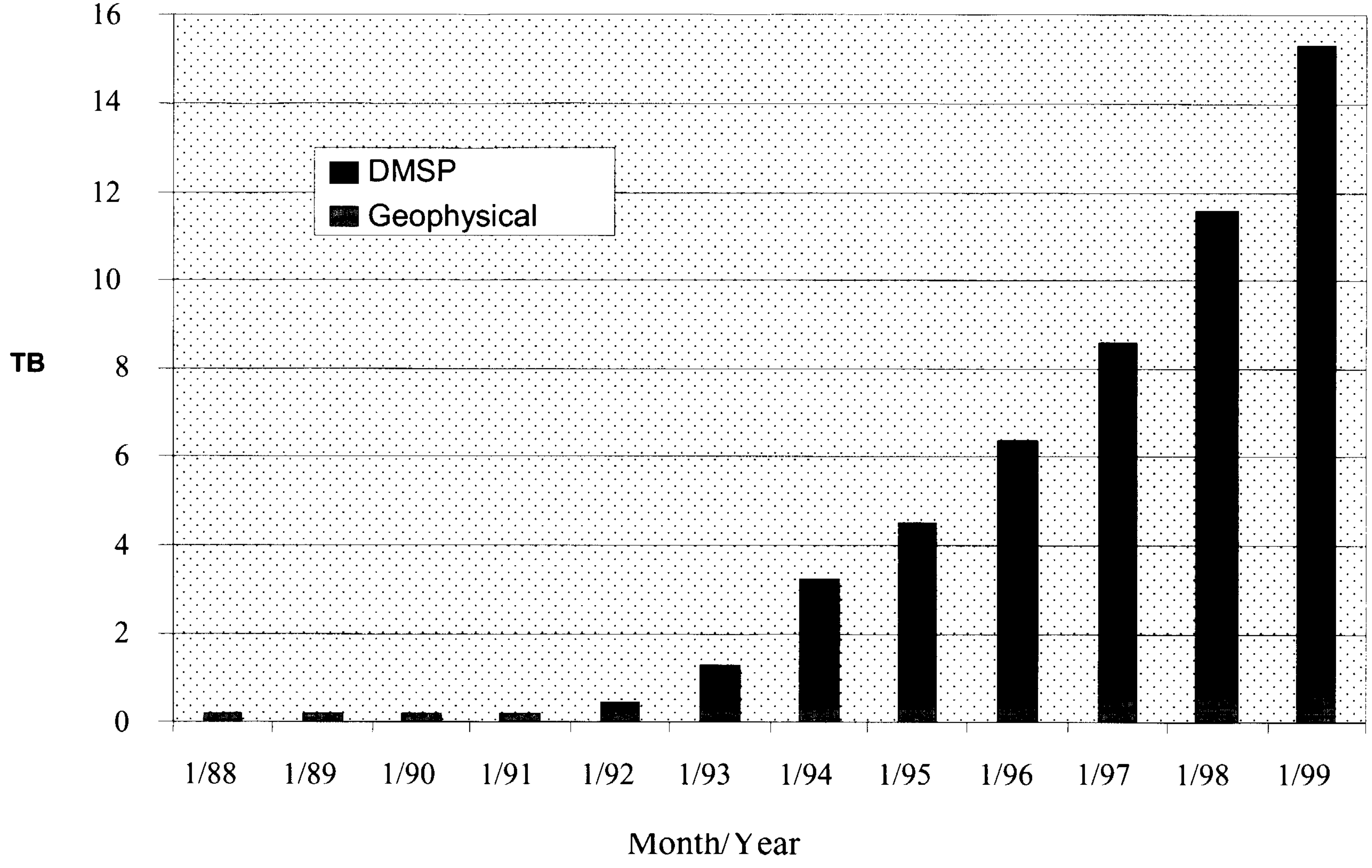


Figure 3

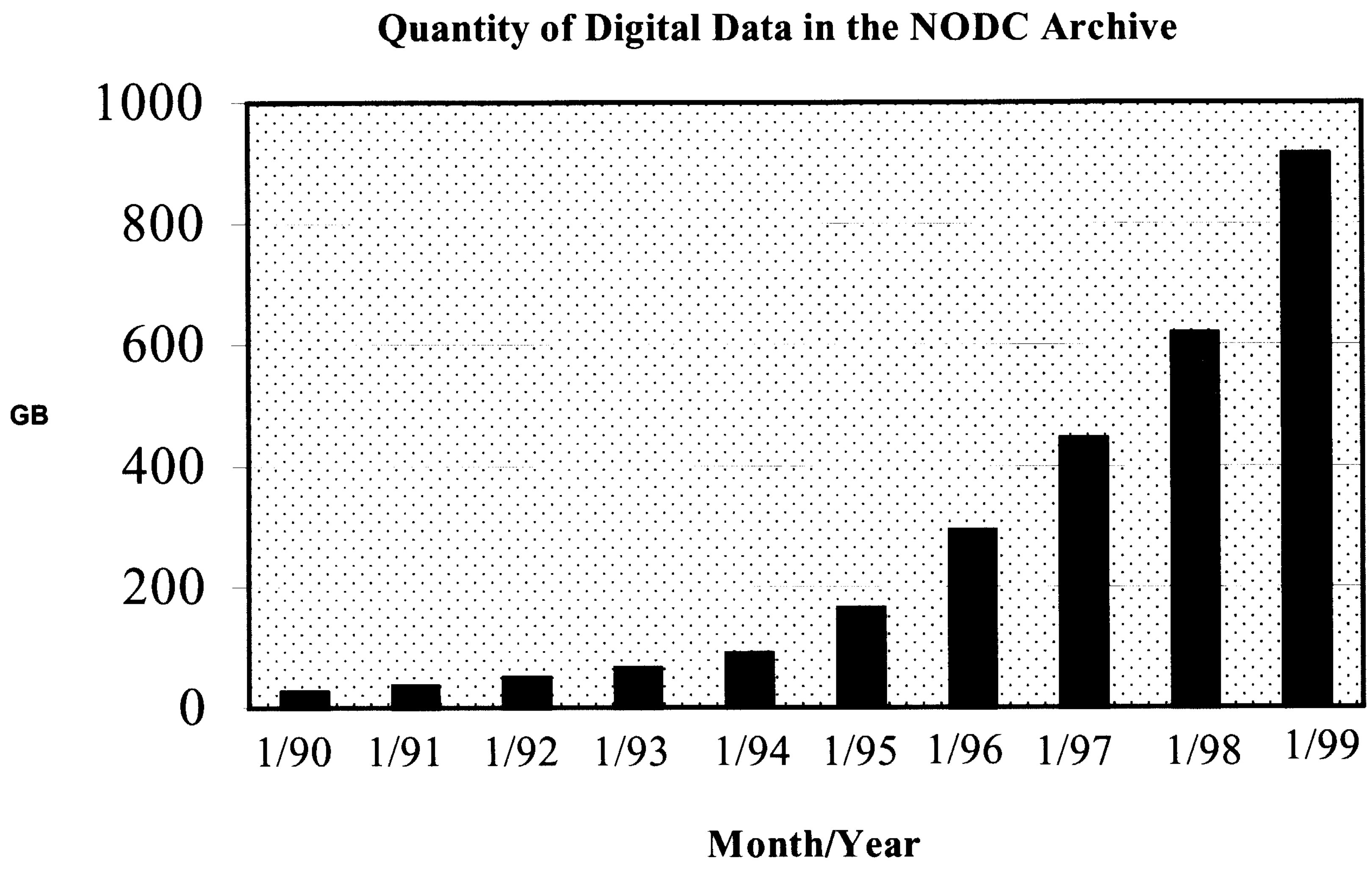

Quantity of Digital Data in the NODC Archive
GB
1000
800
600
400
200
0
1/90
1/91
1/92
1/93
1/94
1/95
1/96
1/97
1/98
1/99
Month/Year

Figure 4

Satellite, Radar (NEXRAD), and In-Situ Observations (ASOS, AWOS, Marine, COOP, Radiosonde, Solar Radiation, Etc.)

(TeraBytes)

	2000	2001	2002	2003	2004	2005	2006	2007	2008	2009	2010	2011	2012	2013	2014
EOS	0	1.46	196.52	379.91	636.76	639.75	640.56	640.48	469.48	274.66	23.92	25.2	24.92	24.89	22.73
NPP/TBD	0	0	0	0	0	173	173	173	173	173	173	173	173	173	173
NPOESS	0	0	0	0	0	0	0	0	228	228	456	456	456	456	456
METOP	0	0	0	28	28	28	28	28	28	28	28	28	28	28	28
POES	2	2	2	2	2	2	2	2	0	0	0	0	0	0	0
GOES	36	36	36	36	36	36	36	36	36	36	72	150	300	300	300
NEXRAD	120	120	120	120	120	120	120	120	120	120	120	120	120	120	120
In situ	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
NCDC Total Addition	176	178.46	374.52	586.91	844.76	1,021.75	1,023.56	1,024.48	1,080.48	886.66	900.92	981.20	1,131.92	1,132.89	1131.73
NGDC Total Addition	6.88	27	27	90.5	90.5	146	146	146	146	146	146	111	111	55.0	55.5
NODC Total Addition	0.72	0.65	0.65	0.65	0.68	0.68	0.66	0.71	0.70	0.70	0.72	0.74	0.74	0.76	0.76
Cum Total Addition	183.60	206.11	402.17	678.06	935.94	1,168.43	1,170.22	1,171.19	1,227.18	1,033.36	1,047.64	1,092.94	1,243.66	1,189.05	1,187.99
Previous Year Total	763.15	946.75	1,152.86	1,555.03	2,233.09	3,169.03	4,337.46	5,507.68	6,678.87	7,906.05	8,939.41	9,987.05	11,080.0	12,323.7	13,512.7
Cum Total	946.75	1,152.86	1,555.03	2,233.09	3,169.03	4,337.46	5,507.68	6,678.87	7,906.05	8,939.41	9,987.05	11,080.0	12,323.7	13,512.7	14,700.7

Projected Data Volume Delivered to the NOAA National Data Centers (NNDC)

TABLE 1

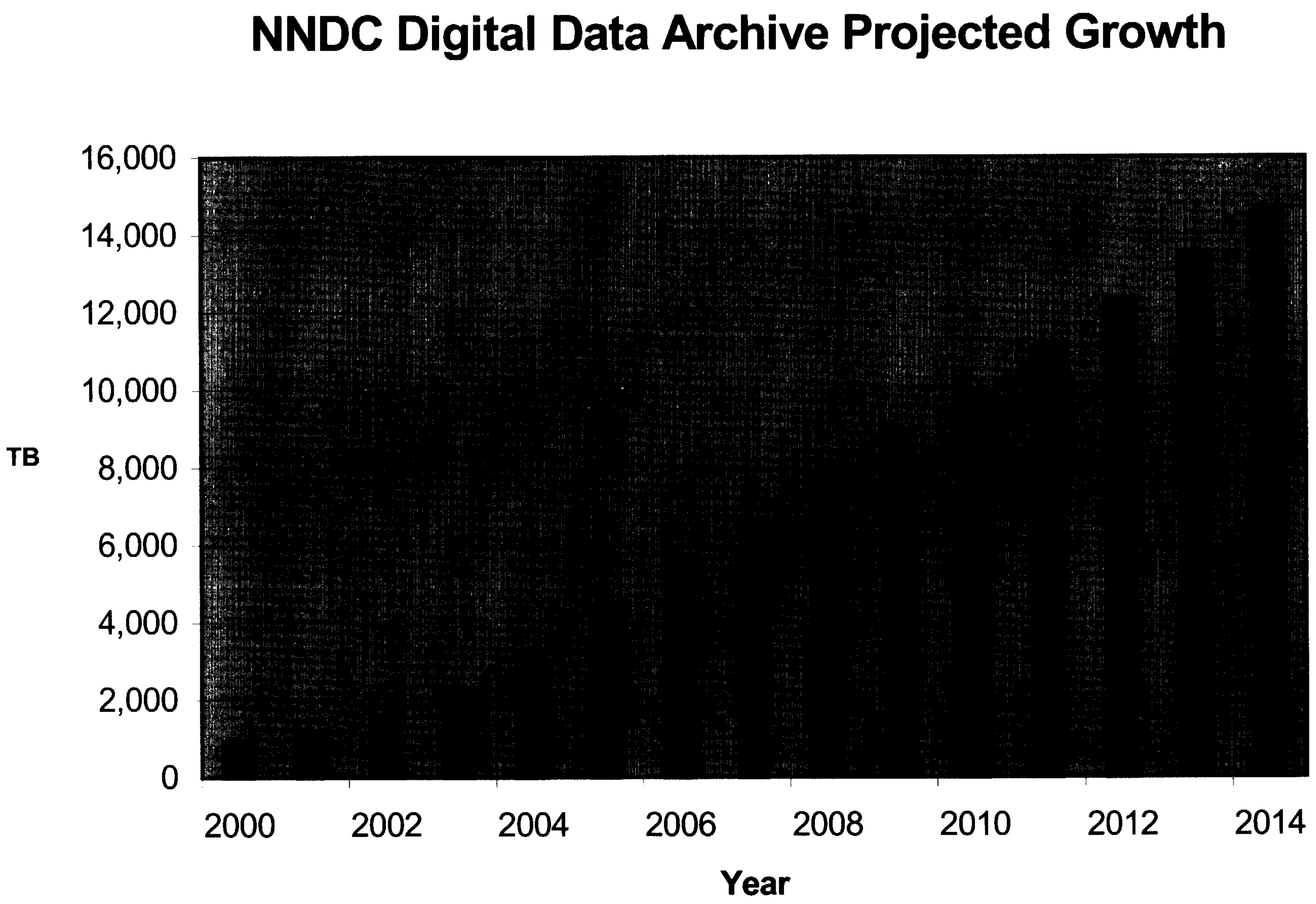

NNDC Digital Data Archive Projected Growth
TB
16,000
14,000
12,000
10,000
8,000
6,000
4,000
2,000
0
2000
2002
2004
2006
2008
2010
2012
2014
Year

Figure 5

Implementing Journaling in a Linux Shared Disk File System

Kenneth W. Preslan - Sistina Software, Inc.

**Andrew Barry, Jonathan Brassow, Russell Cattelan,
Adam Manthei, Erling Nygaard, Seth Van Oort,
David Teigland, Mike Tilstra, and Matthew O'Keefe**

Parallel Computer Systems Laboratory, Dept. of ECE, University of Minnesota
200 Union Street SE, Minneapolis, MN 55455
+1-612-626-7180, okeefe@borg.umn.edu
University of Minnesota

Grant Erickson - Brocade Communications

Manish Agarwal - VERITAS Software

Abstract

In computer systems today, speed and responsiveness is often determined by network and storage subsystem performance. Faster, more scalable networking interfaces like Fibre Channel and Gigabit Ethernet provide the scaffolding from which higher performance computer systems implementations may be constructed, but new thinking is required about how machines interact with network-enabled storage devices.

In this paper we describe how we implemented journaling in the Global File System (GFS), a shared-disk, cluster file system for Linux. Our previous three papers on GFS at the Mass Storage Symposium discussed our first three GFS implementations, their performance, and the lessons learned. Our fourth paper describes, appropriately enough, the evolution of GFS version 3 to version 4, which supports journaling and recovery from client failures.

In addition, GFS scalability tests extending to 8 machines accessing 8 4-disk enclosures were conducted: these tests showed good scaling. We describe the GFS cluster infrastructure, which is necessary for proper recovery from machine and disk failures in a collection of machines sharing disks using GFS. Finally, we discuss the suitability of Linux for handling the big data requirements of supercomputing centers[1].

1 Introduction

Traditional local file systems support a persistent name space by creating a mapping between blocks found on disk drives and a set of files, file names, and directories. These file

[1] The work by Grant Erickson and Manish Agarwal on GFS was performed while they were at the University of Minnesota.

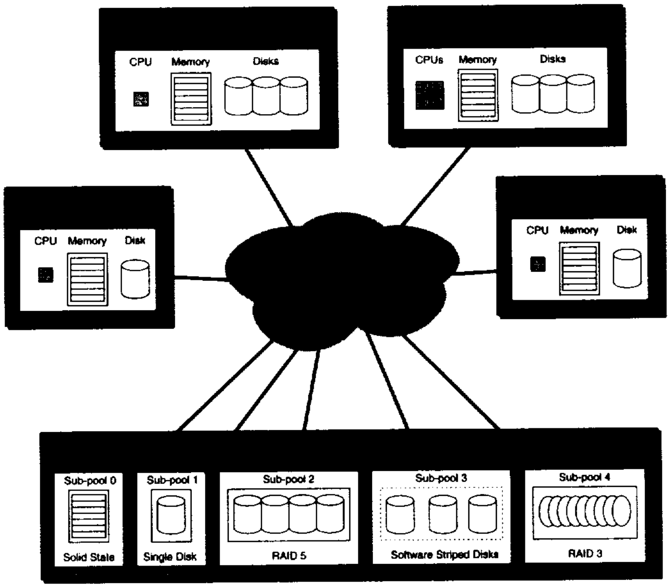


Figure 1: A Storage Area Network

systems view devices as local: devices are not shared so there is no need in the file system to enforce device sharing semantics. Instead, the focus is on aggressively caching and aggregating file system operations to improve performance by reducing the number of actual disk accesses required for each file system operation [1], [2].

New networking technologies allow multiple machines to share the same storage devices. File systems that allow these machines to simultaneously mount and access files on these shared devices are called *shared file systems* [3], [4], [5], [6], [7]. Shared file systems provide a server-less alternative to traditional distributed file systems where the server is the focus of all data sharing. As shown in Figure 1, machines attach directly to devices across a *storage area network* [8], [9], [10].

A shared file system approach based upon a shared network between storage devices and machines offers several advantages:

1. *Availability* is increased because if a single client fails, another client may continue to process its workload because it can access the failed client's files on the shared disk.

2. *Load balancing* a mixed workload among multiple clients sharing disks is simplified by the client's ability to quickly access any portion of the dataset on any of the disks.

3. *Pooling* storage devices into a unified disk volume equally accessible to all machines in the system is possible, which simplifies storage management.

4. *Scalability* in capacity, connectivity, and bandwidth can be achieved without the limitations inherent in network file systems like NFS designed with a centralized server.

We began development of our own shared file system, known as GFS-1 (the Global File System, version 1), in the summer of 1995. At that time, we were primarily interested in exploiting Fibre Channel technology to post-process large scientific datasets [11] on Silicon Graphics (SGI) hardware. Allowing machines to share devices over a fast Fibre Channel network required that we write our own shared file system for IRIX (SGI's System V UNIX variant), and our initial efforts yielded a prototype described in [12]. This implementation used parallel SCSI disks and SCSI reserve and release commands for synchronization. Reserve and release locked the whole device, making it impossible to support simultaneous file metadata accesses to a disk. Clearly, this was unacceptable.

This bottleneck was removed in our second prototype, known as GFS-2, by developing a fine-grain lock command for SCSI. This prototype was described in our 1998 paper [6] and the associated thesis [13]; we also described our performance results across four clients using a Fibre Channel switch and RAID-3 disk arrays. Performance did not scale past three clients due to lock contention and the lack of client caching. In addition, very large files were required for good performance and scalability because neither metadata (or locks) nor file data were cached on the clients.

By the spring of 1998, we began porting our code to the open source Linux operating system. We did this for several reasons, but the primary one was that IRIX is closed source, making it very difficult to cleanly integrate GFS into the kernel. Also, Linux had recently acquired 64-bit, SMP, and floating-point support on Digital Equipment Corporation (DEC) Alpha platforms that were adequate for our computing needs.

In addition, we shed our narrow focus on large data applications and broadened our efforts to design a general-purpose file system that scaled from a single desktop machine to large clusters of machines enabled for device sharing. Because kernel source was available, we could finally support metadata and file data caching, but this required changes to the lock specification, detailed in the 0.9.4 device lock specification [14], [15].

The GFS port to Linux involved a complete re-write of GFS-2, resulting in a new version we call GFS-3 [7]. In addition to support for caching, GFS-3 supported leases on locks which time out if a GFS machine fails to heartbeat the lock. Client IDs are returned with the lock so that callbacks can be made to request that clients release metadata. GFS-3 used extendible hashing for the directory data structure to allow large numbers of files in a single directory. GFS-3's scalability has been measured up to 8 machines and 8 disk units with no measurable interference between machines making independent file accesses and disk requests across the storage network. At this point, we believe that there are no significant limits to GFS scalability.

However, though GFS-3 fixed most of the performance and scalability issues that had arisen for the previous versions of GFS, it did not address a critical requirement for shared disk cluster file systems: fault-tolerance. A production-quality shared file system must be able to withstand machine, network, or shared disk failures without disrupting continued cluster processing [3]. To this end, we have now implemented file system journaling in the latest version of GFS, known as GFS-4. When a GFS machine fails and the failure is detected, the remaining clients in the cluster may recover for the failed machine by replaying its journal. With the addition of journaling, GFS is now ready for consideration for deployment in production environments, after a reasonable beta test phase. We discuss the use of Linux and GFS in processing large datasets later in this paper.

In the following sections we describe GFS-4 (which we will refer to simply as GFS in the remainder of this paper), the current implementation including the details of our journaling code, new scalability results, changes to the lock specification, and our plans for GFS-5, including file system resizing.

2 GFS Background

For a complete description of GFS-3 see [7], for GFS-2 see [6], and for GFS-1 see [12]. In this section we provide a summary of the key features of the Global File System.

2.1 Dlocks

Device Locks are mechanisms used by GFS to synchronize client access to shared metadata. They help maintain metadata coherence when metadata is cached by several clients. The locks are implemented on the storage devices (disks) and accessed with the SCSI device lock command we call *Dlock* [16], [17], [15]. The Dlock command is independent of all other SCSI commands, so devices supporting the locks have no awareness of the nature of the resource that is locked. The file system provides a mapping between files and Dlocks.

GFS-3 used Dlock version 0.9.4 [15], which included timeouts on a per lock basis, multiple reader/single writer semantics, and inclusion of lock-holding client ID information in the SCSI reply data. The latest, journaled version of GFS (version 4) uses Dlock version 0.9.5 [17], which includes some new features we describe in the following sections. Both the 0.9.4 and 0.9.5 Dlock specifications have been implemented as daemon processes that can executed on any machine on the network. A machine that runs a Dlock daemon process is called a Dlock server. A Dlock server can provide provide Dlock functionality in systems constructed with storage devices that do not have SCSI Dlock support [18].

2.1.1 Expiration

In a shared disk environment, a failed client cannot be allowed to indefinitely hold whatever locks it held when it failed. Therefore, each holder must continually update a timer on the disk. If this timer ever expires, other lock holders may begin error recovery functions to eventually free the lock. Expiration is alternately referred to as timing-out, and the act of updating the timer is often referred to as heartbeating the timer or the timer-device. In version 0.9.4, time-outs were per lock; in 0.9.5, they are per-client.

2.1.2 Client IDs

The Client ID is a unique identifier for each client. The client id is completely opaque to the Dlock device. In GFS the client ID is used both as an identifier and to store the IP address of the client, allowing inter-machine communication. The Client ID can be any arbitrary 32-bit number that uniquely identifies a machine.

2.1.3 Version Numbers

Associated with every lock is a version number. Whenever the data associated with a lock is changed, the version number is incremented. Clients may use cached data instead of re-reading from the disk as long as the version number on the dlock is unchanged since the data was last read. The drawback with version numbers is that a client must still read the version number (which is located on the dlock storage device or dlock server); this is often a high-latency operation (even simple SCSI commands that do not touch the disk often require at least 1 millisecond).

Version numbers are an optional dlock feature, and are are unused in GFS-4, which relies instead on callbacks to keep cached metadata consistent. Version numbers may be removed in a future version of the device lock specification.

2.1.4 Conversion Locks

The conversion lock is a simple one stage queue used to prevent writer starvation. In Dlock version 0.9.4, one client may try to acquire an exclusive lock but fail because other clients are constantly acquiring and dropping the shared lock. If there is never a gap where no client is holding the shared lock, the writer requesting exclusive access never gets the lock. To correct this, when a client unsuccessfully tries to acquire a lock, and no other client already possesses that lock's conversion, the conversion is granted to the unsuccessful client. Once the conversion is acquired, no other clients can acquire the lock. All the current holders eventually unlock, and the conversion holder acquires the lock. All of a client's conversions are lost if the client expires.

2.1.5 Enable

In the event that a lock device is turned off and comes back on, all the locks on the device could be lost. Though it would be helpful if the locks were stored in some form of persistent storage, it is unreasonable to require it. Therefore, lock devices should not accept dlock commands when they are first powered up. The devices should return failure results, with the enabled bit of the dlock reply data format cleared, to all dlock actions except refresh timer until a dlock enable is issued to the drive.

In this way, clients of the lock device are made aware that the locks on the lock device have been cleared, and can take action to deal with the situation. This is extremely important, because if machines assume they still hold locks on failed devices or on dlock servers that have failed, then two machines may assume they both have exclusive access to a given lock. This inevitably leads to file system corruption.

2.2 Pool - A Linux Volume Driver

The Pool logical volume driver coalesces a heterogeneous collection of shared storage into a single logical volume. It was developed with GFS to provide simple logical device capabilities and to deliver Dlock commands to specific devices at the SCSI driver layer [19]. If GFS is used as a local file system where no locking is needed, then Pool is not required.

Pool also groups constituent devices into sub-pools. Sub-pools are an internal construction which does not affect the high level view of a pool[2] as a single storage device. This allows intelligent placement of data by the file system according to sub-pool characteristics. If one sub-pool contains very low latency devices, the file system could potentially place commonly referenced metadata there for better overall performance. There is not yet a GFS interface designed to allow this. Sub-pools are currently used in a GFS file system balancer [20]. The balancer moves files among sub-pools to spread data more evenly. Sub-pools now have an additional "type" designation to support GFS journaling. The file system requires that some sub-pools be reserved for journal space. Ordinary sub-pools will be specified as data space.

There are two other volume managers available in Linux. Linux LVM (logical volume manager) was developed by Heinz Mauelshagen [21] and provides traditional volume manager functionality including volume resizing, on-line addition and deletion of volumes (both physical and logical levels), and on-line reallocation of physical disk space. Work is in progress to develop a volume snapshotting capability in Linux LVM. There is also a software RAID driver called MD developed by Ingo Mulnar that supports RAID levels 0, 1, 4, and 5 [22]. Pool does not support software RAID or volume resizing and virtualization, while neither Linux LVM nor MD support multiple clients accessing the same volume. Finding ways to integrate the functionality found in these different volume managers would be very useful.

2.2.1 Modular Block Drivers

Device drivers are the collection of low-level functions in the OS used to access hardware devices. Drivers can provide various levels of functionality. They range from directly manipulating hardware registers to providing a more abstract view of devices, making them easier to program [2]. The "I/O subsystem" refers to the mid-level drivers and OS routines which come between hardware-specific drivers and upper level system calls.

Low-level, device-specific driver functions include:

- Device initialization and control
- Interrupt handling
- Transferring bytes of data into and out of buffers

Mid- and upper-level driver functions include:

- Device naming
- Buffering and caching
- Providing a consistent, programmable interface

In Linux, file systems and drivers may be written as kernel modules. This allows them to be dynamically installed and removed from the kernel. Although not required, this makes development much easier because the entire kernel need not be recompiled and restarted to make a change in a module [23].

[2] The logical devices presented to the system by the Pool volume driver are affectionately called "pools".

Pool is a mid-level block driver built atop the SCSI and FC drivers. This means it conforms to the standard block driver interfaces, but remains at a higher level of abstraction. The following list describes the basic functionality provided by Pool:

- *init_module, cleanup_module* are the functions required by Linux to be a kernel module. They basically call pool_init when the driver is added and free memory when it is removed.

- *pool_init* is the function called when the Pool module is installed in the kernel. It registers Pool in the global table of block drivers. It also initializes data structures maintained by Pool describing currently managed pools. Registering a block driver in the kernel includes specifying a major device number, a name (pool), and a set of functions used to access devices of this type.

- *pool_open, pool_release* are called for a specific pool after the open and close system calls on the pool's device node. Usage counts are incremented or decremented. These would be called due to a mount or I/O on the device node.

- *pool_read, pool_write* are called after a read or write system call on the pool's device node. They pass requests to the Linux routines `block_read()` and `block_write()`. These are not regularly used since the file system calls lower level routines directly.

- *pool_ioctl* is used to request Dlocks, get listings of currently configured pools, add new pools, remove pools, get basic size information or control debugging.

The following functions are internal pool routines called due to pool_ioctl requests:

- *add_pool* is called during *passemble* to configure a new pool. Information describing the new pool is provided by *passemble* and used to allocate new structures in the Pool driver. The parameters describe the pool, sub-pools, underlying physical devices, Dlocks, and striping. Most of the code is OS-independent and is handled in *gen_pool.* All lower-level devices are opened at this stage to verify they can be used. The minor number selected for the new pool is returned to user space.

- *remove_pool* removes a specific pool by closing underlying devices and freeing data structures. Pools are removed when the -r(emove) option is specified in *passemble.*

- *list_pool, list_spool* return descriptions of currently managed pools. This information is needed by the tools *passemble* and *pinfo.* This is also how mkfs_gfs (make a gfs file system command) gets sub pool and Dlock information when creating a new file system.

The last function, *pool_map*, is the most interesting and central to Pool's purpose. It is used to map requests from a logical pool device and block number to a real device and block number. The map functions for Pool and other volume managers are called in an unusual way in Linux. This is the topic of the next section.

```
if (MAJOR(bh->b_rdev) == POOL_MAJOR)
        pool_map(&bh->b_rdev, &bh->b_rsector);

else if (MAJOR(bh->b_rdev) == MD_MAJOR)
        md_map(&bh->b_rdev, &bh->b_rsector);

else if (MAJOR(bh->b_rdev) == LVM_MAJOR)
        lvm_map(&bh->b_rdev, &bh->b_rsector);
```

Figure 2: Pseudocode of mapping functions called directly.

2.2.2 Block Mapping

All reads and writes to block devices occur in chunks defined by the file system block size (usually 4 or 8 KB). Each of these block I/O requests is defined by a `buffer_head` structure (`bh`). All the `bh`'s are managed by the I/O subsystem and a `bh` for a specific request is passed through OS layers from the file system [3] down to the low level disk driver. Two `bh` fields are especially important in specifying the block:

- *rdev*: the device for this request (major, minor numbers)
- *rsector*: the block number on the device rdev

When using a volume driver, a `bh` comes from the file system with `rdev` equal to the *logical* device and `rsector` equal to the logical block. When the `bh` reaches the specific disk driver, `rdev` and `rsector` must specify a *real* device and block number. The I/O subsystem routine which processes the `bh` below the file system is `ll_rw_block()`. Here is where the specific volume manager mapping function is called to change `rdev` and `rsector`.

To call the correct volume manager mapping function (`pool_map`, `md_map`, or `lvm_map`), the original major number of `rdev` is checked because each volume manager has a unique major number. When the specific volume manager is identified, its map function can be called directly. Pseudocode illustrating this is in Fig. 2.

The method of calling specific map functions can be improved by making the code more general. Every block driver has an identification structure in the global block driver table `blk_dev[]` indexed by major number. The function pointer `map_fn` is added to `blk_dev_struct` as seen in Fig. 3. The function pointers are initially set to NULL for every driver. If a block driver has a map function, like `pool_map`, it will set `map_fn` to that function in the driver init routine.

The pseudocode segment in Fig. 2 can then be simplified as seen in Fig. 4. The `map_fn` field is compared to NULL. If the pointer is set, the driver for this block device has defined a map function which is then called through the function pointer. The map function itself

[3] The Linux kernel is moving away from using the buffer cache at these levels as the page cache will be used for most I/O. The buffer cache will be used at lower levels only.

```
typedef void (request_fn_proc)(void);
typedef int (makerq_fn_proc)(struct buffer_head *, int rw);
typedef int (map_fn_proc)(kdev_t, kdev_t *, unsigned long *,
                          unsigned long);
typedef struct request ** (queue_proc)(kdev_t dev);

struct blk_dev_struct {
       request_fn_proc         *request_fn;
       makerq_fn_proc          *makerq_fn;
       map_fn_proc             *map_fn;
       queue_proc              *queue;
       void                    *data;
       struct request          *current_request;
       struct request    plug;
       struct tq_struct plug_tq;
};
```

Figure 3: Entries in block device switch table.

```
dev = blk_dev + major;

if (dev->map_fn && dev->map_fn (bh[i]->b_rdev,
                                &bh[i]->b_rdev,
                                &bh[i]->b_rsector,
                                bh[i]->b_size >> 9)) {
       printk (KERN_ERR ''Bad map in ll_rw_block'');
       goto sorry;
}
```

Figure 4: Generic mapping in ll_rw_block()

determines the new rdev and rsector by using the number and size of each subpool and sub-device. The two fields in the bh are then rewritten.

The same method used for the map function is also used for the make_request function which is less common and used by volume drivers when doing mirroring. The map_fn and make_request functions will probably be combined in the future. Using function pointers for map and request routines was originally designed by Chris Sabol in the GFS group who also worked on the initial Pool port and Pool under IRIX.

The Linux framework for block drivers described above is different from the standard approach in other OS's. In the more common method, each block driver defines a *strategy* function as the single I/O entry point in the block device switch table. The file system and any volume drivers always call the strategy function of a buffer's device. Within a volume driver's strategy routine, mapping is done before calling the next driver's strategy routine.

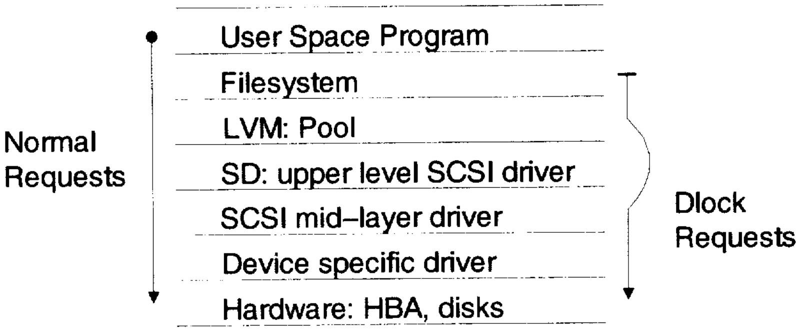


Figure 5: File System and driver layers

The Linux I/O subsystem was designed assuming no volume driver layer which is why mapping routines are called aside from the normal I/O path.

2.2.3 Dlock Support

A unique feature of the Pool driver required by the GFS locking layer is Dlock support. When a pool is configured, particular sub-devices are specified as supporting Dlocks. The Pool driver merges all the available Dlocks into a uniform Dlock space accessible by the GFS locking layer through pool_ioctl. When a Dlock is requested, Pool maps the logical Dlock number to an actual Dlock on a specific device. Because the highest level SCSI driver is not aware of the DLOCK command, Pool needs to construct the appropriate SCSI command descriptor block (CDB) and insert it into the SCSI mid-layer driver.

Pool's handling of Dlock commands is illustrated in Figure 5. There are three layers between the file system and the lowest level hardware driver. First is the volume manager which translates buffer addresses into real devices and block offsets. Not shown in the diagram is other I/O subsystem code which merges and queues buffer requests. Second is the upper level SCSI driver, SD, which breaks buffer requests into actual READ and WRITE SCSI commands. Next is the mid level driver which manages all SCSI devices and sends CDB's to the correct host bus adapters. Because SD deals only with transferring data with READ and WRITE it must be bypassed for other SCSI commands. Below this, the mid level driver sends many other CDB's and does not care about the specific command. This is naturally the level where DLOCK commands can be queued.

Interfacing Pool's Dlock code with the mid-layer SCSI driver[4] required a patch to the upper SD driver. SD maintains the array: `Scsi_Disk *rscsi_disks`. The array is indexed by minor numbers so the detailed structure of a SCSI device is easily accessible given the device number. Part of the `Scsi_Disk` structure is a required parameter to the `scsi_allocate_device()` routine. This allocate routine returns a `Scsi_Cmd` pointer which is passed into `scsi_do_cmd()`. The "do command" function sends the Dlock CDB. So a kernel patch (distributed with GFS and Pool) exporting the `rscsi_disks`

[4] The mid-layer SCSI driver code can be found in /usr/src/linux/drivers/scsi/scsi.c and the upper-level SCSI driver is /usr/src/linux/drivers/scsi/sd.c

symbol from the SD driver is required to access the appropriate `Scsi_Disk` given the minor number of the Dlock device.

Originally, the Pool driver handled all Dlock retries, timeouts, activity monitoring and resets. All the code implementing this was rewritten when moving to Linux, making it simpler and eliminating some incorrect behavior. Eventually as the Dlock specification became more complex, all these functions were moved into a separate locking module. In the current implementation, Pool only maps Dlock requests and sends them to devices.

2.3 File System Metadata

GFS distributes its metadata throughout the network storage pool rather than concentrating it all into a single superblock. Multiple resource groups are used to partition metadata, including data, dinode bitmaps and data blocks, into separate groups to increase client parallelism and file system scalability, avoid bottlenecks, and reduce the average size of typical metadata search operations. One or more resource groups may exist on a single device or a single resource group may include multiple devices.

Resource groups are similar to the Block Groups found in Linux's Ext2 file system. Like resource groups, block groups exploit parallelism and scalability by allowing multiple threads of a single computer to allocate and free data blocks; GFS resource groups allow multiple clients to do the same.

GFS also has a single block, the superblock, which contains summary metadata not distributed across resource groups, including miscellaneous accounting information such as the block size, the journal segment size, the number of journals and resource groups, the dinode numbers of the three hidden dinodes and the root dinode, some lock protocol information, and versioning information.

Formerly, the superblock contained the number of clients mounted on the file system, bitmaps to calculate the unique identifiers for each client, the device on which the file system is mounted, and the file system block size. The superblock also once contained a static index of the resource groups which describes the location of each resource group and other configuration information. All this information has been moved to hidden dinodes (files).

There are three hidden dinodes:

1) The resource index – The list of locations, sizes, and glocks associated with each resource group

2) The journal index – The locations, sizes and glocks of the journals

3) The configuration space dinode – This holds configuration information that is used by the locking modules and transparent to GFS. The Dlock/Dlip modules use it to store namespace information about the cluster. (This is necessary for our "Dlock" cluster infrastructure.)

There are four identifiers that each member of the cluster needs to know about all the other members: Hostname, IP address, Journal ID Number, and Client ID number. The quartets for each host in the cluster are stored in the config space dinode and passed to the lock module as it is initialized. What format the data is in is up to the lock module. The data will be written to the config file using a GFS ioctl call or standard write calls.

This data is stored in files because it needs to be able to grow as the filesystem grows. In previous versions of GFS, we just allocated a static amount of space at the beginning of the filesystem for the Resource Index metadata, but this will cause problems when we expand the filesystem later. If this information is placed in a file, it is much easier to grow the file system at a later time, as the hidden metadata file can grow as well.

The Global File System uses Extendible Hashing [24], [7], [25] for its directory structure. Extendible Hashing (ExHash) provides a way of storing a directory's data so that any particular entry can be found very quickly. Large directories do not result in slow lookup performance.

2.4 Stuffed Dinodes

A GFS dinode takes up an entire file system block because sharing a single block to hold metadata used by multiple clients causes significant contention. To counter the resulting internal fragmentation we have implemented dinode stuffing which allows both file system information and real data to be included in the dinode file system block. If the file size is larger than this data section the dinode stores an array of pointers to data blocks or indirect data blocks. Otherwise the portion of a file system block remaining after dinode file system information is stored is used to hold file system data. Clients access stuffed files with only one block request, a feature particularly useful for directory lookups since each directory in the pathname requires one directory file read.

GFS assigns dinode numbers based on the disk address of each dinode. Directories contain file names and accompanying inode numbers. Once the GFS lookup operation matches a file name, GFS locates the dinode using the associated inode number. By assigning disk addresses to inode numbers GFS dynamically allocates dinodes from the pool of free blocks.

2.5 Flat File Structure

GFS uses a flat pointer tree structure as shown in Figure 6. Each pointer in the dinode points to the same height of metadata tree. (All the pointers are direct pointers, or they are all indirect, or they are all double indirect, and so on.) The height of the tree grows as large as necessary to hold the file.

The more conventional UFS file system's dinode has a fixed number of direct pointers, one indirect pointer, one double indirect pointer, and one triple indirect pointer. This means that there is a limit on how big a UFS file can grow. However, the UFS dinode pointer tree requires fewer indirections for small files. Other alternatives include extent-based allocation such as SGI's EFS file system or the B-tree approach of SGI's XFS file system [26]. The current structure of the GFS metadata is an implementation choice and these alternatives are worth exploration in future versions of GFS.

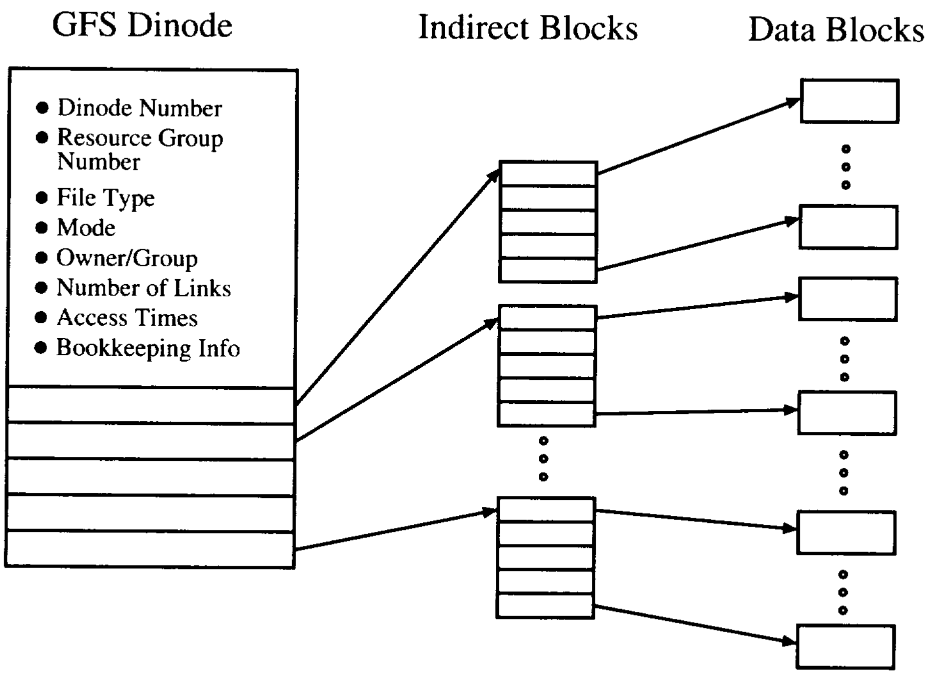


Figure 6: A GFS dinode. All pointers in the dinode have the same height in the metadata tree.

3 Improvements in GFS Version 4

Since our presentation at the IEEE/NASA Mass Storage Symposium last year, there have been many improvements to GFS. We describe some of these improvements in the following sections.

3.1 Abstract Kernel Interfaces

We have abstracted the kernel interfaces above GFS, to the file-system-independent layer, and below GFS, to the block device drivers, to enhance GFS's portability.

3.2 Fibre Channel in Linux

Until the summer of 1999, Fibre Channel support in Linux was limited to a single machine connected to a few drives on a loop. However, significant progress has been made in the quality of Fibre Channel fabric drivers and chipsets available on Linux. In particular, QLogic's QLA2100 and QLA2200 chips are well-supported in Linux, with multiple GPL'ed drivers written by QLogic and independent open source software developers. During testing in our laboratory with large Fabrics (32 ports) and large numbers of drives and GFS clients, the Fibre Channel hardware and software has performed well. Recent reductions in adapter card and switch costs have made it possible to cost-effectively build large, Fibre-Channel-based storage networks in Linux.

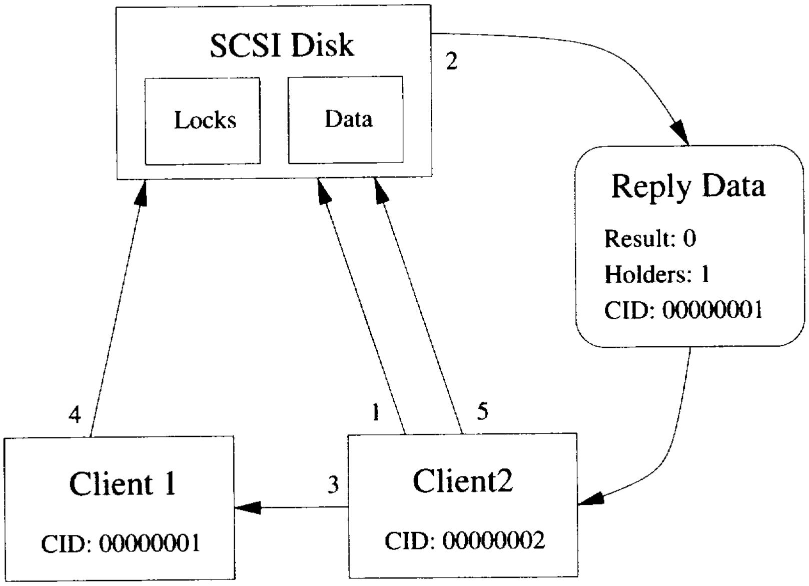


Figure 7: Callbacks on glocks in GFS

However, it is possible to use GFS to share network disks exported through standard, IP-based network interfaces like Ethernet using Linux's Network Block Device software. In addition, new, fast, low-latency interfaces like Myrinet combined with protocol layers like VIA hold the promise of high performance, media-independent storage networks.

3.3 Booting Linux from GFS and Context-Sensitive Symbolic Links

It is possible to boot Linux from a GFS file system. In addition, GFS supports context-sensitive symbolic links, so that Linux machines sharing a cluster disk can see the same file system image for most directories, but where convenient (such as /etc/???) can symbolically link to a machine-specific configuration file.

These two features provide building blocks for implementing a single system image by providing for a shared disk from which the machines in a cluster can boot up Linux, yet through context-sensitive symbolic links each machine can still maintain locally-defined configuration files. This simplifies system administration, especially in large clusters, where maintaining a consistent kernel image across hundreds of machines is a difficult task.

3.4 Global Synchronization in GFS

The lock semantics used in previous versions of GFS were tied directly to the SCSI Dlock command. This tight coupling was unnecessary, as the lock usage in GFS could be abstracted so that GFS machines could exploit any global lock space available to all machines. GFS-4 supports an abstract lock module that can exploit almost any globally accessible lock space, not just Dlocks. This is important because it allows GFS cluster architects to buy any disks they like, not just disks that contain Dlock firmware.

The GFS lock abstraction allows GFS clients to implement callbacks, as shown in Figure 7. When client 2 needs a dlock exclusively that is already held by client 1, client 2 first sends it's normal dlock SCSI request to the disk drive (step 1 in the figure). This request fails and returns the list of holder ClientIDs, which happens to be client 1 (step 2). Client 2 sends a callback to client 1, asking B to give up the lock (step 3). Client 1 syncs all dirty (modified) data and metadata buffers associated with that dlock to disk (step 4), and releases the dlock, incrementing the version number if any data has been written. Client A may then acquire the dlock (step 5).

Because clients can communicate with each other, they may hold dlocks indefinitely if no other clients choose to read from inodes associated with dlocks that are held. As long as a client holds a dlock, it may cache any writes associated with the dlock. Caching allows GFS to approach the performance of a local disk file system; our goal is to keep GFS within 10-15% of the performance of the best local Linux file system systems across all workloads, including small file workloads.

In GFS-4, write caching is write-back, not write-through. GFS uses Global Locks (glocks), which may or may not be dlocks. GFS uses interchangeable locking modules, some of which map glocks to Dlocks. Other locking methods, such as a distributed lock manager [9] or a centralized lock server, can also be used. Our group has developed a centralized lock server known as the GLM (Global Locking Module) [18]. GFS sees the Glocks as being in one of three states:

1. Not Held – This machine doesn't hold the Glock. It may or may not be held by another machine.

2. Held – this machine holds the Glock, but there is no current process using the lock. Data in the machine's buffers can be newer than the data on disk. If another machine asks for the lock, the current holder will sync all the dirty buffers to disk and release the lock.

3. Held + Locked – the machine holds the Glock and there is a process currently using the lock. There can be newer data in the buffers than on disk. If another machine asks for the lock, the request is ignored temporarily, and is acted upon later. The lock is not released until the process drops the Glock down to the Held state.

When a GFS file system writes data, the file system moves the Glock into the Held+Locked state, acquiring the Dlock exclusively, if it was not already held. If another process is writing to that lock, and the Glock is already Held+Locked, the second process must wait until the Glock is dropped back down to Held.

The Write is then done asynchronously. The I/O isn't necessarily written to disk, but the cache buffer is marked dirty. The Glock is moved back to the Held state. This is the end of the write sequence.

The Buffers remain dirty until either bdflush or a sync causes the buffers to be synced to disk, or until another machine asks for the lock, at which point the data is synced to disk and the Glock is dropped to Not Held and the Dlock is released. This is important because it allows a GFS client to hold a Glock until another machine asks for it, and service multiple requests for the same Glock without making a separate dlock request for each process.

3.5 GFS and Fibre Channel Documentation in Linux

We have developed documentation for GFS over the last year. Linux HOWTOs on GFS and Fibre Channel can be found at the GFS web page: http://www.globalfilesystem.org. In addition, there are conventional man pages for all the GFS and Pool Volume Manager utility routines, including mkfs, ptool, passemble, and pinfo[18].

4 File System Journaling and Recovery in GFS

To improve performance, most local file systems cache file system data and metadata so that it is unnecessary to constantly touch the disk as file system operations are performed. This optimization is critical to achieving good performance as the latency of disk accesses is 5 to 6 orders of magnitude greater than memory latencies. However, by not synchronously updating the metadata each time a file system operation modifies that metadata, there is a risk that the file system may be inconsistent if the machine crashes.

For example, when removing a file from a directory, the file name is first removed from the directory, then the file dinode and related indirect and data blocks are removed. If the machine crashes just after the file name is removed from the directory, then the file dinode and other file system blocks associated with that file can no longer be used by other files. These disk blocks are now erroneously now marked as in use. This is what is meant by an inconsistency in the file system.

When a single machine crashes, a traditional means of recovery has been to run a file system check routine (fsck) that checks for and repairs these kinds of inconsistencies. The problem with file system check routines is that (a) they are slow because they take time proportional to the size of the file system, (b) the file system must be off-line while the fsck is being performed and, therefore, this technique is unacceptable for shared file systems. Instead, GFS uses a technique known as file system journaling to avoid fsck's altogether and reduce recovery time and increase availability.

4.1 The Transaction Manager

Journaling uses transactions for operations that change the file system state. These operations must be atomic, so that the file system moves from one consistent on-disk state to another consistent on-disk state. These transactions generally correspond to VFS operations such as create, mkdir, write, unlink, etc. With transactions, the file system metadata can always be quickly returned to a consistent state.

A GFS journaling transaction is composed of the metadata blocks changed during an atomic operation. Each journal entry has one or more locks associated with it, corresponding to the metadata protected by the particular lock. For example, a creat() transaction would contain locks for the directory, the new dinode, and the allocation bitmaps. Some parts of a transaction may not directly correspond to on-disk metadata.

Two types of metadata buffers are involved in transactions: primary and secondary. Primary metadata includes dinodes and resource group headers. They contain a generation number that is incremented each time they are changed, and that is used in recovery. There must always be one piece of primary metadata for each lock in the transaction. Secondary metadata includes indirect blocks, directory data, and directory leaf blocks; these blocks do not have a generation number.

A transaction is created in the following sequence of steps:

(1) start transaction

(2) acquire the necessary Glocks

(3) check conditions required for the transaction

(4) pin the in-core metadata buffers associated with the transaction (i.e., don't allow them to be written to disk)

(5) modify the metadata

(6) pass the Glocks to the transaction

(7) commit the transaction by passing it to the Log Manager

To represent the transaction to be committed to the log, the Log Manager is passed a structure which contains a list of metadata buffers. Each buffer knows its Glock number, and its type (Dinode, RG Header, or Secondary Metadata). Passing this structure represents a commit to the in-core log.

4.2 The Log Manager

The Log Manager is separate from the transaction module. It takes metadata to be written from the transaction module and writes it to disk. The Transaction Manager pins, while the Log Manager unpins. The Log Manager also manages the Active Items List, and detects and deals with Log wrap-around.

For a shared file system, having multiple clients share a single journal would be too complex and inefficient. Instead, as in Frangipani [4], each GFS client gets its own journal space, that is protected by one lock that is acquired at mount time and released at unmount (or crash) time. Each journal can be on its own disk for greater parallelism. Each journal must be visible to all clients for recovery.

In-core log entries are committed asynchronously to the on-disk log. The Log Manager follows these steps:

(1) get the transaction from the Transaction Manager

(2) wait and collect more transactions (asynchronous logging)

(3) perform the on-disk commit

(4) put all metadata in the Active Items List

(5) unpin the secondary metadata

(6) later, when the secondary metadata is on disk, remove it from the Active Items List

(7) unpin the primary metadata

(8) later, when the primary metadata is on disk, remove it from the Active Items List

Recall that all journal entries are linked to one or more Glocks, and that Glocks may be requested by other machines during a callback operation. Hence, callbacks may result in particular journal entries being pushed out of the in-core log and written to the on-disk log. Before a Glock is released to another machine, the following steps must be taken:

(1) journal entries dependent on that Glock must be flushed to the log

(2) the in-place metadata buffers must be synced

	Glock #			
Journal Entry	2	3	6	8
1	X	X		
2		X	X	X
3	X	X		
4			X	X

X represents in-memory metadata buffers which will be written to the journal

Figure 8: Journal Write Ordering Imposed by Lock Dependencies During GFS Lock Callbacks

(3) the in-place data buffers must be synced

Only journal entries directly or indirectly dependent on the the requested Glock need to be flushed. A journal entry is dependent on a Glock if either (a) it references that Glock directly, or (b) it has Glocks in common with earlier journal entries which reference that Glock directly.

For example, in Figure 8, four journal entries in sequential order (starting with 1) are shown, along with the Glocks upon which each transaction is dependent. If Glock 6 is requested by another machine, journal entries 1, 2, and 4 must be flushed to the on-disk log in order. Then the in-place metadata and data buffers must be synced for Glock 6, and finally Glock 6 is released.

4.3 Recovery

Journal recovery is initiated by clients in several cases:

(a) a mount time check shows that any of the clients were shutdown uncleanly or otherwise failed

(b) a locking module reports an expired client when it polls for expired machines

(c) a client tries to acquire a Glock and the locking module reports that the last client to hold that Glock has expired

In each case, a recovery kernel thread is called with the expired client's ID. The machine then attempts to begin recovery by acquiring the journal lock of a failed client. A very dangerous special case can result when a client (known as a zombie) fails to heartbeat its locks, so the other machines think it is dead, but it is still alive; this could happen, for example, if for some reason the "failed" client temporarily was disconnected from the network. This is dangerous because the supposedly failed client's journal will be recovered by another client, which has a different view of the file system state. This "split-brain" problem will result in file system corruption. For this reason, the first step in recovery after acquiring the journal lock of a failed client is to either (1) forcibly disable the failed client or, (2) fence out all IO from the client using the zoning feature of a Fibre Channel switch.

Once a client obtains the journal lock for a failed client, journal recovery proceeds as follows: the tail (start) and head (end) entries of the journal are found. Partially-committed entries are ignored. For each journal entry, the recovery client tries to acquire all locks associated with that entry, and then determines whether to replay it, and does so if needed.

All expired locks are marked as not expired for the failed client. At this point, the journal is marked as recovered.

The decision to replay an entry is based on the generation number in the primary metadata found in the entry. When these pieces of metadata are written to the log, their generation number is incremented. The journal entry is replayed if the generation numbers in the journal entry are larger than the in-place metadata.

Note that machines in the GFS cluster can continue to work during recovery unless they need a lock held by a failed client.

4.4 Comparison to Alternative Journaling Implementations

The main difference between journaling a local file system and GFS is that GFS must be able to flush out transactions in an order other than that in which they were created. A GFS client must be able to respond to callbacks on locks from other clients in the cluster. The client should then flush only the transactions that are dependent on that lock. This means that GFS can't combine transactions into compound transactions until just before the transaction is committed to the disk.

When a GFS client unlinks a file from the directory structure, the file isn't actually deallocated until all clients have stopped using it. In order to determine which clients are using a given dinode, GFS must maintain an "nopen" count in each dinode. This is a counter of the clients that are using a dinode. When a client crashes, it leaves nopen references on all the dinodes that it was using. As part of recovery, the machine doing the recovery must determine which dinodes the failed client was using and decrement nopen count on those dinodes.

Hence, each GFS client maintains a list of all the dinodes it has nopen references on. Every time an inode is opened or closed, a marker is put in the journal describing the operation. Since the log can wrap many times during the time that a dinode is held by the client, this list is periodically re-logged in its journal.

GFS also has to label some metadata blocks with generation numbers that are incremented when transactions are committed. These generation numbers and the current state of the global locks are used to decide whether or not a given journal entry should be replayed during recovery.

As mentioned previously with respect to generation numbers, GFS has two types of metadata: Primary and Secondary. Primary metadata has version numbers and must be persistent on the disk – once the block is allocated as primary metadata, it can never be reused for real data or secondary metadata. Secondary metadata isn't subject to either of these two restraints.

One difference between Frangipani [4] and GFS is that Frangipani doesn't make a distinction between primary and secondary metadata. All Frangipani metadata is primary metadata. This is a good choice for Frangipani because of the unique nature of the Petal [27] block device underneath it. GFS is greatly simplified, however, by not having to maintain lists of unused indirect blocks, directory blocks, and other secondary metadata. The trade-off is that GFS has the extra constraint that secondary metadata must be flushed to the disk before any primary metadata for each compound transaction.

5 Performance Results

Figures 9 and 11 represent the current single client I/O bandwidth of Linux GFS (GFS-3 release Antimatter-Anteater was used for the tests. This release does not include journaling.) The tests were performed on a 533 MHz Alpha with 512 MB of RAM running Linux 2.2.13. The machine was connected to eight Seagate ST19171FC Fibre Channel drives on a loop with a Qlogic host adapter card. A 4,096-byte block size was used for these tests. (A block size of 8,192 bytes yields numbers that about 10 percent better, but this larger block size isn't available on all Linux architectures.) The Transfer Size axis represents the size of the file being transferred, whereas the Request Size represents the actual size of each file transfer request. So, for example, a 4096 MB file transferred using 4 requests yields a request size of 1024 MB and a transfer size of 4096 MB.

The bandwidth of first time creates, shown in Figure 9, peaks at around 55 MB/s. The read bandwidth shown in Figure 11 peaks at about 45 MB/s.

The GFS-3 single client performance can be compared to our previous GFS-2 single client performance numbers reported in [7] and shown in Figures 10 and 12. The machine configurations for these tests were essentially the same, although we used Linux kernel 2.2.0-pre7 for the GFS-2 tests. GFS-2 read bandwidth peaked out at 42 MB/s while GFS-3 reads peaked out at 48 MB/s. (notice the scale differences in the figures between GFS-2 and GFS-3 results).

Though the peak bandwidths are relatively close, notice how GFS-3 read performance is much better for smaller request sizes. For GFS-3 creates, the maximum performance is 50 MB/s versus 18 MB/s for GFS-2. As in the read case, GFS-3 create performance is much higher at smaller request sizes than GFS-2 create performance.

The significant GFS-3 performance advantage comes from the fact that GFS-2 had only read buffer caching, whereas GFS-3 has both read buffer and lock caching. GFS-2 would need to check the lock (a separate SCSI request) each time a buffer was read or written, whereas GFS-3 uses callbacks to enable lock caching. As long as no other client needs the lock, no lock request is made to disk during buffer accesses. The performance improvement is even more pronounced for writes, since writes can nearly always be cached. The actual write to disk only occurs later during a periodic sync operation, or when the buffer cache is pressured by the kernel to release buffer space to applications.

Figure 13 is a comparison of the extendible hashing directory structure in GFS-3 to the linear directory structure of Ext2. The test involved creating a million entry directory. Creates per second were measured at regular intervals as the directory was filled. The GFS curve levels off because of un-cached hash tables. Even for large directory sizes (10s of thousands directory entries), GFS can create 100s of files per second. Fast directory operations for directories with thousands of files are necessary to support applications with millions of small files.

Figure 14 shows one to four GFS-3 hosts being added to a constant size file system and each performing a workload of a million random operations. These four machines were connected across a Brocade Fabric switch to 4 4-disk enclosures, each configured as a single 4-disk loop. The workload consisted of 50 percent reads, 25 percent appends/creates and 25 percent unlinks. Each machine was working in in its own directory and the direc-

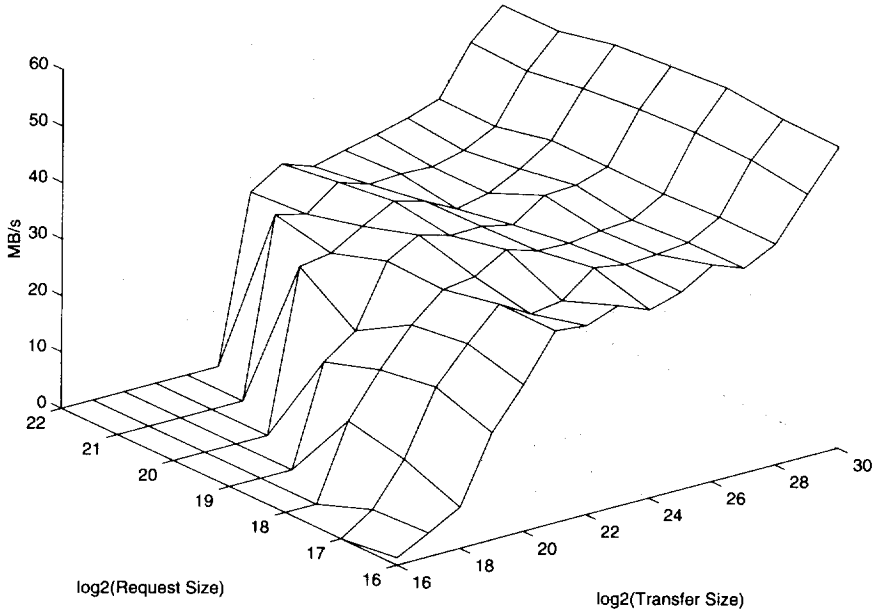


Figure 9: GFS-3 single machine create bandwidth

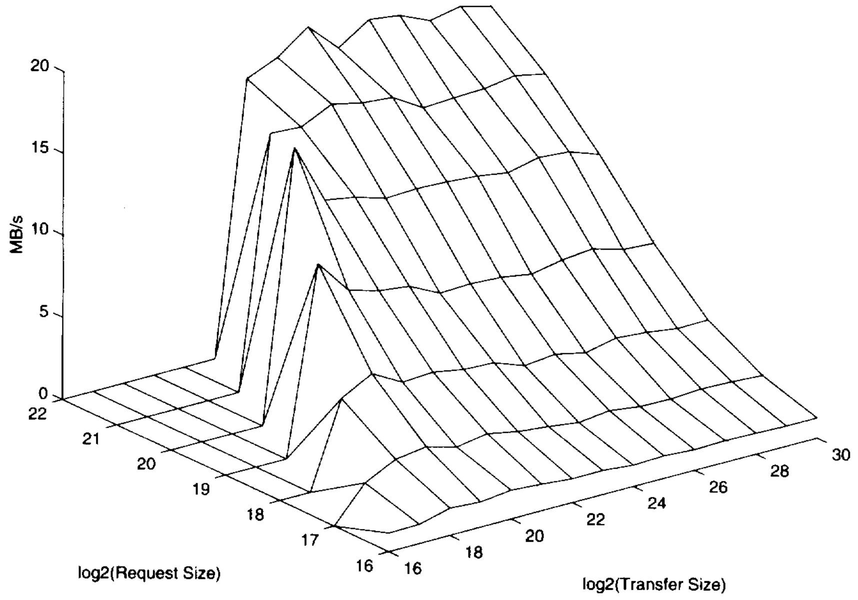


Figure 10: GFS-2 single machine create bandwidth

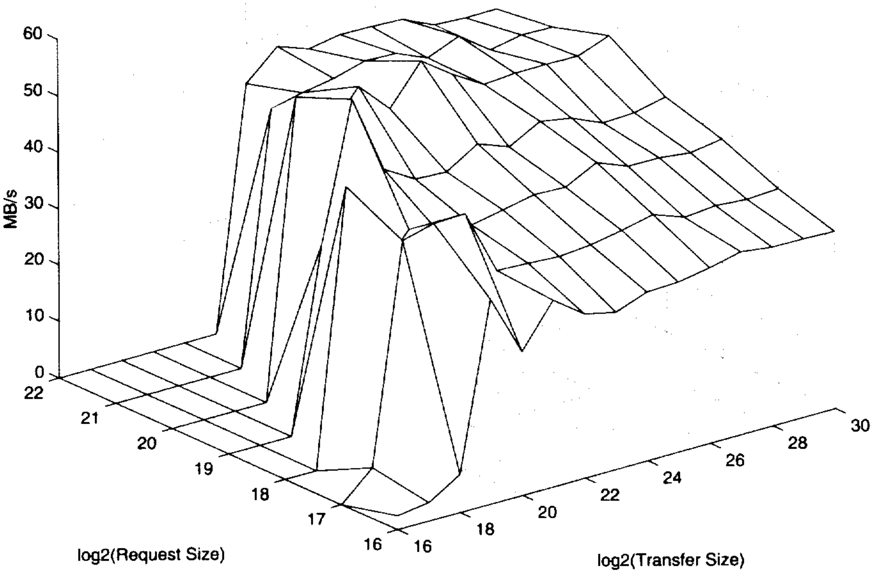


Figure 11: GFS-3 single machine read bandwidth

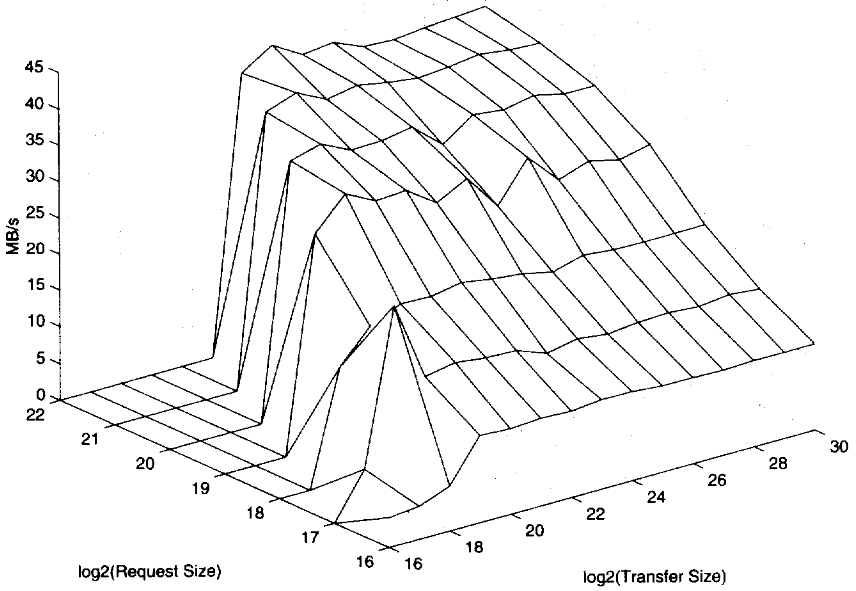


Figure 12: GFS-2 single machine read bandwidth

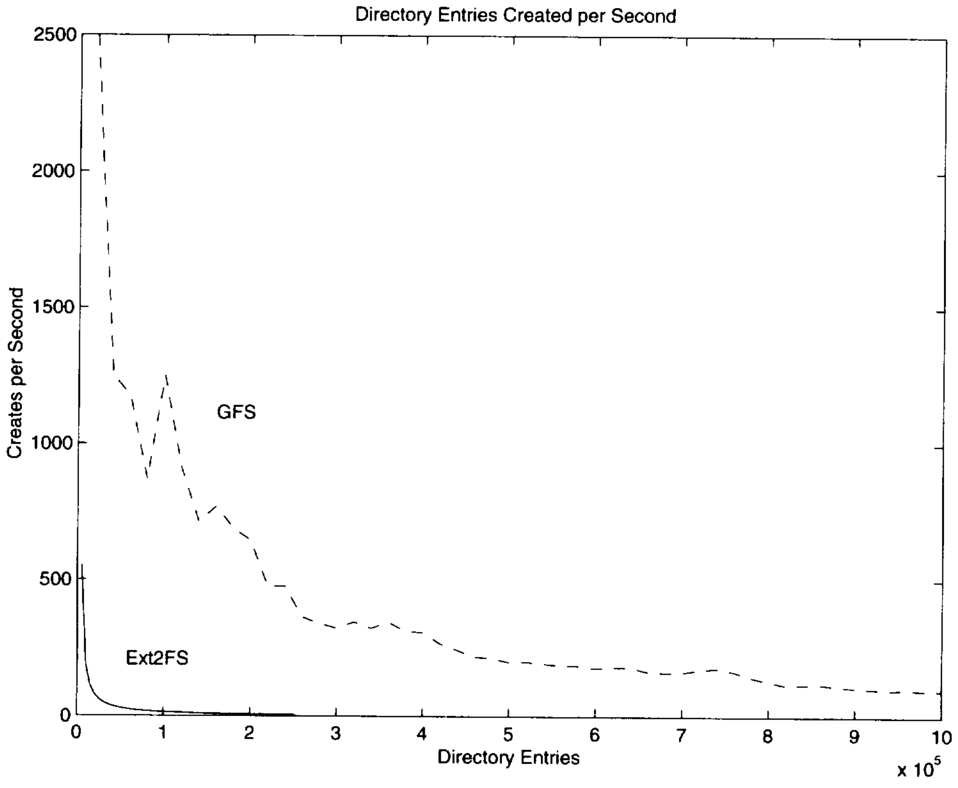


Figure 13: File creates per second versus directory size

tories were optimally placed across the file system. Notice that the scalability curve shows nearly perfect speedup. Similar results were achieved for an 8-way cluster. These new results compare favorably with the dismal scaling results obtained for the early versions of GFS [6], which didn't cache locks, file data, or file system metadata.

6 Prospects for Linux in Big Data Environments

The ability of Linux to handle supercomputing workloads is improving rapidly. In the past year, Linux has gained 2 journaled file systems (ext3fs and Reiserfs) with two more on the horizon (XFS and GFS). SGI is porting its XFS file system to Linux. SGI has also open-sourced its OpenVault media management technology. Improvements in NFS client performance, virtual memory, SMP support, asynchronous and direct IO, and other areas will allow Linux to compete and surpass other UNIX implementations.

The open source nature of Linux provides better peer review on both architecture and code. Linux is free, and appears to be well on its way towards becoming the standard server operating system of the future. This means that most server applications will be ported to it in time, and that competition for Linux support and specialized services will develop.

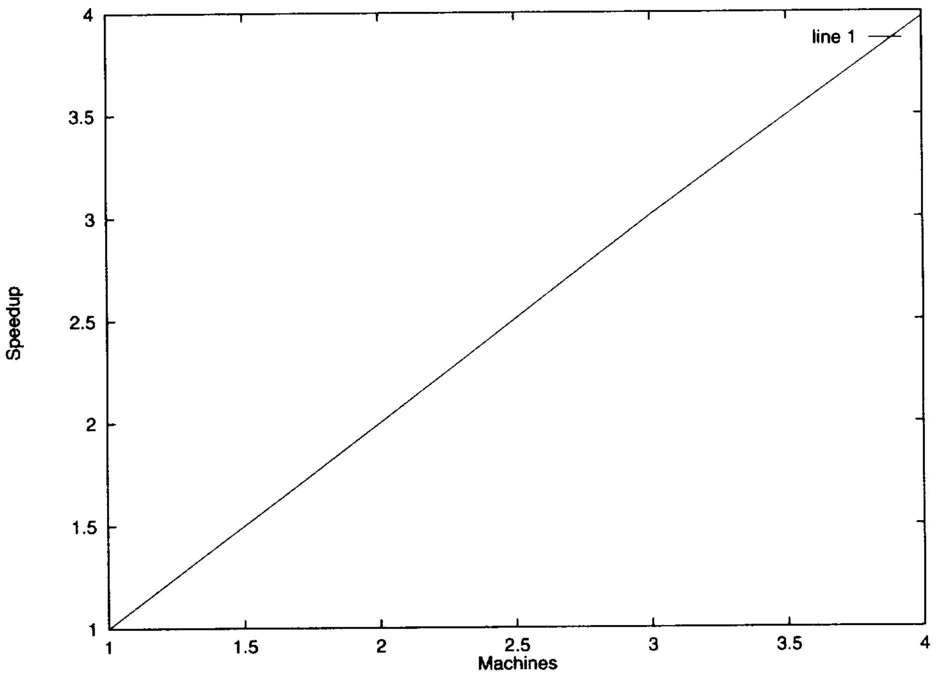


Figure 14: Four machine speedup for independent operations

7 Conclusions and Future Work

In this paper, we described the GFS journaling and recovery implementation and other improvements in GFS version 4 (GFS-4). These include a lock abstraction and network block driver layer, which allow GFS to work with almost any global lock space or storage networking media. The new lock specification (0.9.5) provides for better fairness and other improvements to support journaling and recovery. In addition, a variety of other changes to the file system metadata and pool volume manager have increased both performance and flexibility. Taken together, these changes mean that GFS can now enter a beta test phase as a prelude to production use. Early adopters who are interested in clustered file systems for Linux are encouraged to install and test GFS to help us validate its performance and robustness.

Once the work on journaling and recovery is complete, we intend to consider several new features for GFS. These may include file system versioning for on-line snapshots of file system state using copy-on-write semantics. File system snapshots allow an older version of the file system to be backed up on-line while the cluster continues to operate. This is important in high-availability systems. Heinz Mauelshagen is implementing snapshotting in the Linux LVM volume manager [21], and so it may not be necessary to support this feature in GFS if we can use LVM to create GFS pools.

The ability to re-size the file system on-line is also very important, especially in storage area networks, where it will be quite common for new disks to be continually added to the SAN.

Finally, Larry McVoy, Peter Braam, and Stephen Tweedie are developing a scalable cluster infrastructure for Linux. This will include a Distributed Lock Manager (DLM) and mechanisms to detect and recover from client failures and cluster partitioning. This infrastructure could be very helpful in implementing recovery in GFS.

8 Acknowledgments

This work has been supported by NASA Ames Research Center through grant NAG2-1308, and by the Department of Defense High Performance Computing and Modernization Office (HPCMO) via Logicon contract number 9720954.

Many people have contributed code, ideas, and financial support to the GFS Group over the years. John Lekashman of NASA Ames provided both ideas and funding through NASA. Without John's support, we could never have gotten as far as we have with GFS.

Our industry sponsors include Brocade (Mark Sausville and Mike Lovas), Seagate (Mike Miller), EMC (Richard Wheeler), VERITAS Software (Alex Mirschenchenko), SGI (Jim Mostek and Dan Koren), and StorageTek (Aloke Guha and Jim Hughes). These sponsors helped make GFS possible with both funding and equipment.

Steve Soltis worked on early versions of GFS. Other people contributing ideas included Tom Ruwart, Alan Poston, Aaron Sawdey and Ben Gribstad. Jim Wayda of Dot Hill helped develop the 0.9.5 dlock specification, in particular the conversion lock approach. Tony Hecker of Seagate implemented the 0.9.4 specification in the latest Seagate Fibre Channel drives; Nate Larsen implemented th 0.6 dlock specification in an earlier generation of Seagate disk drives. Jim Coomes, Gerry Houlder, and Mike Miller of Seagate helped develop the device lock SCSI command.

Several Linux developers helped contribute implementation and architecture ideas to improve GFS, including Stephen Tweedie, Ted Ts'o, Peter Braam, and David Brower. Jim Mostek, Dan Koren, and Bill Earl of SGI helped us understand journaled file system implementations. Heinz Mauelshagen helped review this paper.

Chris Worley of Alta Technology has been our earliest and most aggressive early adopter, and we appreciate his GFS testing efforts. Tim Riker of Caldera, another aggressive early adopter, built a search engine using GFS. Paul Hayes, Laurie Miller and JD Harrington at the University of Minnesota used GFS to support their parallel electromagnetic simulations. Luke Lonergan and Greg Lindahl of HPTI installed GFS for evaluation on a Beowulf cluster at the Forecast Systems Lab in Boulder, Colorado. John Kaitschuck and Andy Hendrickson are evaluating a GFS cluster for supporting the NFS workload at Industrial Light and Magic. Tad Kollar is building a GFS cluster as part of a Beowulf machine at the NASA Glenn Research Center.

The authors would also like to acknowledge Jim Finlayson, who is on the Program Committee, for guiding the paper through to publication. Thanks also to Mike Declerck of NERSC for reviewing this paper and the GFS HOWTO, and for insights on the impact of real production use on several GFS features.

References

[1] L. McVoy and S. Kleiman. Extent-like performance from a unix file system. In *Proceedings of the 1991 Winter USENIX Conference*, pages 33–43, Dallas, TX, June 1991.

[2] Uresh Vahalia. *Unix Internals: The New Frontiers*. Prentice-Hall, 1996.

[3] Roy G. Davis. *VAXCluster Principles*. Digital Press, 1993.

[4] Chandramohan Thekkath, Timothy Mann, and Edward Lee. Frangipani: A scalable distributed file system. In *Proceedings of the 16th ACM Symposium on Operating Systems Principles*, pages 224–237, October 1997.

[5] Matthew T. O'Keefe. Shared file systems and fibre channel. In *The Sixth Goddard Conference on Mass Storage Systems and Technologies in cooperation with the Fifteenth IEEE Symposium on Mass Storage Systems*, pages 1–16, College Park, Maryland, March 1998.

[6] Steve Soltis, Grant Erickson, Ken Preslan, Matthew O'Keefe, and Tom Ruwart. The design and performance of a shared disk file system for IRIX. In *The Sixth Goddard Conference on Mass Storage Systems and Technologies in cooperation with the Fifteenth IEEE Symposium on Mass Storage Systems*, pages 41–56, College Park, Maryland, March 1998.

[7] Kenneth W. Preslan et al. A 64-bit, shared disk file system for linux. In *The Seventh NASA Goddard Conference on Mass Storage Systems and Technologies in cooperation with the Sixteenth IEEE Symposium on Mass Storage Systems*, pages 22–41, San Diego, CA, March 1999.

[8] Alan F. Benner. *Fibre Channel: Gigabit Communications and I/O for Computer Networks*. McGraw-Hill, 1996.

[9] N. Kronenberg, H. Levy, and W. Strecker. VAXClusters: A closely-coupled distributed system. *ACM Transactions on Computer Systems*, 4(3):130–146, May 1986.

[10] K. Matthews. Implementing a Shared File System on a HiPPi disk array. In *Fourteenth IEEE Symposium on Mass Storage Systems*, pages 77–88, September 1995.

[11] Aaron Sawdey, Matthew O'Keefe, and Wesley Jones. A general programming model for developing scalable ocean circulation applications. In *Proceedings of the 1996 ECMWF Workshop on the Use of Parallel Processors in Meteorology*, pages 209–225, Reading, England, November 1996.

[12] Steven R. Soltis, Thomas M. Ruwart, and Matthew T. O'Keefe. The Global File System. In *The Fifth NASA Goddard Conference on Mass Storage Systems and Technologies*, volume 2, pages 319–342, College Park, Maryland, March 1996.

[13] Steven R. Soltis. *The Design and Implementation of a Distributed File System Based on Shared Network Storage.* PhD thesis, University of Minnesota, Department of Electrical and Computer Engineering, Minneapolis, Minnesota, August 1997.

[14] Matthew T. O'Keefe, Kenneth W. Preslan, Christopher J. Sabol, and Steven R. Soltis. X3T10 SCSI committee document T10/98-225R0 – Proposed SCSI Device Locks. http:// ftp.symbios.com/ ftp/ pub/ standards/ io/ x3t10/ document.98/ 98-225r0.pdf, September 1998.

[15] Ken Preslan et al. Dlock 0.9.4 specification. http:// www.globalfilesystem.org/Pages/dlock.html, December 1998.

[16] Kenneth W. Preslan et al. Scsi device locks version 0.9.5. In to appear in *The Eighth NASA Goddard Conference on Mass Storage Systems and Technologies in cooperation with the Seventeenth IEEE Symposium on Mass Storage Systems*, College Park, MD, March 2000.

[17] Ken Preslan et al. Dlock 0.9.5 specification. http:// www.globalfilesystem.org/Pages/dlock.html, December 1999.

[18] Mike Tilstra et al. The gfs howto. http://www.globalfilesystem.org/howtos/gfs_howto.

[19] David Teigland. The pool driver: A volume driver for sans. Master's thesis, University of Minnesota, Department of Electrical and Computer Engineering, Minneapolis, Minnesota, October 1999. http://www.globalfilesystem.org/Pages/theses.html.

[20] Manish Agarwal. A Filesystem Balancer for GFS. Master's thesis, University of Minnesota, Department of Electrical and Computer Engineering, Minneapolis, Minnesota, June 1999. http://http://www.globalfilesystem.org/Pages/theses.html.

[21] Heinz Mauelshagen. Linux lvm home page. http:// linux.msede.com/lvm.

[22] Ingo Mulnar. Linux software raid driver. http://ostenfeld.dk/j̃akob/Software-RAID.HOWTO/Software-RAID.HOWTO.html#toc4.

[23] Alessandro Rubini. *Linux Device Drivers.* O'Reilly & Associates, 1998.

[24] Ronald Fagin, Jurg Nievergelt, Nicholas Pippenger, and H. Raymond Strong. Extendible Hashing – a fast access method for dynamic files. *ACM Transactions on Database Systems*, 4(3):315–344, September 1979.

[25] Michael J. Folk, Bill Zoellick, and Greg Riccardi. *File Structures.* Addison-Wesley, March 1998.

[26] Adam Sweeney, Doug Doucette, Wei Hu, Curtis Anderson, Mike Nishimoto, and Geoff Peck. Scalability in the xfs file system. In *Proceedings of the USENIX Annual Technical Conference*, pages 1–14, San Diego, CA, January 1996.

[27] Edward Lee and Chandramohan Thekkath. Petal: Distributed virtual disks. In *ASPLOS VII*, pages 84–92, San Diego, CA, October 1996.

[28] John Lekashman. Building and managing high performance, scalable, commodity mass storage systems. In *The Sixth Goddard Conference on Mass Storage Systems and Technologies in cooperation with the Fifteenth IEEE Symposium on Mass Storage Systems*, pages 175–179, College Park, Maryland, March 1998.

All GFS publications and source code can be found at
`http://www.globalfilesystem.org`.

Design for a Decentralized Security System for Network Attached Storage

William Freeman[1,2] and Ethan Miller[1]

[1]Dept. of Computer Science and Electrical Engineering,
University of Maryland Baltimore County, Baltimore MD
[2]Laboratory for Telecommunication Sciences, Adelphi MD
wef@lts.ncsc.mil, elm@csee.umbc.edu

ABSTRACT

This paper describes an architecture for a secure file system based on network-attached storage that guarantees end-to-end encryption for all user data. We describe the design of this system, focusing on the features that allow it to ensure that data is written and read only by authorized users, even in the face of attacks such as network snooping and physically capturing the storage media.

Our work shows that such a system is feasible given the speeds of today's microprocessors, and we discuss benchmark results using several popular encryption and authentication algorithms that could be used on storage devices in such a system. Based on these calculations, we present the overall performance of the system, showing that it is nearly as fast as the non-encrypted file systems in wide use today.

1. Introduction

While computers have provided a great service in the office automation arena, they have led to billions of dollars in lost revenue due to attacks by both hackers and insiders. Most offices and universities rely heavily on their distributed computer environment, which for the purposes of this study, consists of workstations and a shared file system. This file system is typically stored on a centralized file server that is managed by a system administrator with super-user privileges. The need to back up the file system requires that the super-user has the ability to read the entire file system. When the end users want to read a file, the file is sent across the network without any protection against rogue users simply reading the data as it travels. A more sophisticated hacker could also modify or prevent the modification of data.

In our first paper published on this topic [5], the technical feasibility of placing cryptographic controls in a performance-critical system was established. This paper addresses the security and performance concerns of today's distributed file systems. The performance problems are caused by having few (possibly one) network connections, and shared hardware resources (namely the backplane) in the file server. Each disk drive in a file server can be coupled with a low-cost board computer to make an intelligent disk. These disks can be distributed across the network which in today's switched LANs, has far greater network bandwidth than a single server. The physical separation of the drives also increases reliability, since one drive having a catastrophic failure such as catching on fire will not damage the other drives.

The security of today's distributed file systems such as NFS and AFS merely provide a weak scheme for access control, but neglect data integrity and confidentiality. This is especially true when considering that the system administrator with super-user privileges can read and write any user data in the system. Our proposed system requires that all user data be encrypted and signed at the user's workstation. This mechanism ensures that any files on the file server are protected from reading or from undetected modification by anyone, even someone with unrestricted physical access to the drives. The drives and the tape backups of the drives will thus contain no sensitive data in non-encrypted form.

2. Motivation

Many systems have been designed to try to solve the performance problems or security problems of modern distributed file systems. A few have even attempted to solve both problems. These systems are discussed below with respect to their shortcomings.

2.1 Security Issues

The primary security capability of current distributed file systems is access control. This is comprised of preventing unauthorized release of information, unauthorized modification of information, or unauthorized denial of resource usage. This is usually provided by something as simple as a user ID passed in the clear or a security token, which often can be thwarted using elementary attacks. Many system designers choose to ignore the fact that MAC addresses (unique per-interface identifiers), IP addresses, TCP sequence numbers, user names, user passwords, and host names are sent in the clear across unsecure networks. NFS, for example, uses file handles to identify files that are being accessed, with two parts being public and one being secret. It is well-known that the "secret" number can often be easily guessed or calculated. Amazingly, the main security function in NFS was not designed for security use, but for preventing two clients from simultaneously accessing different versions of the file [9].

To further complicate the security problem, the standard protocols for Internet communications are the Transmission Control Protocol (TCP) and the User Datagram Protocol (UDP) over the Internet protocol (IP). Many systems rely on the "security" provided by this protocol. An example of this is authenticating the beginning of a session and then just trusting that further communications on that TCP session are legitimate. This is a poor method to provide security because there are many well known attacks that can "steal" an open connection such as IP address spoofing [8].

There are many methods being developed for protecting traffic while traversing a network. For example, IPsec can encrypt the entire datagram in transit between two firewalls, as well as encrypt only the data field for host-to-host communications. Hosts can also set up a Secure Sockets Layer (SSL) which is a transport layer encryption scheme.

There are also many methods in use and in development to provide strong authentication of the user for access control. Instead of merely sending the user's password in the clear, one-time-use data could be sent. For example, S/Key uses the one-way property of certain hashing algorithms to prevent a password replay attack [7]. Kerberos can be implemented in a fairly secure manner but does not scale well with large systems and does not provide for protection of data in transit. Kerberos is a critical component for many systems recently developed, including work securing network attached disks [3].

Although many of these schemes are great ideas and will protect the data as it traverses the network, most do nothing to protect the data as it sits on the disk or backup tapes. The super-user can still read any user's data, and the disks and backup tapes still contain sensitive data, thus requiring physical security. Work has been done to provide a secure local file system, and perhaps even secure a particular user's NFS files, but the idea of providing a secure network attached storage system with decentralized security has not been investigated.

Frangipani, developed at DEC, was one attempt to make a secure distributed file system. It was built over the Petal distributed virtual disk system. Frangipani used a client/server configuration, but unlike other distributed file systems, any Frangipani machine can read or write any block of the Petal virtual disk. This imposed the requirement that any Frangipani computer runs a trusted operating system [11].

Some system designers have tried to fix the single point of failure problem with a centralized security scheme by having some small number of computers at the heart of the security mechanism. While this makes the system more reliable, there is still a privileged user that can compromise the entire system. Decentralized security means that there is no central computer(s) responsible for providing data security services such as access control. There are different degrees of decentralization of information security, determined by how much of the security services are performed at a common, central computer.

There are significant benefits from a decentralized security scheme. From a security standpoint, there is no longer the concept of a "super-user" that has the capability to read and write any of the system's data files. Only each end-user has the capability to decrypt files they have access to, and only the end-user can sign data blocks for writing. The creator / modifier of stored data creates the encrypted keys for users to access the data, because this function can not be performed by anyone who does not have access to the data.

2.2 Performance Problems with Existing and Previously Proposed Systems

The standard office automation system in use today is comprised of many computers located on various users' desks, attached to file server(s) through a high-speed local area network. 10 Megabits per second was the standard local area network speed until recently, when 100 Megabit Ethernet hit the mainstream marketplace. A second major change in local area networking is the use of switching hubs. These allow multiple pairs of users to communicate at full bandwidth, which was not possible with the shared-medium of previous LANs.

Computer hard drives have been increasing in speed each and every year, and more advanced techniques are being developed to maximize the performance of existing and new hard drives. A commodity SCSI disk can sustain transfer rates of about 200 Megabits per second. Traditional file servers held several SCSI disks striped in a RAID configuration to increase speed and reliability, but the backplane connecting the SCSI controllers is now a bottleneck. If we have eight SCSI drives communicating at 200 Megabits per second, the total transfer rate is 1600 Megabits per second. The standard PCI bus in Intel based computers can only support 1056 Megabits per second (33 MHz * 32 bit,), so the drives are limited by the bus. This problem is exacerbated because the buffers on the drives can transfer data at over 3 times faster than their media transfer rate. To compensate for this deficiency, most file server manufacturers have resorted to expensive proprietary high-speed backplanes. Even this specialized hardware will not be able to keep up with more than 8 hard drives in the near future. There is also a problem with memory bandwidth in the server which is a bottleneck in some faster systems.

There have been attempts to alleviate the bottleneck at the file server by partitioning the directory tree across multiple file servers, but this does not work as well as one might expect. Since the number of file servers is still small, hot spots will still cause throughput problems at particular file servers at various moments in time. With a distributed network file system, the aggregate system bandwidth increases almost linearly with respect to the number of network devices comprising the file system [1]. By striping the data along with parity, the data can still be accessed in spite of a single drive failure. Since each drive in the proposed system is on a separate computer, the probability of a second drive failing before the first failure can be replaced is very small. A drive failure in a modern file server can cause a second drive to fail. For example, one drive could catch fire and damage the other drives. The physical separation of the drives in a network attached file system leads to a very reliable system. The TickerTAIP parallel RAID scheme developed at Hewlett-Packard laboratories has many of the benefits of a network attached storage system, but lacks the reliability of a physically distributed system [4].

The increasing speed of hard disks and networks along with bandwidth intensive applications necessitate high bandwidth network file systems. If the storage is distributed across multiple network segments, the file system could provide aggregate transfer rates well over the rate of a single network connection (1 Gigabit/second soon.) For example, if ten users request data uniformly striped across ten drives, the aggregate transfer rate of the file system could theoretically hit 2 Gigabits per second (10 * 200 Megabits per second for each drive.) Many network file system have used aggressive file caching to improve performance, but with high-speed network storage, caching files on a local disk may become a thing of the past. Caching in local RAM will, of course, still be faster than retrieving data off the network. Network attached disks should provide excellent performance during times of burst traffic because the data will be striped across a large number of disks. By choosing the hashing algorithm correctly to evenly spread out the loads, slow downs due to burst hot-spots can be avoided.

3. Data Structures

The system we propose is called Network Attached Storage with Decentralized Security NAS/DS, building on the Network Attached Secure Disk work performed at CMU [6]. There are four basic new data structures that are used with this system. Secure data objects contain a block of encrypted user data, along with sufficient information to validate the sender of the data, and the data integrity. File objects consist of the file ID, associated key file ID, and one or more secure data objects. Key objects are associated with a file or group of files and contain sufficient information (less the user's private key) to decrypt the file blocks. Finally, certificate objects are stored on each network drive and are used to determine if a particular user is permitted to write or delete data from a file object.

3.1 Secure Data Objects

The secure data object shown in Figure 1 is the basic unit of data that is written to and read from the network drives. Each object contains sufficient information to verify the cryptographic controls on the data. The HMAC (Hash-based Message Authentication Code) proves the integrity of the data and the sender of the data [2]. The timestamp prevents a replay attack - sending an old block instead of a new one. The IV (Initialization Vector) is needed to decrypt the block assuming a CBC (cipher-block chaining) mode encryption is used. This is further described in Section 4. The client computer is responsible for making sure that any secure data object created has a timestamp (counter) that is greater than the block it is replacing, as the drive will not write the data otherwise. This prevents a hacker from writing an old stored block over a new one. Each file, if sufficiently large, is divided into a sequence of secure data objects. The size of the secure data object in this study is 64000 bytes. The larger the secure data object, the less overhead that is associated (per byte) for encrypting and for placing a header on the data. The problem with large secure data objects is that the entire object must be re-written even if a single byte is modified, unless a code-book encryption is used (and this is not recommended).

HMAC	Block_id	UID	Timestamp
IV	Data		

Figure 1. Secure Data Object

3.2 File Objects

A file object contains some meta data for maintaining the file. It consists of the file identifier (fid), key file identifier (key_fid), and a list of block numbers that form the file. When a user wants to read an entire file or a portion of a file, she first reads the file object. This tells what file the encrypted cryptographic keys are stored in, as well as the list of blocks that make up the file. The file object is stored on one network disk, however, the secure data blocks identified by the block_id may be stored across multiple disks. It is the job of a higher level file system built on NAS/DS to manage this hierarchy. Upon reading the file object and key file object, the user can read and write an arbitrary number of secure data objects (assuming permission was granted.)

3.3 Key Objects

Each key object shown in Figure 3 contains two types of information. At the beginning of the key object are the file identification field (fid) and the user identification field (uid.) These are used to determine if a request to modify the actual key object will be allowed. The other information contained in the key object are sets of three-tuples containing a uid, key', and permission bits. The key' is the symmetric key used to encrypt the file — encrypted with the user's public key. The permission bits are similar to the Unix file sys-

fid	key_fid
Block_id 1	
Block_id 2	
...	
Block_id N	

Figure 2. File Object

tem's permission bits. Unix has separate permission fields for the files owner, group, and others; there would be three entries in the key object for similar granularity.

fid		uid
uid	key'	permission bits
uid	key'	permission bits
...		
gid	key'	permission bits

Figure 3. Key Object

3.4 Certificate Objects

Each network drive contains a certificate object, shown in Figure 4, that the drive will use to decide whether or not to grant a write operation. Information for each individual user (UID) and group (GID) is kept as a row in this file. The Key$_{MAC}$ is the shared-secret used for the HMAC generation and verification between each user and the drive. The timestamp field is updated each time a file block is written and is used by the drive to prevent a reply attack. The optional quota field is used by the drive to prevent a particular user from writing more than their allowed amount of data to the drive.

fid			uid	
UID	Key$_{PUB}$	Key$_{MAC}$	Timestamp	Quota
UID	Key$_{PUB}$	Key$_{MAC}$	Timestamp	Quota
UID	Key$_{PUB}$	Key$_{MAC}$	Timestamp	Quota
GID	Key$_{PUB}$	Key$_{MAC}$	Timestamp	Quota

Figure 4. Certificate Object

There is the obvious problem of who gets to write the certificate object. One possibility is that with physical access to the drive, a public key that will be used to validate write requests to the certificate object can be loaded manually. Another possibility is that a privileged user could log into the drive via the network using some secure protocol to write the file. Even if this file is written by a hacker, the confidentiality of the data is still maintained, but a denial of service will likely occur. This assumes that the users do not rely on the certificates stored on the network drives to get public keys when encrypting the symmetric keys, since there is no reason to do this. An error such as using this certificate object for encrypting keys is exactly why the security of this system must be carefully thought out. Subtle errors are usually the key to defeating a security system such as this.

4. System Design and Operation

In this section, the general design of the network attached storage with decentralized security will be discussed. The operation of the system will be described, including such operations as data reads and writes.

4.1 Data Security

The basis of data security in this system lies in the secure data objects. Provided that the user obtains the symmetric encryption key (RC5 key) from the key object, the secure data object contains sufficient information to protect the confidentiality and integrity of the data it contains. This simply means that even if an adversary is able to obtain all of the data that is stored on the network disks, and snoop all of the data that traverses the network, data confidentiality and intégrity are still maintained.

This system uses a keyed-hash approach to authenticate the writer of a data block. In particular, the MD4 hash algorithm is used in a manner similar to MD5-HMAC [2]. Note that although MD4 has known weaknesses, it still provides weak-collision protection which is sufficient for this application. Using HMAC for writer authentication has the disadvantage that the network disk contains sufficient information to forge a data block. Keyed-hash functions have the property that the verifier of the keyed-hash can also create the keyed-hash since it is symmetric - same operation for generating and verifying. If a network disk is compromised with this scheme, it is possible that the adversary could write information to the drive. If the system is built properly, this would require that they were able to obtain the writer-authentication keys from the drive or gained physical access to the hardware. These keys could be stored encrypted for greater security. The compromised system still prevents an adversary from accessing any encrypted data stored on the drive or any data in transit. The alternative to HMAC for user authentication is using a digital-signature, but this is too processor intensive for current computers. Perhaps in four to five years processors will be fast enough for this approach, and end-to-end authentication will be feasible.

This scheme works by including an HMAC as part of each secure data object. The drive is able to determine that the writer corresponding with the provided uid or gid that created the block has write access to the drive by using the writer-authentication key stored in the certificate object. Only someone with access to this key would be able to create HMAC on the block.

Performing an HMAC is substantially faster than a signature generation. The main weakness is the loss of end-to-end integrity assurances. There is no guarantee that the drive did not corrupt the data, since the ability to verify a keyed-hash implies the ability to generate a new one. The corrupted data could almost certainly be detected since it is encrypted. A hash of the plaintext could be appended to each block before it is encrypted to further ensure data integrity.

This system provides the following cryptographic services on secure data objects:

- **Confidentiality** - Each file is written in 64000 byte blocks; the last block may be smaller. Each of these blocks is encrypted with a fast encryption algorithm. This analysis used the RC5 encryption algorithm in CBC mode with a 128-bit key [10]. This ensures the data can not be read without the accompanying RC5 key. This key is used to encrypt each block in a file or group of files. The key is generated by the user upon the creation of a file or file group. This key must be provided to the other users that need access to this file (if any); this is done by encrypting this RC5 key with the public key of each user (UID) or group of users (GID.) These encrypted keys are stored in a key file associated with the data file. Both the data and key files are stored on the network disks.
- **Data Integrity** - Provided by the HMAC of the data. The key used for HMAC can be sent to the drives using RSA when each new user is added to the system.
- **Authentication** - Just as in the other schemes, no user authentication is necessary for the read operation on this system since all of the data is encrypted. Authentication for the write operation is provided by the HMAC. The number of cryptographic functions performed by the host computer and disk computer is shown in Figure 5. The network disks only need to perform one HMAC (Hash) function over the block for each read or write operation, where the host computer needs to perform a hash and encryption. Since the encryption takes longer than the hash (see Table 1), the network disk has less than 1/2 the work to perform as the host computer. This will help the network disks scale to multiple hosts. The bottleneck at the network disk is the network interface as the HMAC function takes substantially less time than the time to send the packet over the 100 Mbps link.

Operation	**Host**	**NAS**
En/Decrypt:	1	0
Hash:	1	1
Signature:	0	0
Verification:	0	0

Block Write

Operation	**Host**	**NAS**
En/Decrypt:	1	0
Hash:	1	1
Signature:	0	0
Verification:	0	0

Block Read

*note: A key-exchange operation is needed upon file creation for each user that needs access. Since this is only done once for each file (or group of files) regardless of the number of blocks, it is omitted.

Figure 5. Number of Cryptographic Operations

4.2 Basic Operation

The network file system will support the basic distributed file system calls. Both user data files as well as directory information are stored as data objects on the network drives. Higher level operations are accomplished via the simpler operations. For example, revoking permission for a particular uid on a file is accomplished by rewriting the key object for the file. This will prevent the user from writing any new data written to the file. If it is necessary to prevent a user from reading data from files that they were previously able to read, the data must be re-encrypted. Of course, this accomplishes little since the user could have simply cached the file on their local system. The data objects on the network disks are subject to reads, writes, and deletes as described below.

4.2.1 Block Write

The write operation starts with encrypting a block of data. This provides data confidentiality as the data traverses the network as well as while it is stored on the disks. The block is then given a timestamp and an HMAC is appended. This forms the secure data object, and it is sent to the network disk as shown in Figure 6. If a block is simply modified, the file object does not need to be changed. However, if a block is added or deleted, the block identifier needs to be added or deleted from the file object. Note that writes to the file object are protected by an HMAC just as writes of any other objects on the network drives.

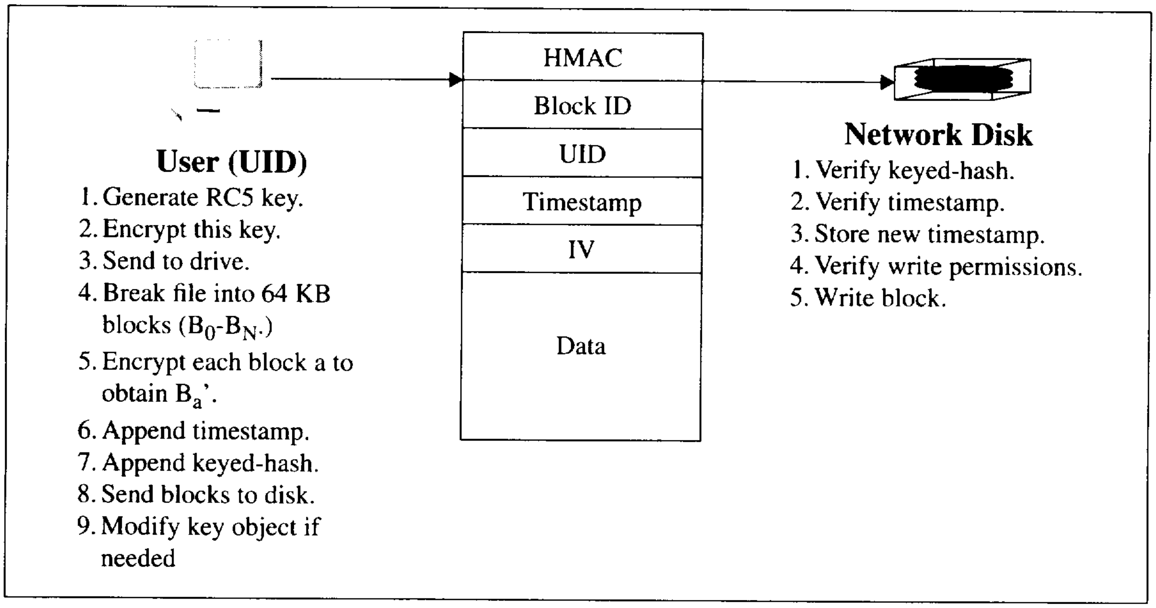
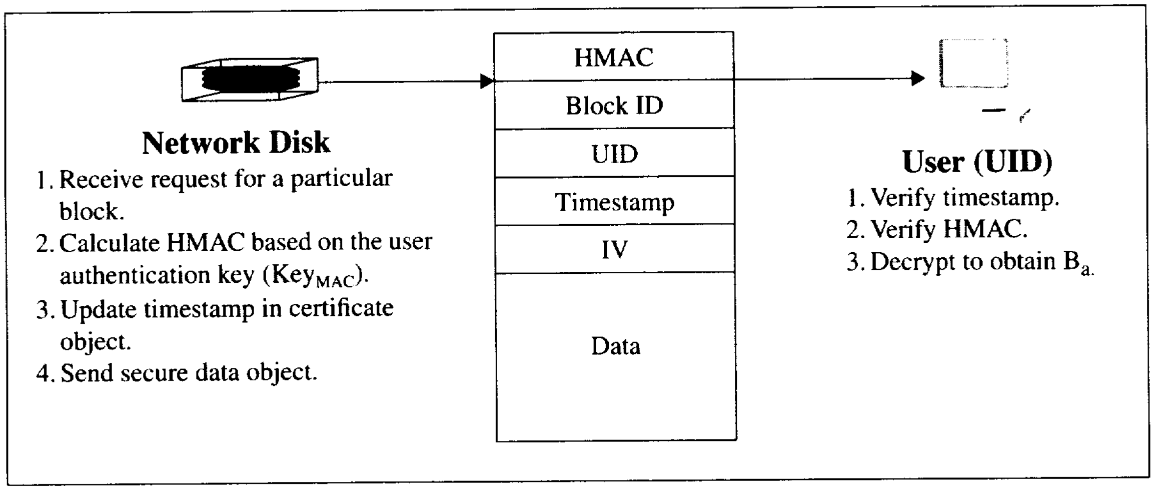


Figure 6. Writing a File

The actual HMAC on the secure data object does not necessarily need to be written to the disks. The old HMAC is not sent when a block is read since only the original writer of the block and the network disk could use it.

4.2.2 Block Read

For the read operation the disk needs to append a keyed-hash for the user requesting the block as well as a timestamp newer than the one last received from that user. For group access, the keyed-hash calculated by the writer could be used. For individual access, the new keyed-hash must be calculated because the reader does not have access to the writer's writer-authentication key. This exchange is shown in Figure 7.



Network Disk

1. Receive request for a particular block.
2. Calculate HMAC based on the user authentication key (Key_{MAC}).
3. Update timestamp in certificate object.
4. Send secure data object.

User (UID)

1. Verify timestamp.
2. Verify HMAC.
3. Decrypt to obtain B_a.

Figure 7. Reading a File

4.2.3 Block Delete

A delete request is handled basically the same way as a write. The drive verifies that the uid in the delete request is the owner of the file, that the signature on the request is valid, and the timestamp is in order. If these conditions are true, the data object is deleted. It is the responsibility of the file system software on The Final System.

5. System Performance

A test system was built to verify the feasibility of this system. This section describes the exact hardware and software of the prototype system, along with performance measurements. The cryptographic component performance, network performance, as well as the system performance is presented.

5.1 System Description

The system used for this study is comprised of 2 Motorola VME boards connected to a 100 Mbps hub as shown in Figure 8. The hub in use does not provide switching capabilities, so the aggregate performance is limited to 100 Mbps. An attempt to use a Cisco Catalyst 2900 switch was thwarted by unsolved device driver issues. Both of the computers in this test setup run the VxWorks real-time operating system. This operating system was chosen to limit the variables introduced with a large multi-user OS such as Unix or Windows.

5.2 Cryptographic Component Performance

The performance of various cryptographic algorithms on a variety of platforms has been tested to verify the feasibility of the proposed file system [5]. For this study, the performance of the hash algorithms to be used in an HMAC (MD4 or MD5) and the encryption algorithm RC5 is shown in Table 1.

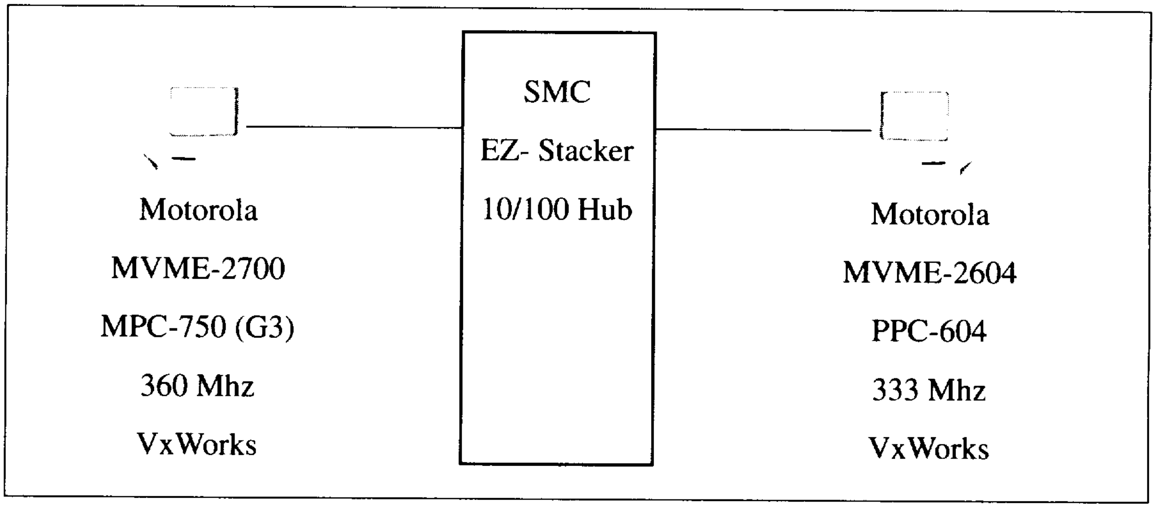


Figure 8. Hardware Configuration

Operation (64000 Bytes)	**MVME-2604 PPC604/333**	**MVME-2700 MPC750/360**
MD4	1.39 ms	1.31 ms
MD5	3.20 ms	2.30 ms
RC5 Encrypt	4.27 ms	3.20 ms
RC5 Decrypt	5.33 ms	4.60 ms
RSA-512 Sign	25 ms	21.2 ms
RSA-512 Verify	2.3 ms	2.0 ms

Table 1. Cryptographic Operation Time - 64000 Byte Block

5.3 Network Performance

The performance of the networking stack implemented in VxWorks for these Motorola boards needed to be investigated to ensure the network was not responsible for a bottleneck. The network buffers and timings were tuned to obtain the results shown in Table 2 it is clear that using UDP, the network performance is not a limiting factor. The difference between the UDP performance and the optimal 100 Mbps is simply the overhead of lower

Protocol	**MVME-2604 PPC604/333**	**MVME-2700 MPC750/360**
TCP	77 Mbps	80 Mbps
UDP	96 Mbps	96 Mbps

Table 2. Network Protocol Performance

level protocols. The VxWorks based computers are capable of generating the UDP datagrams significantly faster, the network is the bottleneck.

5.4 System Performance

The combined system performance for the block read and block write operation was tested for this scheme. The block write followed the protocol shown in Figure 6, thus involved the MPC750 performing a 64000 byte RC5-CBC encryption followed by an MD4 HMAC operation. Upon receipt of the block, the PPC604 acting as the network disk was required to verify the MD4-HMAC, and write the block into memory. The writing of the block to a actual disk was not tested at this point. However, the performance of the Seagate Cheetah drives is much greater than the 100 Mbps network, and should pose no bottleneck.

The read operation tested was similar to the protocol shown in Figure 7, and involved the PPC604 at the network disk performing an MD4 HMAC operation, and the MPC750 at the workstation performing the slower RC5-CBC decrypt operation followed by an MD4 HMAC operation. The difference was the client performing the read operation for these measurements send the datagram over the network instead of receiving it because the code to read from the network at the client was not yet finished. The performance of this read and write operation is shown in Table 3.

Operation (64000 Bytes)	Overall Performance
Block Write	66.4 Mbps
Block Read	56.5 Mbps

Table 3. System Performance

6. Conclusion

This paper presented the details of the Network Attached Storage with Decentralized Security system along with performance measurements. It is clear that this type of system is feasible with today's computing power, and will become even more attractive as processors become faster. The write operation was able to run at over 2/3 of the optimal performance, while the read was limited to 59% of optimal performance. This distributed file system solves many of the performance and security problems in existing systems today. This system protects user data confidentiality and integrity from the moment it leaves the client computer. The distributed disks should perform substantially better than centralized file servers, and provide better reliability. Having the security functionality decentralized will improve performance and scalability. It also removes the single point of failure that plagues many proposed centralized security schemes to date.

REFERENCES

[1] Thomas E. Anderson, Michael D. Dahlin, Jeanna M. Neefe, David A. Patterson, Drew S. Roselli, and Randolph Y. Wang. *Serverless Network File Systems,* ACM Transactions on Computer Systems, Feb. 1996, pp 41 - 79.

[2] M. Bellare, R. Canetti and H.krawczyk. *Keying Hash Functions for Message Authentication,* Proc. Advances in Cryptology, pp 1-15, CRYPTO, 1996.

[3] Steven M. Bellovin and Michael Merritt. *Limitations of the Kerberos Authentication System.* Computer Communications Review, 1991, pp 1 - 15.

[4] Pei Cao, Swee Boon Lim, Shavakumar Venkataraman and John Wilkes. *The TickerTAIP Parallel RAID Architecture.* ACM Transactions on Computer Systems, August 1994, pp 236 - 269.

[5] William E. Freeman, Ethan L. Miller. A*n Experimental Analysis of Cryptographic Overhead in Performance-Critical Systems.* IEEE Mascots '99, pp 348-357.

[6] Garth A. Gibson, David F. Nagle, Khalil Amiri, Fay W. Chang, Howard Gobioff, Erik Reidel, David Rochberg and Jim Zelenka. *Filesystems for Network-Attached Secure Disks.* www.pdl.cs.cmu.edu/PDL-FTP/NASD/CMU-CS-97-118.pdf.

[7] N. Haller. *The S/KEY One-Time Password System.* IETF RFC-1760.

[8] Nelson E. Hastings. *TCP/IP Spoofing Fundamentals.* Proceedings of IEEE Fifteenth Annual International Phoenix Conference on Computer and Communications, 1996, pp 218 - 224.

[9] Jim Reid. *Plugging the Holes on Host-based Authentication.* Computers and Security, 1996, pp 661 - 671.

[10] R. Rivest. *The RC5 Encryption Algorithm,* RSA Labs' CryptoBytes, Vol. 1 No. 1, Spring 1995. http://www.rsa.com/rsalabs/pubs/cryptobytes.html.

[11] Chandramohan A. Thekkath, Timothy Mann and Edward K. Lee. *Frangipani: A Scalable Distributed File System.* ACM Operating System Principles, 1997, pp 224 - 237.

Jiro™ Storage Management

Bruce K. Haddon, Ph.D.
The Java Centers, Sun Microsystems, Inc.
500 Eldorado Boulevard, UBRM01
Broomfield, CO 80021-3400, U.S.A.
Bruce.Haddon@sun.com
tel +1-303-272-8418
fax +1-303-272-5011

William H. Connor, Ph.D.
Network Storage, Sun Microsystems, Inc.
2990 Center Green Court South, UBOL03
Boulder, CO 80803-2216, U.S.A.
William.Connor@sun.com
tel +1-303-272-8414
fax +1-303-272-8427

Abstract

The *Jiro*™ technology provides an environment intended for the implementation of storage management solutions. A product based on Jiro technology is an implementation based on the *Federated Management Architecture (FMA) Specification*, which describes extensions to the *Java™* language environment. The FMA initiative addresses system management, particularly storage management. In addition to the platform, the FMA Specification defines a component model, *i.e.*, the *FederatedBeans™* model, and a set of services. The Jiro technology is effectively the application of this model to the design and implementation of storage management solutions.

The FMA assumes a three-tier architecture for the design of storage management applications: the first or top tier is the client/presentation layer, or interaction layer with user's or systems acting as a client of the storage management application; the third or lowest tier represents the storage and related resources being managed; and the second or middle tier is that containing the logic (the programs) that define and effect the management actions required by the user upon the storage resources. It is the middle tier in which the FederatedBeans components are deployed.

FederatedBeans components are each implementations of the concept of a *Jini™* service. Each FederatedBeans component is an embodiment of some function that provides a service to other entities in the second and first tier. The success of the initiative surrounding the Jiro technology will be availability of a wide variety of FederatedBeans components from different suppliers, each providing significant functionality for the construction of storage management applications. The FMA platform supports automated communication between networked Java Virtual Machines, thus promoting applications that are federations of the constituent components.

Within FMA, the resources being managed are expected to conform to the *Common Information Model* (CIM), although provision is made for the management of resources by other means. The CIM provides modeling for all common storage system elements.

1 Introduction

Today's sciences and businesses depend on information—a vast mountain of information. As the demand for storage to hold all of this continues to grow at phenomenal rates, the management challenges are growing too [1]. In fact, the diversity of installed storage systems and the wide distribution of those systems are on the verge of creating a crisis in management, sometimes described as a "nightmare."

The amount of management that needs to be done is also increasing [2]. The cost of management is one of the significant areas of rising cost in using computer systems. Since management is a continuing cost, it has a large influence on the assessment of "return on investment" (ROI). All aspects of a system need management: software, processors, boards, options, network connections, storage devices and networks, modems, printers, and all the other pieces that are put together to make a "system." Management, in this context, includes installation, configuration, asset deployment and inventory, performance monitoring, error and failure detection, upgrade and replacement; as well as more difficult things like capacity planning, service level contracts, load balancing, all of which may be controlled by policy decisions made by the owning organization.

Because storage and storage subsystems are amongst the most complex parts of the management problem, as the storage systems often including processors, switches, and storage area networks, effective solutions to storage management problems are urgently required. Thus, the initiative surrounding the Jiro technology focuses its attention on storage management.

Developers lack standard middleware infrastructures for efficiently building capable solutions for heterogeneous management tools, applications, and services. Further complicating this situation, to provide storage solutions, developers must port their products to multiple proprietary platforms, a costly, time consuming process. The answer is based on building with software components designed for a platform specific to the purpose, being an open management platform based on Java [3] and Jini [4] technologies.

1.1 The Jiro Technology Solution

The need for the Jiro technology is acute due to a storage landscape dominated by point products that do not interoperate, creating large islands of information that are difficult to integrate, complex to manage, or that actually prohibit cross-platform information management. The intent of the Jiro technology is to bring the benefits of community source processes to the development of storage management solutions. This platform, together with basic services and a component model, brings an order to the creation of the software that can automate or add intelligence to all management functions. The elements and their relationship are defined in the Federated Management Architecture (FMA) [5].

The history of the Jiro technology activities starts with a proposal made to develop a Java language extension designed to make it easier for the developer to create new storage management applications, enable faster design cycles, lower development costs, and offer a wider market potential. It is further intended to alleviate the need to conform to multiple API's and interface specifications. Following the requirements of the *Java Community Process*^{SM} (JCP) [6], a call for experts (CAFE) was issued, followed by the formation of the Expert Group, a work group made up of representatives from a number of interested industry leaders. The Group was convened under the terms of the Process, and the FMA Specification is the result of the work of that Group.

1.2 The High Level Architecture

Three areas of specialization can be seen in the management problem:

- the representation of the resources to be managed, including the management data they contain, any behavior they exhibit, and a means of understanding their topology and other interrelationships;
- the interaction with the wishes of the user or users of these resources, including all means of invoking and scheduling management activities; and,
- the computational logic that is needed in order to translate the wishes of the users into the desired actions on, or using, the resources, or, as importantly, the translation of events taking place within the collection of resources into the presentation of useful analyses to a representative of the users.

In the diagram below (Figure 1), the third tier in reality comprises software objects, some representing the actual hardware and software being managed (solid rectangles), and some representing the relationships between them (striped rectangles). Every entity in the domain being managed, whether hardware (*e.g.*, a disk), or software (*e.g.*, a database manager), is modeled by a representative object. This view of management sited beside the elements of the data stack has been proposed earlier [7].

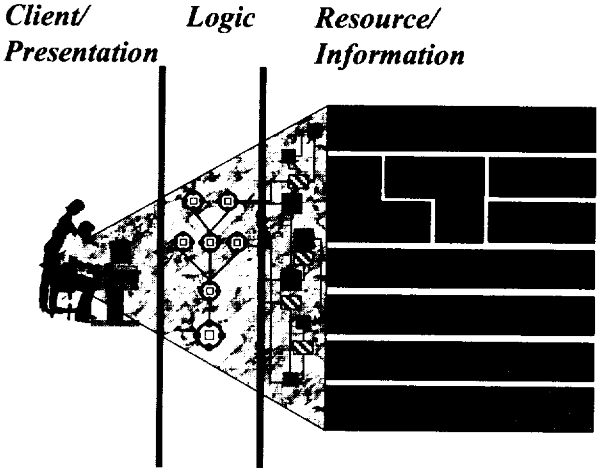


Figure 1

These are the requirements that, in other enterprise, business, and Internet applications, have led to the adoption of what are now known as "three tier" architectures. A pictorial depiction of the three-tier architecture as it applies to management is shown above.

1.3 The resource/information tier

In storage management, the resource/information tier contains both the physical hardware used to create the storage resources: the disks, tapes, disk subsystems, automated libraryies, channel interconnects, storage area networks, and so on, as well as the logical elements: the storage extents upon the media, logical disks, volumes and virtual volumes, file systems, databases, and so on. The ***management*** information contained in this tier includes the attributes of each of the pieces of storage, such as its size, capacity (which may be larger or smaller than size, depending upon reserved areas of the media, and compression, *etc.*), speed, access time, geometry, and a large number of other attributes.

Not so obviously, the management information also includes the relationships between the resources. These relationships include, for example, how the resources are connected to each other or the host, to hubs and switches involved in creating network segments, and which media are involved in the realization of a file system. The relationships are not only important to the management software for monitoring the behavior of those resources, but often are the elements most directly being managed. For example, if a particular interface card fails, it is be desirable that the monitoring management software be able to reroute traffic *via* another interface, should there be another interface available.

This third tier must be considered "active." Therefore, the representation of the resources must be able to define behaviors of the resources in a variety of ways. An example is the formatting of a disk—a common behavior, but perhaps implemented differently on different manufacturer's disks. The definition of these behaviors, taken together with the attributes of the resources and the information implied by the relationships between them, is the management interface of the resources, and hence of the resource/information tier.

In the three-tier model, the third tier represents the "state" of the system, in this case, the resources of the management system. In the case of storage resources, the state is the sum of the management data and relationship information.

There is a very critical relationship between the resources being managed and the application software. Undoubtedly, the whole object of storage management is to enable and optimize the delivery of data to those applications. Application data usually go through a number of paths and transformations in moving from "raw" bits on the storage media, to the form in which they are presented to the application and even further transformations in order to present information to users. Therefore, "behind" the resource representations as seen by the management logic, there are layers of hardware and software responsible for these data deliveries and transformations. Figure 1 also shows this.

1.4 The client/console

In the first diagram, the client/console tier has been represented by a system administrator or installation manger sitting, literally, at a console. This depiction is only meant to be representative. Certainly one of the objectives of enterprise-class management systems is to bring relevant monitoring information to some central place, where the "big picture" may be evaluated effectively and efficiently.

For an automated management system, it is not sufficient that operations be initiated from a console. It is necessary to be able to initiate actions from CLI's, scripts, periodic schedulers, and components of the management system itself. This latter facility is required so that when decisions are programmed into the management system, those decisions may be used to initiate appropriate actions, without the real-time intervention of an administrator (which is the definition of "automated management").

1.5 The management logic tier

This tier is intended to contain the actual programs that are the applications implementing the logic needed to perform management decisions and management functions.

Such applications typically contain a number of modules, architectural "sub-applications," that deal with various aspects of the overall tasks to be performed, by doing an individual task, or computing a specific result, and so on. Component models, based on object-oriented methodologies, and usually based on specific object implementations, are a way of describing these modules. The FMA specifically defines such a component model, *i.e.*, FederatedBeans.

All component models, in essence, define a "platform" and a set of "services" in addition to the component model itself. A number of other things also normally surround the successful use of the component model for real implementations, which includes some or all of an interface specification language, an intercommunication protocol, a deployment methodology, and various tools to support the creation and use of the component model, which together constitute a software development kit.

However, more than each of those, a successful component model is one that leads and encourages implementations of actual, real, useful components, that is, a library of components, which implementers may use without further development.

The largest development leverage lies in being able to obtain significant components out of which an application may be built for appropriate amounts of money, while capping the time and effort required to learn how to use and to support them (when deployed). The other effect is that the components may be common to more than one application. Thus a component has only to be deployed once, the footprint costs are paid once, and many applications can share the use of the component. Such components take on the nature of additional services, but without requiring a new definition of the *core*.

So, by analogy, once a platform based on Jiro technology is available, then the objective of on-going activities is building useful applications using components, and, as experience is gained, determining which of those components are candidates for libraries of re-usable components. Such candidates will be evaluated, and perhaps re-engineered, for standardization, interchangeability, substitutability, *etc.* Attention and growth in this area will eventually lead to a classification of components, and a catalog from which developers will be easily able to choose the components most suited to their current needs.

1.6 Generation of management applications

The usefulness of the three-tier architecture is related to the decoupling that occurs between the tiers. For the three-tier approach to be valid, the degree of decoupling between the tiers should, in general, be greater than the decoupling introduced by the problem or task decomposition used within the tiers.

The most usual method to present one tier to another is through API's, and most particularly API's that are wrappers for protocols. From the point of view of a lower tier presenting itself to an upper tier in the form of a particular API, the requirements of that API define a framework. That is, it requires that certain conventions be observed so that functions may be called, results passed back, call-backs created if needed, and so on.

Provided that these API's or protocols exist, the internal structure of each tier may be driven by the needs and requirements of that tier. Consequently, the overall philosophy of the Jiro technology has been to use the "best of breed" examples for each tier. The FMA proposes the technology for the second tier, and encourages support of the DMTF CIM for the third tier. Work is continuing on defining interfaces to the first tier.

1.7 Requirements of the Third Tier

There are many requirements, in detail, for this tier. The following may be considered just the major requirements:

- a software-accessible representation of the each of the actual physical resources that are part of a computing system, including all the usual asset information (*e.g.*, manufacturer, type, size, *etc.*);
- a software-accessible representation of the system resources made available by the existence of these physical resources (*e.g.*, the storage extent actually made available by the installation of a particular storage device);
- a representation of the relationships between resources of both types (*e.g.*, this disk is part of this RAID subsystem, this file system is implemented on this stripe of these disks), and the ability to update these as the system evolves, or fails;
- an ability to recognize that various resources are implementations of the same class of resource, as well as being able to identify and take advantage of specific differences (*e.g.*, a laser printer is a (generic) printer, but is capable of duplex operation);
- a reasonable ability to find out what resources are available (*e.g.*, this particular host has a tape drive, whereas some other does not); and,
- agreement on the names and meaning of various critical attributes and behaviors, with support for being able to find the resources represented by means of queries based on those attributes.

1.8 The CIM Specification

The *Common Information Model (CIM) Specification* [8] is a description of an object model, and of a language in which to describe the classes and the instances of objects of that model. This particular object model has, as do many other object models, rules of inheritance and the overriding of methods and properties. In particular, the inheritance is single inheritance, and overriding is explicit.

Nominally, there is only one kind of object in the CIM, but it is more instructive to think in terms of two principal kinds of object. There are those objects that represent the actual entities being managed, which are those needed to satisfy the requirements above; and, those objects that represent the relationships between those entities (providing a way to satisfy other requirements, above). These objects are called *associations*. For example, a network hub and a cable of the network are each entities. The cable will have a relationship to the hub, which can be described as "connects to." The "connects to" object contains a reference to the cable as being the source ("antecedent") of the relationship, containing also a reference to the hub as being the target (the "dependent") of the relationship. Of course, associations may be one-to-one, one-to-many, many-to-one, or many-to-many.

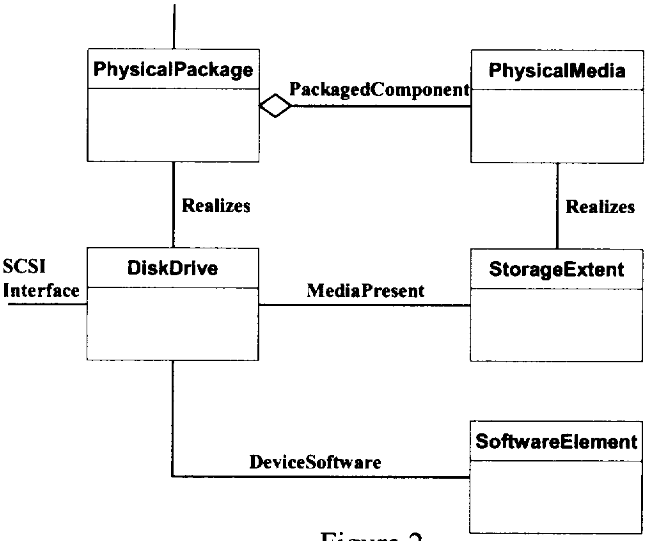


Figure 2

The diagram above (Figure 2) shows a typical model of a hard disk drive, in its own packaging. It shows the both physical and logical elements, and the relationships between them, these being the associations illustrated as labeled lines. It should be understood that each of the associations in this diagram is itself a CIM object, and thus may be queried and interrogated in the same manner as the objects representing the system elements.

1.9 The common XML protocol (WBEM)

While there is considerable value to the concept of implementation independence when defining a common information model, a plethora of implementations is not something with which developers of management implementations wish to cope. There is, however, a middle ground permitting *Object Manager* (OM) implementers freedom, but also allowing use by applications without having to re-implement their code for interacting with the OM. This middle ground is based on having a common protocol to be used to communicate between the application and the Object Managers.

The DMTF has standardized a protocol for this purpose. The content of the protocol is based on XML [9], and the use of HTTP as a transport mechanism is also defined [10]. While it is possible to use the XML "documents" directly to invoke actions upon the OM, the use of the HTTP binding allows the XML payloads to be transmitted across the Internet, and, where permitted, through firewalls. The XML format provides also for conveying results of method invocations on the Object Manager back to the OM client.

The combination of a CIM Object Manager implementation with the HTTP-transported XML protocol packet for OM operations, is "Web-Based Enterprise Management" (WBEM) [11], which is the platform- and language-independent technology for using CIM and CIM Object Managers. The diagram below (Figure 3) shows the WBEM model for the use of a CIM OM, where the communication between a client and the OM is by means of the defined XML-based protocol, and the CIM OM obtains information about the managed objects by means of privately defined code elements called "providers."

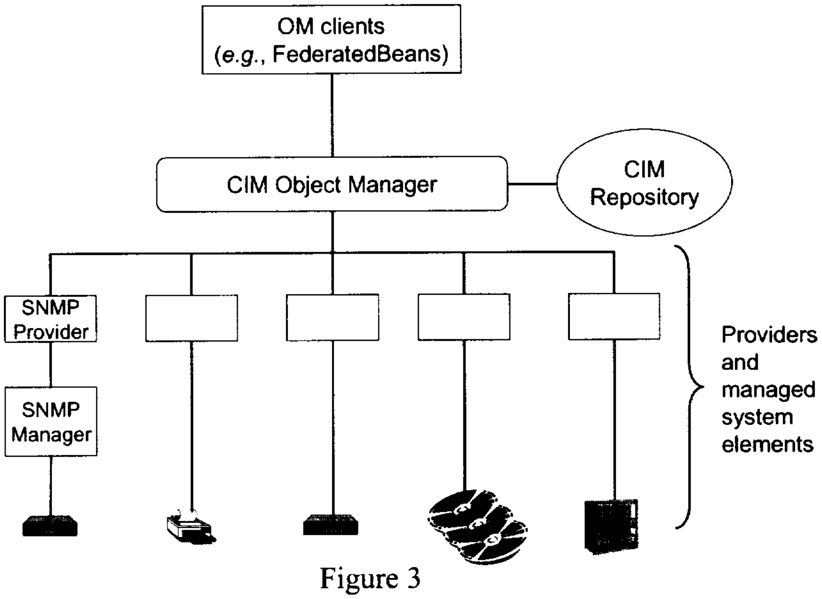


Figure 3

The existence of a CIM Object Manager implemented using the Java language is an important development. It means there can be a CIM OM available everywhere that the Jiro technology is available, with no further investment in the development of the OM. This is undoubtedly a significant advantage when developing management applications.

1.10 The requirements for the Client/Console Tier

As mentioned above, this tier is not specifically a target of the current Jiro technology activities. However, for some completeness, the requirements of this tier are outlined here, with some description of the considerations. The requirements include at least:

- the ability to interact with users, either in terms of graphical user interfaces, CLI's, and perhaps more than one means of scripting the interaction;
- support for remote communications when the user must be able to access the management system from other than a fixed location. Most newer management systems contemplate the use of the Internet as a means of access (see next item);
- provision for gathering information to enable authentication of the user, in order to be able to use the security features provided by the second and third tier;
- the possibility of supporting interfaces to existing management solutions, since there is already a great investment in these. The first tier could provide a good proportion of the bridging between such legacies and newer Jiro technology-based solutions;
- effective means of navigation for each of the user interfaces supported. Most computing systems, particularly those supporting storage area networks (SAN's) and similar large and complex subsystems, require means of showing them that can be mapped to and from the model representations that make sense to the user;
- presentation technologies, either screen or print based, to show the navigation and other views, upon demand; and,
- the capability of launching management applications into the appropriate environments, with, preferably, the ability to monitor the status of those applications.

There exist many "console" and client-side solutions, offered as complete products by many companies. In addition to the actual first tier functionality, these same companies often offer modules that provide second and third level functionality. These product offerings are sometimes embedded in other services, including monitoring, notification and response management, and sometimes even "first response" field engineering.

All of the above services remain relevant in a Jiro technology-based system. These systems offer the promise of even more capable applications, and hence even more desirable value-added services.

2 The Management Logic Tier

This is the tier specifically addressed by the FMA Specification. The Specification defines a component model for the development of management applications (programs that implement storage management). Because the Specification contains all the needed detail about this tier, only a brief overview is offered here.

2.1 Requirements for this tier

The question is often asked, given that the third tier contains so much capability, why is there a need for the second or "logic" tier. The requirements for the second tier include at least the following, each going beyond what is (conveniently) possible in the third tier:

- a component model (more detail below), in order to be able to create a library of solutions to be used as construction elements for new management applications;
- support for distributed applications, in order to support scalability (*e.g.*, use of more than one processor), efficiency (placing logic near to its source of information), enterprise capability (separation of management environments along organizational lines), and redundancy (to support applications utilizing high availability capabilities);
- deployment of components in standard ways, so that packaging is done once, the deployment problem needs not be re-solved for each release of each application and for each implementation of the second tier;
- basic services, that are needed by all applications, inclusive of component location and loading, logging, scheduling, *etc.*;
- control arbitration, whereby a component can claim sole access to a second or third tier resource, and then act as a "gatekeeper" for allowing appropriate access (*e.g.*, a classic multiple reader, single writer regime);
- an ability to ensure consistency across resources, even when those resources are in different environments (*i.e.*, namespaces), such as a remote disk mirror;
- the ability to compute logic across resources, in a similar way, such as switching between remote disk mirrors;
- a facility to share logic implementations between clients, by allowing multiple clients to re-use (or multi-thread, if appropriate) the same logic components;
- the ability to compute logic across time, by accumulating historical information about present and past states of the resource/information tier, and performing appropriate (*e.g.*, statistical) analyses; and,
- the definition of higher level of management abstractions, with appropriate interfaces (*e.g.*, a charge-by-usage service of a virtual disk system across the Internet).

2.2 Jiro Technology as the second tier

The Federated Management Architecture has been specifically proposed as a means of structuring the second tier. It addresses specifically the following characteristics:

- the ability to dynamically and easily introduce new behavior while the system is in operation;
- the necessary locking to support control arbitration (as described above);
- support for wide-spread consistency across management applications;
- transactions across arbitrary parts of the management state;
- the possibility of a single packaging, that support "talks to" relationships between components written by different authors, and a universal "runs on" relationship to the Jiro technology (thus fostering neutrality with respect to physical platforms);
- fine-grained security, in that the security context is carried and made available to all components in the distributed system;
- source available under community agreements;
- provision for the support for higher availability solutions; together with,
- support for adequate scalability of management solutions; and,
- versioning, to provide a methodology for not having to update an entire universe of solutions at one instant.

The resource/information layer (3rd tier) models that which is being managed (systems, storage, networks, *etc.*). The model includes all the manageable attributes and behavior of the resource. These attributes and behaviors of the model are "static" in the sense that they are in one-to-one correspondence with the attributes and behaviors of the real-world resource being managed. These attributes and behaviors of the model should not be changed (even if the underlying implementation technology would permit it) to represent anything other than the attributes and behaviors of the real-world resource. It is not expected that the "external logic" of a managed resource should change in any significant way during its lifetime. The resource/information layer is *not* static in the sense of being unchanging—resources are expected to come and go, being replaced, upgraded, and extended, in the normal course of the system lifecycle.

The logic layer (2nd tier) reflects the pattern or structure of the decisions that need to be made in order to manage the resources. These decisions will be made based on information that is corralled and collated from the data present in the information model. Since management is undertaken in order to meet (organizational) goals, the nature of the decisions, and the behavior based on those decisions, needs to change as the goals change or evolve. Thus, the objects (or components) of the logic layer need to be replaceable with new versions, which, behind the same API, implement new behaviors. The logic layer also needs to be "dynamic" in that an object or component may be introduced into an execution environment where none has existed before, thus defining new behavior.

2.3 The FMA Specification

The fundamental notion supported by the FMA is that of components, where it is intended that these be the unit of assembly and installation of management logic. The component model consists of a set of naming and construction rules; with these components being termed "FederatedBeans." FederatedBeans components are based on the Java object model, and conform to the set/get conventions of *JavaBeans*™ [12]. To support assembly, it must be possible to discover ways in which management components can be connected to one another in both anticipated and unanticipated ways. The components have to find the appropriate interface offered by other components in order to create a coherent application. As illustrated in the diagram above (Figure 4), the connections between the FederatedBeans components may be an arbitrary topology (not necessarily hierarchical), and a given service might be used both directly and indirectly. In many programming environments, the choice of interface is made at program writing time. An effective component model will allow these to be found at installation and/or execution time. A component may present different interfaces for different purposes, or even just to provide the same functionality in more convenient forms.

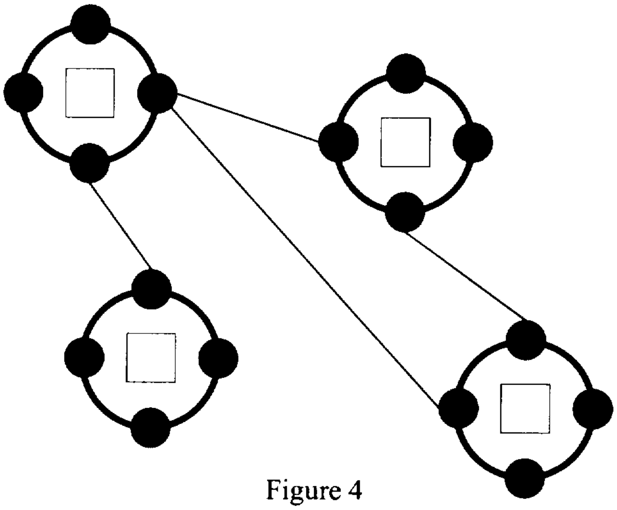


Figure 4

Because the FMA platform supports a distributed programming environment, most frequently it is not the actual interface that is the point of connection between components. The point of connection is a *proxy* [13] for that interface. The proxy is always local to the using component (*i.e.*, present in the same *Java Virtual Machine* (JVM™) [14]), and the component (or object) for which the proxy is acting may be in the same virtual machine, or other virtual machine accessible within the domain of the application.

Deployment: It must be possible to deploy, or install, components in a standard manner on a running system. Deployment includes installing class files, resources, components, and objects.

Controllers: An important objective of the Jiro technology is providing the infrastructure to support control arbitration. Controllers attempt to control resources through components. Resources, therefore, may be subject Jiro technology providing the access mechanism to support control arbitration. The primitive required for arbitration is called the controller *aspect* of the management services model, and this in turn must support durable (long term) exclusive locking of resources.

Transactions: Most distributed component models provide some form of transaction support to aid in protecting the integrity of the resource/information layer [15]. The transactions provided by the Jiro technology are focused on supporting large numbers of heterogeneous resources, rather than a single large resource (*e.g.*, a database), and not necessarily large numbers of clients. A FederatedBeans component needs a transaction aspect to participate in a transaction.

Security: A management environment must support validating clients for actions that they attempt to take, since such actions may have far-reaching results. The basic model is that of the Java model [16], but with provision made for ensuring that necessary security information is transmitted between Java machines as needed. Access to this information requires a security aspect.

Logical Threads: As the FMA is intended to support active, autonomous, management applications, components must be able to support concurrent and re-entrant conditions with respect to threads. Management applications are made of distributed components, so the FMA introduces the concept of a logical thread that spans processes, and in particular, is capable of spanning execution threads in different virtual machines [17]. Thus, component behavior with respect to threads may be specified with respect to *logical* threads instead of just the provided Java language threads.

2.4 The Use of Jini Technology

Jini technology is used within the Jiro technology for a number of purposes. It is used to discover federated Java virtual machines (termed *stations*), which are the active component of the Jiro technology, and supports addressing domains within a federation of stations, as the transaction manager (including *leases*), and for a variety of lookup operations, including discovery of CIM Object Managers, and various other components, interfaces, and services running on those platforms.

2.5 Other Basic Services

The basic services of the FMA include those provided by the Jini environment, plus logging, scheduling [18], and so on. Services are regarded as "basic" within the Jiro technology if they are assumed present on every platform. The criterion for regarding a particular service as being basic is usually the need for it to be pervasively used throughout management applications.

"Services" that are not pervasive may be supplied by components, and may be discovered (see "discovery", as discussed under "The Use of Jini Technology" above) when needed.

2.6 The use of components for filling out services and functionality

As hinted at in the previous paragraph, a significant use of FederatedBeans components is providing additional functionality and services on a Jiro technology platform, without need to define an extension to that platform. As the use of this platform matures, it is expected there will be a large number of service components supplied by interested parties.

Should the use of any of those services become so frequent as to fulfill the "pervasive" criterion, consideration could be given at that time to re-awakening the community process. to modify the FMA Specification, defining an *extension* to the architecture.

2.7 Five stakeholders in the value proposition

Five broad classes of "stakeholder" in the Jiro technology may be identified. A "stakeholder" is a person or user that has something to gain, or lose, by use of the technology. In the following sections, each of these stakeholders is identified, and their "stake" described.

Stakeholder—the Resource Vendor

The resource vendor is the manufacturer of such things as disks, tape drives, storage subsystems, software products, and so on.

Hardware and software vendors offering products in the range of a few tens of dollars to thousands of dollars (US) face strong competitive pressures with thin profit margins. Vendors in this space typically produce 10^5 to 10^7 devices per year at very low cost and profit margin. Example of device retail costs (at this time) include:

Device	*Market Price*
CD-ROM (40x)	$29
8 GB Tape Drive	$59
10 GB Disk Drive	$121
Celeron Computer	Free or $399

Products in this price range are extremely price sensitive as consumers often care little about brand name or quality and will often purchase the lowest price product. Any additional cost to support manageability is unacceptable. Vendors in this arena can reduce distribution costs by providing any software (on floppies or CD's) already bundled in the box, their support software with management application vendors software, or by Web download only. Vendors can participate in the management arena by simply developing a CIM provider for their device, a once-only development cost.

Vendors producing products on the low-end of cost and profit will benefit from Jiro technology in several ways:

- the vendor can play in the CIM/WBEM world at a very low initial cost and also be managed in the Jiro technology-based world as well. In Microsoft Windows, the OS where most commodity hardware is installed, the vendor develops a CIM provider. It is then supported in the Microsoft Management Console (MMC). This first step allows the device to be managed by exposing all the device's "knobs." This allows the device to be managed at a higher level by intelligent FederatedBeans components in concert with other devices and services;
- a large demographic of the customer population needs manageable devices and will be swayed in their purchasing decision by the device's integration with the FMA. The cost of developing the support can be amortized over a large number of devices and the cost of manufacturing the software for inclusion with each device is low;
- in a two-step development lifecycle, the vendor can "get into the game" with the low entry cost of developing the CIM provider, perhaps giving away the management

software, and later developing the intelligent FederatedBeans components that manage the device in the most efficient and effective manner possible; and,
- for very little investment, the vendor can provide the kind of functionality and manageability that was previously available in devices costing 10× - 100× as much by leveraging the Jiro technology infrastructure.

The benefits for the high-end vendor are very similar, but with the added benefit of the vendor probably wanting to, and being able, to provide a FederatedBeans solution. This may provide special control or understanding of the larger subsystem (including, perhaps, a "contact the support center" function that implements a 24 × 7 maintenance policy that requires no intervention by the customer). The advantages are:
- the CIM technique allows the description of sub-systems of arbitrary complexity;
- the FederatedBeans components approach allows a vendor to supply system-specific components that may be also utilized in other management products; the manufacturer does not have to develop a "complete" management system in order to enable the one or two essential features needed for the added value of their product; and,
- The FederatedBeans approach can ensure that the sub-system appears on management consoles in a way that the manufacturer wishes (together with that manufacturer's own appearance and message).

Stakeholder—the Component Vendor

Much of the success of Jiro technology will be in the existence of an active market in Jiro components, *i.e.*, software components that can be used as building blocks in the creation of storage management applications. The leverage of this approach is that developers of management applications can recast their work in terms of integration of components, rather than the design, development and testing of every component needed to make a complete application.

In many other parts of the software industry, software components have become a very successful, and necessary, part. Software components can take the form of source and binary libraries, dynamically loadable libraries, shared objects, DCOM [19] components, foundation component sets, as ActiveX components and Java packages and applets.

There are two approaches to the use of components: by those wishing to provide specialized components, such as those described above, where specific and dedicated functionality is provided as a component behind standard interfaces, and by those providing general functionality in components with interfaces that extend the range and capability of the entire system.

Examples of this latter type of component could include:
- a health monitor, that collects the values on certain attributes, and delivers warnings when any of these values move outside predetermined limits;
- directory and lookup services, powered by various difference sources of information, *e.g.*, DNS, or by different access standards, *e.g.*, LDAP;
- an asset manager, that integrates what is installed (visible to the component) with an enterprise inventory system;

- event handlers, that do correlation, in order to deduce the root cause of an event storm, *e.g.*, failure events from many routers about not being able to reach certain hosts may all be due to a power failure on just one segment of a network;
- time series analysis, given various observations of some measure at known time intervals, so that future values may be predicted;
- virtual volume tuner, that uses performance statistics from the virtual volumes to adjust the behavior of real disks and their interconnections to improve the performance of the virtual volumes;
- a database configurator, that given a set of parameters about the intended use of a database, can configure virtual volumes, table layouts, and other controls, in order to either enable the intended database use, or to optimize behavior;
- a capacity planner, which not only can assess what is installed, but may also be able to reach product information on manufacturer's web sites, so that an upgrade plan can be derived by playing "what if";
- a storage area network tool, that analyzes a topography of a traffic pattern, and advises on the addition or movement of existing network access points in order to balance use of the network segments; and so on.

The "value" of components may be in their intrinsic value, and thus be traded and sold "off the shelf", like many other applications, or in the value that they enable in other equipment (so-called "drag"), where the software is essentially given away, in order to improve sales of the equipment.

Stakeholder—the Management Application Vendor

For those developing management application, the advantages are:

- that FMA enables developers to build applications with advanced, automated functions that realize the goal of managing storage or storage networks, where many of those automated functions may be obtained "off the shelf";
- that FMA provides the FederatedBeans model that enables interoperability among diverse applications, services, and devices, and is also an aid in the architecture and design phases of the application;
- relieving the application developer of the necessity to design and implement the means of accessing the management information of devices, and for these to be easily added, removed, or provisioned for service; and,
- reducing downtime by enabling automatic updates to applications or services.

If a new application or device is Jiro technology-enabled *and* a management component vendor has a FederatedBeans product implemented, the new application or device will be immediately capable of being incorporated into the management environment. For most devices, being WBEM-enabled will be sufficient for Jiro enablement. From the customer's point of view, the new equipment will be capable of being managed (at a higher rather than lower level) and reported a standard manner. A Jiro-enabled disk array, for example, would be able to report capacity, which could then be used by applications such as capacity planners. The fact that the disk array is there and has been recognized means that volume managers can take advantage of it automatically—rather than having to be told its whereabouts.

Stakeholder—the Information / Data Application Vendor

Information and data application vendors can improve the performance of their applications by being able to interact directly with the management components of the data storage system. For example, if a the data application is a backup suite, the implementation of that suite could:

- use the management system to discover which files need backing up, without having to directly use the file system interfaces; this reduces porting costs in development;
- use a propriety interface to set up the backup application (it would be possible to develop a CIM interface, but an intermediate solution would be to use existing CLI's *via* a specially developed facade);
- similarly, proprietary interfaces might be used to collect information, say from a log file, which could be used as key for the generation of events that a FederatedBeans component could use to report upon the status of the backup; and,
- since the CIM model includes objects for the management of tape libraries, the management of the tape pools could be integrated with the backup application to ensure correct rotation of tapes, and the observation of the correct policy rules for the keeping of tapes in the rotation.

Stakeholder—the Customer

The CIO: the Jiro technology, the FederatedBeans components, and the basic services define a baseline against which management and data applications can be measured. The CIO is assured that an Jiro-enabled product meets basic requirements for interoperability with other applications. Further certainty may be obtained by insisting on management applications that have been Jiro *certified*, and by ensuring that the producer has participated in interoperability tests with other products.

Over a period, by invoking careful acquisition policies, a CIO and the IT Department may build a more fully integrated set of management capabilities by looking for Jiro-compliant applications.

The System Administrator: From the point of view of the system administrator, the acquisition of Jiro-based applications, and value-based FederatedBeans components:

- minimizes barriers for providing management of many hardware/OS platforms; eliminates or minimizes platform porting, enables solution developers to support platforms that may have a lower priority in the company's target market;
- provides broad device support: any device with a WBEM provider or supporting SNMP can be managed through a FederatedBeans component; and the support of private interfaces allows management of non-WBEM and non-SNMP devices;
- enables a finer application granularity: components allow users to pick and choose Jiro-enabled applications and 3rd party FederatedBeans solutions rather than others;
- brings management capability from a storage-specific console or an enterprise management console: The Jiro 3-tier architecture separates management logic from user interface and avoids mandating particular user interface solutions.
- developing through component design and assembly allow the user to take more ownership of policy and automation.

3 Related Work

The following are possible choices, among others, to implement a second tier in a management system:

- *Enterprise JavaBeans*™ (EJB™) [20] component architecture is designed to be the most capable technology for second tier "business logic," providing single threads of logic execution that normally originate in the client tier, and transactions that are usually with respect to a single third tier database. The FederatedBeans model is suited to the creation of management solutions as it more naturally supports thread concurrency, makes specific provision for the support of a CIM-based third tier; and has the ability to support arbitrary transactions with respect to that third tier;
- WBEM, which is the preferred choice for the third tier, could also to provide an object model and schema for the second tier. Further work would be required, as the CIM Schema would have to be endowed with the appropriate new objects or extensions. Even then, it would remain a "double technology" for implementation, *i.e.*, one technology for the definition of the objects, and another (platform dependent) for the definition of methods. To be completely capable for utilization in the second tier, WBEM would also need to be given a component model that addresses the same issues as listed for the FederatedBeans model;
- CORBA [21] appears to be an appropriate choice, but has had limited use in the implementation of management applications. Its success in business logic does not argue for success in the management arena. CORBA objects are still platform-specific, thus creating a "porting" problem, even though there are no impediments to inter-platform communication.

It must be feasible to choose the first and second tiers independently. In order for there be a choice, the second and third tier technologies must decoupled by an appropriate choice of interfaces between the tiers. Jiro technology appears to be the appropriate choice.

4 Future Work

By the time this paper is published, a reference implementation of the Jiro technology, implementing the FMA, should be publicly available. It is also intended that by that time a number of the original Expert Group participants will have also applied the FederatedBeans concept to the production of a useful number of components, that the interoperability of these components will have been demonstrated, and their usefulness in creating management solutions be in the course of evaluation.

As further FederatedBeans components are developed, some will be found pervasive enough to be "basic" in the sense of the FMA Specification. At that time, the Expert Group could reconvene to integrate candidates into revisions of the FMA Specification.

5 Conclusions

The development of an accepted and useful architecture for the building of management applications marks an important turning point in the arena of storage management. The FMA, and Jiro technology represent both merging and emerging developments making further advance possible. The realization of this architecture in real products is the next major objective.

Thanks

The authors wish to thank, particularly, the members of the FMA Expert Group for the efforts that went into creating, editing, and revising the FMA Specification. It is from this foundation that the Jiro technology has developed into a viable storage management solution. The authors also want to thank the session chair for the efforts he has marshaled, including the anonymous reviewers, to correct and revise this paper. As always, any final errors and omissions are the responsibility of the authors.

References

[1] Shiers, Jamie: “Massive-Scale Data Management using Standards-Based Solutions,” ***Proc. 16th IEEE Symp. Mass Storage***, San Diego, CA, IEEE Computer Society Press (March, 1999).

[2] Coyne, R.A.; and Hulen, H.: “An Introduction to the Mass Storage System Reference Model, Version 5,” ***Proc. 12th IEEE Symp. Mass Storage***, Monterey, CA, IEEE Computer Society Press (April, 1993).

[3] Gosling, James; Joy, Bill; and Steele, Guy: **The *Java*™ Language Specification.** Addison-Wesley, Reading, Massachusetts. ISBN 0-201-63451-1 (1996)

[4] Edwards, W. Keith: **Core *Jini*™.** Prentice Hall PTR, Upper Saddle River, New Jersey. ISBN 0-13-0114469-X (1999).

[5] FMA Expert Group: **Federated Management Architecture Specification, Draft Version 1.0.** Sun Microsystems, Inc., http://www.jiro.org/specs.html. (January, 2000).

[6] Sun Microsystems, Inc: **The *Java*™ Community Process.** Sun Microsystems, Inc., http://java.sun.com/aboutJava/communityprocess/. (May, 1999).

[7] IEEE Storage Systems Standards Working Group: **Mass Storage System Reference Model, Version 5.** http://ssswg.org/public_documents/MSSRM/V5toc.html (September, 1995).

[8] Technical Development Committee: **The CIM Specification, Version 2.2.** http://www.dmtf.org/spec/cims.html, Distributed Management Task Force, (June, 1999).

[9] DMTF XML Working Group: **CIM XML Mapping, Version 2.0.** http://www.dmtf.org/XML/CIM_XML_Mapping20.htm, Distributed Management Task Force, (June, 1999).

[10] DMTF XML Working Group: **The CIM HTTP Mapping, Version 1.0.** http://www.dmtf.org/XML/CIM_HTTP_Mapping10.htm, Distributed Management Task Force, (June, 1999).

[11] Technical Development Committee: **The WBEM Initiative.** http://www.dmtf.org/wbem/, Distributed Management Task Force, (1999).

[12] Englander, Robert**: Developing Java Beans.** O’Reilly and Associates, California. ISBN 1-56592-289-1 (1997).

[13] Gamma, Erich; Helm, Richard; Johnson, Ralph; and Vlissides, John: **Design Patterns: Elements of Reusable Object-Oriented Software.** Addison-Wesley, Reading, Massachusetts. ISBN 0-201-63361-2 (1994).

[14] Lindholm, Tim; and Yellin, Frank: **The *Java*™ Virtual Machine Specification.** Addison-Wesley, Reading, Massachusetts. ISBN 0-201-63452-X (1996).

[15] Bernstein, Philip A.; and Newcomer, Eric: **Transaction Processing.** Morgan Kauffman Publisher, Inc, San Francisco, California. ISBN 1-55860-415-4 (1997).

[16] Oaks, Scott: ***Java™* Security.** O'Reilly and Associates, Sebastopol, California. ISBN 1-56592-403-7 (1998).

[17] Haddon, Bruce K. and Connor, William H.: "Software for Distributed Monitor Concurrency Control," Patent Pending, U.S. Patent and Trademark Office (November, 1998)

[18] Haddon, Bruce K.: **Machine-Independent Real-time Operating System Interfaces**. Ph.D. Thesis, Department of Computer and Electrical Engineering, University of Colorado, Boulder (1979).

[19] Eddon, Guy; and Eddon, Henry: **Inside Distributed COM.** Microsoft Press, Washington. ISBN 1-57321-849-X (1998).

[20] Valesky, Thomas B.: **Enterprise JavaBeans™.** Addison Wesley Longman, Inc., Massachusetts. ISBN 0-201-60446-9 (1999).

[21] Joint Revised Submission: **CORBA Components.** Object Management Group, Inc. (1999)

THIS PAPER AND ITS CONTENTS ARE PROVIDED WITHOUT ANY WARRANTIES, INCLUDING ANY IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT. IN NO EVENT WILL SUN BE LIABLE FOR ANY CLAIMS FOR DAMAGES, HOWEVER CAUSED AND REGARDLESS OF THE THEORY OF LIABILITY, ARISING OUT OF OR RELATED TO THE USE OF THIS PAPER OR ITS CONTENTS.

The technology and products described in this paper may be protected by one or more U.S. patents, foreign patents, and/or pending applications.

A blueprint for Representation Information in the OAIS model

David Holdsworth, Derek M. Sergeant
The Cedars Project (ISS department)
The University of Leeds
LS2 9JT, UK
D.Holdsworth@leeds.ac.uk, D.M.Sergeant@leeds.ac.uk
tel +44-113-233-5402, +44-113-233-5698
fax +44-113-233-5411

Abstract

The CEDARS* project within UK academia seeks to develop a demonstrator system to recommend techniques for long-term storage of digital data primarily within the research library context. The intention is that this demonstrator will conform to (the spirit of) the OAIS (Open Archival Information System) model [1], and put some flesh on the model's generic bones. This paper describes our current scheme for representation information.

1 Overview

We are firmly of the opinion that data outlives the medium upon which it is recorded. Technology obsolescence is a bigger threat to data retention than is media drop-out. For example, failure to read a reel of 7-track magnetic tape is more likely to be due to non-availability of the tape drive than drop-out on the medium.

However, a bit-stream can be preserved indefinitely [2,3].

Our blueprint provides for transforming of a digital object into a bit-stream preserved indefinitely, and for subsequent access to the intellectual content of the original digital object. We believe that the approach is valid for plain ASCII (American Standard Code for Information Interchange) text files, or for emulation of preserved computer systems, and for a whole host of situations in between.

2 Ingest

We propose a 2-stage process of ingest which has the steps:

- First step is separation of the data from the medium
- Second step is to map to a bit-stream (i.e. make the *data-object* part of the AIP).
- Followed by preservation of the bit-stream (i.e. keep the AIP in an archival store).

The form of the data between steps 1 and 2 is the *underlying abstract form* (UAF - see below). These two steps are the parts of *ingest* process that involve the data itself. We find that the UAF concept enables us to structure the representation information, and to build representation nets which facilitate evolution with emerging technology.

This paper does not address the other meta-data aspects that must also comprise part of the ingest process.

* CURL (Consortium of University Research Libraries) exemplars in digital archives

3 Information Package

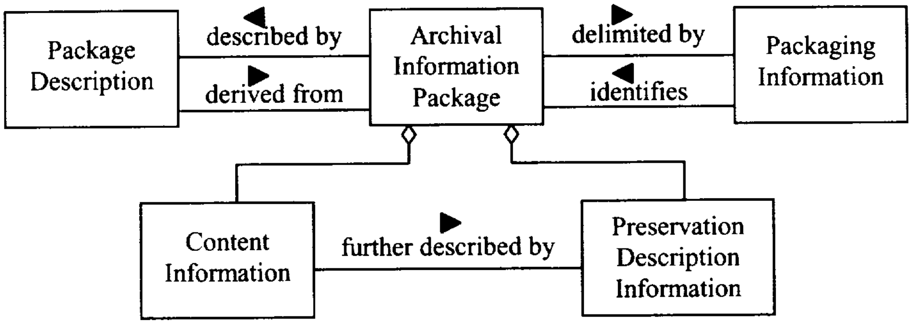


Figure 1. OAIS fig4-16: Archival Information Package (AIP)

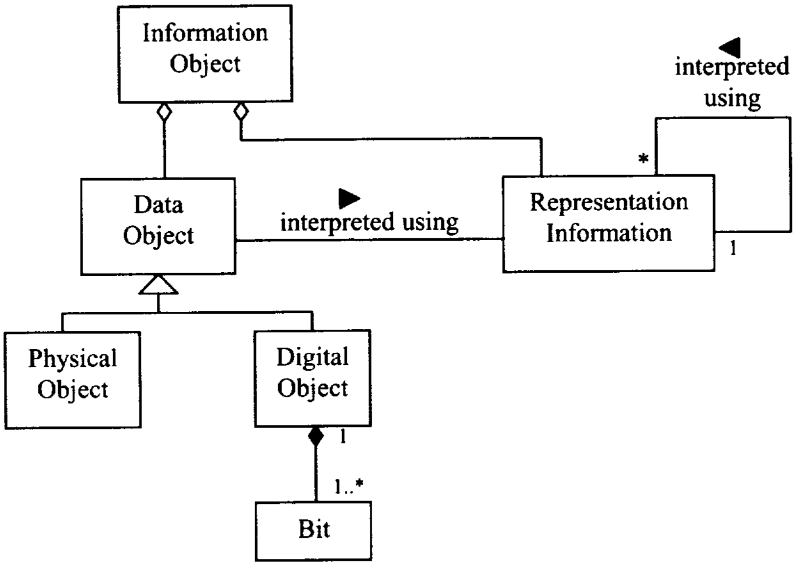


Figure 2. OAIS fig4-9: Information Object

We are concerned here with generating the representation information for inclusion within the content information component of an archive information package (AIP) (see fig 1 and fig 2 taken from the OAIS model). Each component of the AIP is packaged in a formalism appropriate to the data being stored, using an ASN.1 (Abstract Syntax Notation) wrapper to package the components together as an AIP:

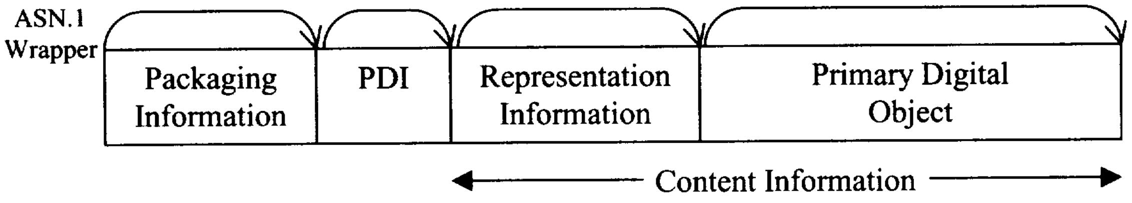


Figure 3. The package structure of an AIP

The packaging information is a CRID (Cedars Reference ID) (discussed later in this paper) which provides a version number for the internal structure of the AIP and references a specialised AIP whose purpose is to document the data format description (DFD) of each component. For example this latter AIP would provide an XML DTD for the PDI (Preservation Description Information) component, and a specification of the Java Properties used in the RI component.

The purpose of the *representation information* (RI) is to enable access to the preserved digital object in a meaningful way. Java property files are a simple formalism to store the information needed to do this. Their use integrates well with our demonstrator software written in Java, and yet the format is simple enough to admit of a brief description. As the Cedars' demonstrator evolves the RI will be migrated into an XML formalism which provides hierarchical structuring and also integrates well with our Java implementation.

Enabling meaningful access to the preserved object includes such processes as recreating the experience of viewing (or even interacting with) the original, or analysing it for a concordance. For particularly complex objects, emulation is likely to be involved.

At its most basic, the RI enables a reversal of the ingest process to deliver a copy of the original. This observation provides a useful minimum criterion for testing the acceptability of any scheme for RI and also for the RI of any particular information object. If this test is performed alongside the ingest process then a critical comparison of the copy produced by ingest reversal and the original can be made. This suggests that the standards for ingest should require that:

> The representation information must allow the recreation of the significant properties of the original digital object, if one assumes that appropriate hardware technology is available.

4 Significant Properties

Whoever takes the decision that a particular digital object should be preserved will have to decide what properties are to be regarded as significant. The submission agreement could usefully specify a list of significant properties. In a library context, the decision to preserve is taken by the collection management activity [4,5,6].

5 Underlying Abstract Form

We use the term *underlying abstract form* (UAF) to encapsulate the recognition that the data has an existence and a content separate from the medium upon which it is written. This underlying abstract form contains all the significant properties of the data, and is independent of the medium upon which the data is written. Any given digital object is likely to have a number of possible UAFs. Choice of the UAF for preservation is part of the ingest process (either in the *Receive Submission* box or the *Quality Assurance* box in fig 4 taken from the OAIS model). (There may be some difficulties here for the case of multimedia data objects, but we put them aside for the moment, so as better to develop the concept.)

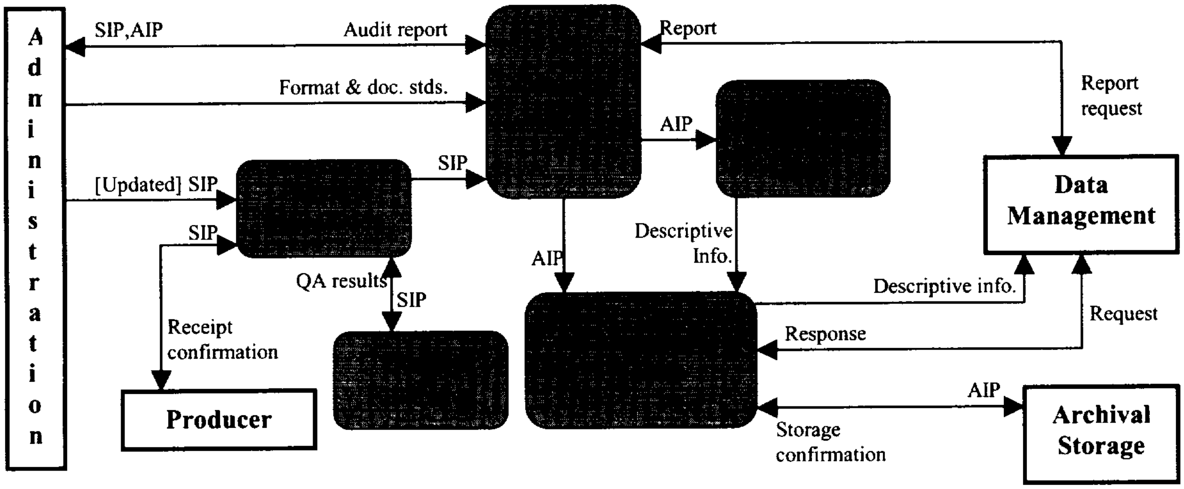


Figure 4. OAIS fig4-2: Functions of Ingest

Some examples:

- Many CD's actually contain a file system, and successful operation only relies on that file system. Copying such a file system onto a partition on a hard disk delivers an equivalent working representation. File placement is unimportant. Thus the file system is a viable underlying abstract form.
- In some cases it is only important to have a file tree, and the CD contents can be copied into a directory within an existing file system.
- Data held in a relational data-base can equally well reside in a variety of data-base engines, and still deliver its original content. The comma-separated files holding the content can be used as a system-independent representation of that content.
- A plain text document consisting of lines of characters drawn from the ASCII character set is meaningful in a variety of environments. Internet RFC's (Request For Comments) are typical of such documents.

Access involves realising the UAF on the technology appropriate to the time of access in such a way that the desired form of access (which may not necessarily be viewing) can be achieved. If we take the simple example of the Internet RFC, the same UAF is stored slightly differently on a UNIX system from a PC file system, because of the different conventions for line termination. However, the UAF of lines of text is the same in each case, and the UNIX `cat` command clearly displays the same information as the PC's NOTEPAD. The same lines of text represented in an EBCDIC (Extended Binary Coded Decimal Interchange Code) system would be represented very differently in terms of binary digits. The same data would be displayed in the same form. The underlying abstraction of lines of text is the same, but the different platforms represent the information differently internally.

Fig 5 illustrates the way in which we perceive data being preserved and accessed. The original object is identified as having a particular underlying abstract form, and this is used in the creation of a bit-stream for indefinite preservation. Software access to the information will be via some API (Application Programming Interface), which may go through several layers of abstraction before mapping onto the original data via its underlying abstract form. Two such layers of intermediate abstraction are illustrated by the empty ovals in fig 5.

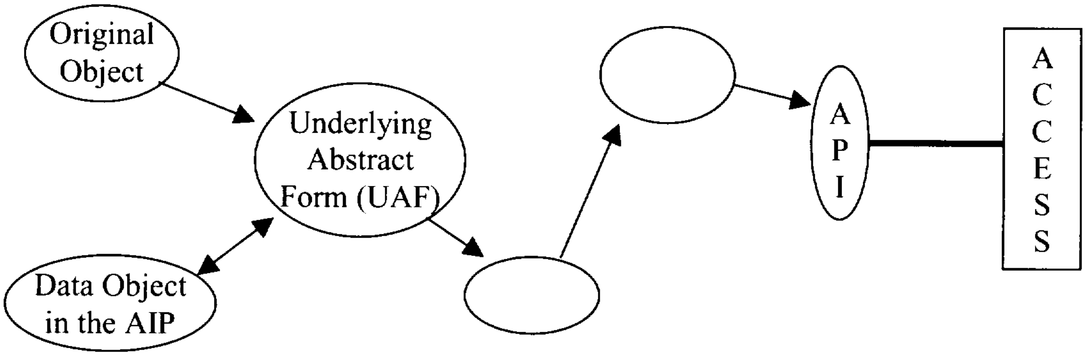


Figure 5. An Access path to the Preserved Data Object

6 Role of Representation Information

Clearly, the primary (sole?) purpose of RI is to enable access, i.e. to provide the "road map" for selection of a route such as depicted in fig 5. With this model of access via the UAF, the RI has two distinct roles.

1. enable generation of the UAF from the dissemination information package (DIP) (roughly equivalent to the AIP for those not familiar with the OAIS model), and

2. enable exploitation of the UAF on an available (and suitable) platform.

By *platform* we mean some computational facility which is capable of storing a representation of the UAF, and offers capabilities for running necessary software for accessing the intellectual content of the data in the desired manner.

A digital object may be configured for a variety of platforms (e.g. Many CD's will work with both MAC and PC), and the chosen UAF may well encapsulate this. It is up to collection managers to decide whether it is a significant property of the original digital object. The technology should give them that option.

7 Specifics concerning RI and UAFs

Fig 6 (taken from the OAIS model) splits the RI into 2 components corresponding with the division given above:

- **Structure Information** (StI) provides the information necessary for generating the UAF from the bit-stream that is the preserved data object.

- **Semantic Information** (SeI) which is usage information from the original release, modified to remove media dependency. There may be multiple entries in this, corresponding to multiple platforms.

- The UAF provides the formalism with which the horizontal line in fig 6 *adds meaning.*

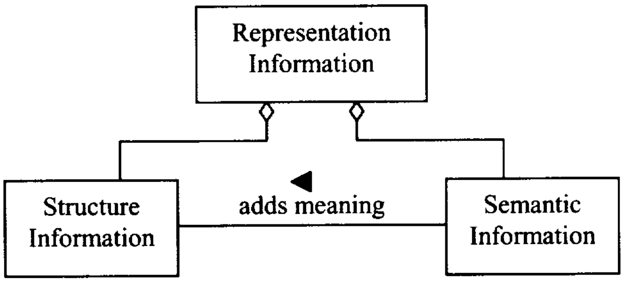


Figure 6. OAIS fig4-10: Representation Information Object

We gave examples of possible UAFs above. For many digital objects there is a set of possible abstract forms, and the choice of one that might be said to be *underlying* is not always trivial.

We propose a policy of choosing the highest level abstraction that discards no significant information. In a library context, the issue of what constitutes *significant information* is a collection management issue. The following examples illustrate the issue:

- A single PDF file held on a diskette can be considered as a file system, a file tree, a byte-stream, or a PDF file. It is our contention that the PDF file is the most useful UAF, as there is real prospect of finding several platforms that can make sense of it for decades to come.
- A set of PDF files might be treated as a set of objects, each of which has the above UAF, or we can decide that a set of PDF files is a *valid UAF.*
- A set of PDF files with a plain text READ.ME file, and perhaps a copy of the Acrobat reader on a CD raises more questions. The UAF is a file tree, but it is useful to include in the RI the information that a set of PDF files is incorporated within that file tree.

In introducing the notion of a *valid UAF* we imply that the management of the archive keeps track of all UAFs -- see below on *Gödel Ends.*

8 Platforms

A platform is thought of as a computational facility, though in extreme cases, it may be a human being reading plain text.

Dissemination involves delivering the UAF (or possibly something derived from it) in a form suitable for storage on the chosen platform. Better still, is the ability not only to store the UAF, but actually to render it (audio-)visually or to analyse it. The representation information in the dissemination package (RI in the DIP) provides access software, or information on which access software to use. Our proposals for management of Gödel ends (see below) address the issues of hardware and software obsolescence (referred to in OAIS section 4.2.1.3.2).

A platform might be:

- Browser
- SQL database
- PC file system
- application software (e.g. WordView)
- Khoros
- HDF software
- a very simple text viewer such as NOTEPAD
- or one of an ever growing number of options.

The archive information package (AIP) must hold in its RI all the information necessary to produce platform specific RI for inclusion in DIPs.

9 Names for Archived Objects

For the purposes of the CEDARS demonstrator we are allocating each object a Cedars Reference ID (or CRID). The CRIDs are used to form the links in the representation nets. We expect that a CRID will eventually be a URN (Unique Resource Name) [7,8], but at the present we have a simple name resolving technology using a simple server bound onto the socket port "crid:6386". The machine name "crid" can be translated locally by the hosts file, or by the local DNS (Domain Name Server), so that the locations of objects need not be fixed, and local copies can be accessed preferentially.

The OAIS AIP-id is probably the tail end of our expected URN. Our CRID has a component indicating the participating member of the archive federation, followed by an identifier allocated by the archive that creates the AIP.

More detail of the architecture for the CEDARS archive is given elsewhere [9].

In summary, the architecture has a number of archive stores, each of which is fronted by a gateway with a Web interface. The gateway organises access to meta-data, and organises delivery of preserved objects, subject to the satisfaction of access rights. All reference to a preserved object goes via a CRID name-server which then redirects the request to the appropriate gateway. This structure allows the evolution over time, as preserved objects may well be transfered to new systems, getting a new URL while retaining their original CRID.

In Fig 7 the numbered lines show the stages involved in the retrieval of an object, from resource discovery to delivery. The first stage shows the Web interaction between the search engine and the end-user's Web browser. Then an HTTP call is sent to the name-server (stage 2) where the object's CRID is translated into the URL for the gateway of the desired object (this redirection occurs very quickly). The end-user then interacts with the gateway using their Web browser (stage 3). Once the manifestation and platform for delivery are selected the gateway responds to the end-user (stage 4), these could be instruction to use FTP, or await a package delivered by ordinary mail, it could also be a continued Web interaction with the browser providing the object-specific access facility.

The final stage is the delivery of the digital object (stage 5 shows the process by which the object, in the form of a DIP, is delivered and interpreted by the end-user). The remaining (un-numbered) lines show management data paths, especially with regard to meta-data.

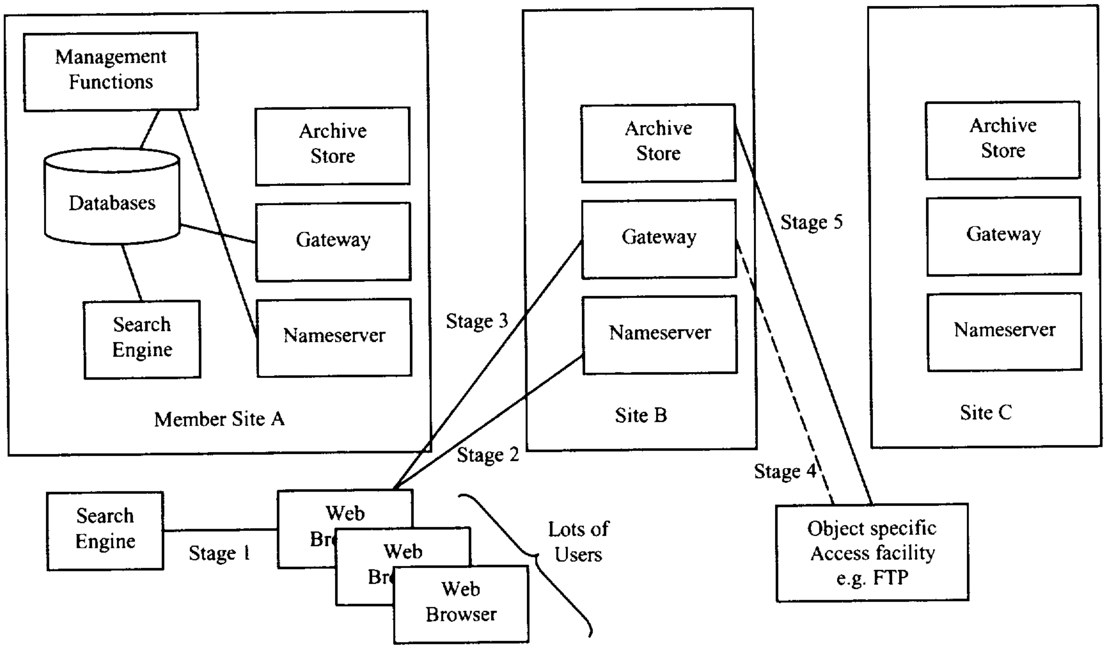


Figure 7. The Cedars Archive Architecture

Each member of the archive federation is seen as running a name-server, gateway and archive store, each with its management functions and databases (only shown for Site A for brevity). In fig 7 a three member federation is shown. Search engines may be independent or linked with the archive management.

10 Representation Nets

We are building a system of representation nets, and have implemented a tool that can be used for browsing such a net. In this section we use our own acronyms so that anyone who wishes to challenge our assertions of OAIS compliance has a language in which to do so.

Our representation nets involve 3 main types of node:

- **Data Format Definitions DFD** which define a data format, which is sometimes actual bytes and sometimes a more abstract entity such as an API. There is also a DFD for describing a real magnetic tape, or a human being interacting with a desktop WIMP interface.
- **Render/Analyse Engines RAE** take in data in one format, and deliver it in another. Deliver is interpreted very loosely, in that the output may take any of the forms described in a DFD.

- **Platforms** are typically computer systems, usually with storage, and are necessary for the execution of RAEs. They must contain or have access to storage suitable for storing the data that they are processing.

A *Data Format Definition* includes 2 lists of *Render/Analyse Engines*. One list enumerates those RAEs capable of accepting the defined format as input, and of delivering another format as output. The other list enumerates those RAEs capable of delivering the defined format as output.

An Underlying Abstract Form is a specialisation of DFD that includes RAEs that can accept the raw byte-stream of the *primary data object* as input and deliver the defined format as output.

Example:

> If a filesystem has been preserved as a tar file, it could be described as having a UAF of a filetree. Its UAF object could include a link to an RAE describing UNIX tar for generation of the UAF on a UNIX filesystem, and also an RAE describing WinZIP for generation of the UAF on a PC filesystem.

For any AIP the RI has two components at the top level, StI and SeI.

StI, the structure information, is concerned with describing (and especially regenerating) the UAF and consists of:

- CRID of the *UAF object* (i.e. a DFD including RAEs for building the UAF from the preserved data object).

- any parameters needed by the UAF object

SeI, the semantic information, is concerned with interpreting the UAF, and consists of a list of CRIDs of *render/analyse engines* (RAEs). With each CRID is held the parameter values needed by the particular RAE. Each RAE contains information (often in the form of software) for some particular processing of this AIP. In particular, the RAE includes a reference to the platform upon which it is to operate.

In some (many?) cases SeI may be null, and rely entirely on rendering facilities accessed via the UAF description. This depends on the extent to which the UAF is defined as a high level abstraction.

If we have a multimedia object, such as postulated in fig 8 taken from section 4.2.1.3.2 the OAIS model, we see the *multimedia mapping rules* as combination of the unpacking of the file-tree, and the association of file extensions with different data formats, which are themselves the references to representation information. The *multimedia operations and relationships* describe how to combine the different elements of the multimedia object to produce the intended rendition. A more formal example is given in the appendix.

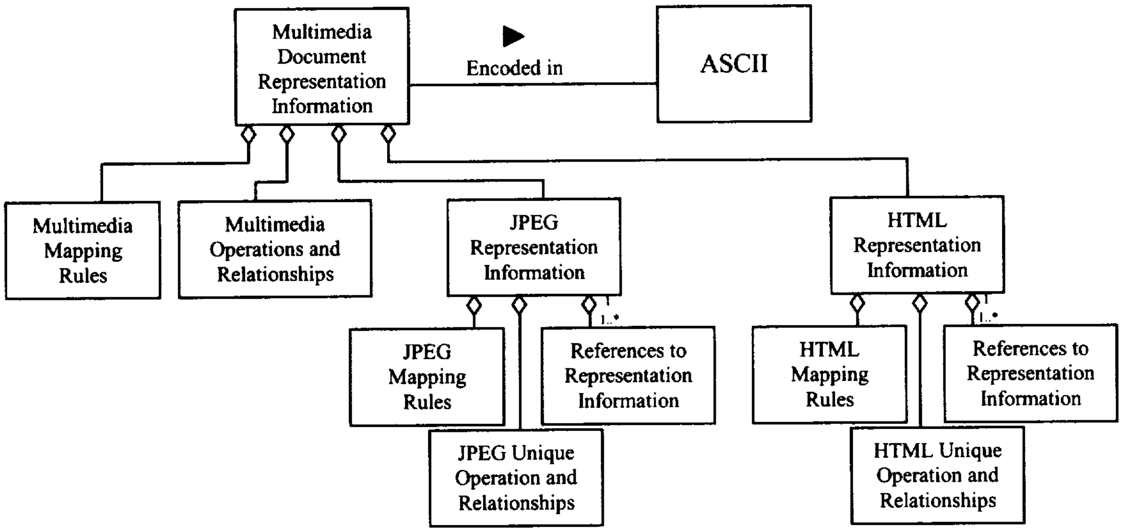

Multimedia Document Representation Information
Encoded in
ASCII
Multimedia Mapping Rules
Multimedia Operations and Relationships
JPEG Representation Information
HTML Representation Information
1..*
JPEG Mapping Rules
References to Representation Information
HTML Mapping Rules
1..*
References to Representation Information
JPEG Unique Operation and Relationships
HTML Unique Operation and Relationships

Figure 8. OAIS fig4-12: Example Representation Network

We see a value in attaching rendering capability at the data format level, because new facilities can be recorded at the data format level, and immediately become relevant to many preserved objects. This is particularly important in regard to following technological evolution. Our approach, means that the *references to representation information* of figure 9 (duplicated from the OAIS model), may be achieved via an indirection through an RAE with the capability to process the particular format.

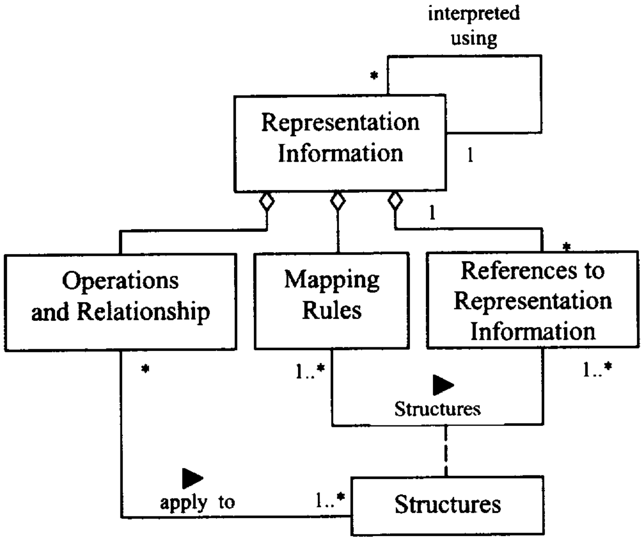

interpreted using
*
Representation Information
1
1
Operations and Relationship
Mapping Rules
*
References to Representation Information
*
1..*
Structures
1..*
apply to
1..*
Structures

Figure 9. OAIS fig4-11: Representation Network Object Model

11 True MultiMedia Objects

A CD with video and audio tracks, using the specific properties of a CD drive poses significant problems. The UAF would appear to be the actual spiral stream of bits on the CD, and the long term prospects for rendition seem to involve driver software that emulates this stream of bits on some other medium. It may be technically laborious, but the issue does not invalidate our approach.

12 Gödel Ends

As Gödel's theorem tells us, any logical system has to be incomplete. There must be truths which the system cannot itself deduce.

The representation nets must have ends corresponding to formats that are understood without recourse to information in the archive, e.g. plain text using the ASCII character set, the Posix API. All references to such a format must be via the same CRID, and the management of the archive must have an inventory of such objects. As a format becomes obsolete, the object referenced by the CRID can then be updated to permit understanding of the obsolete format in terms of current practice.

In our scheme, the *platforms* upon which the *render/analyse engines* (RAEs) run are the things that become obsolete as a result of the march of technology. A data format becomes less and less accessible as the platforms of the relevant RAEs become obsolete. As a long stop there is the documentation of the format, which we might consider as a special RAE whose platform is human (e.g. a programmer). In the case of proprietary formats we may not have this information.

Thus the platforms (e.g. Win32) are the things that are outside the archive, and as such are the true Gödel ends of the system. As Jeff Rothenberg [10] has pointed out, emulation of the original computational environment, gives the very best hope for recreation of the experience of a preserved digital object. This can be a laborious process (see Emulation below), and technology shifts may render a true recreation impossible. It is quite possible that the technology of the time actually limited the access to the intellectual content, and a far better access to archived material can be achieved by implementing viewers that operate on the obsolete format directly.

We therefore conclude that the archive administration keeps an inventory of the platforms which occur in the archive's representation nets, and that the Gödel end platforms each contain the IDs of the data formats that rely on that platform. The administration needs to keep this inventory under review, and to be open to the possibility of adding extra RAEs to data format objects in order to maintain accessibility.

Some platforms may not be true ends, as they may have been realised by emulations. An emulation is a type of RAE, and as such depends on a platform. This gives us a facility for representing a chain of emulations on the Rothenberg model.

13 Proprietary Formats

Any archive has an understandable reticence about keeping data which is held in undocumented formats. However, *reticence* must not be seen as a synonym for *rejection*. A current proprietary format may not have any publicly available documentation, but may have readily available render facilities (e.g. WordView) on current platforms. As obsolescence threatens, the commercial value of the documentation is minimal, and there is real prospect of adding it in later.

For instance, anti-virus companies have already reverse engineered the specifications of Microsoft formats.

14 Emulation v Format Conversion

We are led to the view that emulation does not always give the most useful rendering of preserved information.

We have looked in particular at two early systems:

- GEORGE3, a mainframe system common in the UK in the 1970s, running on ICL 1900 series machines, and
- The BBC-Micro computer, produced by Acorn and prevalent in UK schools and colleges in the 1980s

Several emulators for the BBC-micro are available over the Web, and provide a good platform for running much of the educational software that exploited the machine's facilities. The machine also had an early word processor (called VIEW) in which some quite large documents were produced (included the system's manual). Although one can run VIEW in emulation, a better access to the information would be achieved if it were to be converted to a more modern word-processor format.

Emulation of the GEORGE3 system is of a different character. The 1900 machine was character oriented with a 24-bit word. The system had a hierarchical filestore with integral management of a tape library. Our emulation of the system creates quite a lot of the feel of operating the real machine (although the modern PC screen fails to generate the ambience of the teletype operator's console, or the buzz of the 2000 cards-per-minute reader). Our CEDARS preservation of this system is a composite object, and holds its tape library. It can also function as a platform for other tapes from that system.

The source text of GEORGE3 was held on 2 magnetic tapes. In the real system this was a cumbersome object, and searching it was rarely a quick process. However, the tape format is not very complex, and a program to render these tapes as an ASCII file was easily written, and forms a much more convenient RAE for these tapes than does the GEORGE3 emulation platform.

On the other hand, the ICL 1900 was the platform for the world's first Algol68 compiler, and access to the intellectual content of this is at its best on the emulated 1900 platform.

We therefore argue that access to the intellectual content should take place at an appropriately chosen level of abstraction. Where emulation is appropriate, it is a powerful technique, with great potential for historical reconstruction and nostalgia. Elvis Presley on CD sounds better than on a 1956 78rpm wind-up gramophone.

We are trying to gauge the effectiveness of our preservation techniques, by pretending that they were available 15 years ago, and then using them on the material of that time. Providing access to the intellectual content of this material on today's hardware is enabling us to assess the merits of different emulation approaches.

15 Emulation Practicalities

The GEORGE3 operating system functions at different levels.

- The underlying hardware, where different members of the range have different interfaces to peripherals
- upon which is run the EXECUTIVE program, which implements
- the GEORGE3 executive interface with a uniform set of system calls for driving peripheral devices, upon which is run
- GEORGE3 itself which controls all the resources and provides various facilities for end users and operators, and implements
- the applications program interface (API), in which there are file access facilities and other features used by
- applications programs -- the ultimate *raison d'être* of the computer system.

Our emulation has taken place at the interface between GEORGE3 and EXECUTIVE, although some limited emulation has also taken place at the API level.

In terms of today's PC material, it may well be that the best future access to intellectual content is achieved by emulation at the BIOS level, whereas the Windows API may be a better level of emulation in other cases. We doubt that emulation of the raw hardware under the BIOS will ever be valuable, beyond study of the BIOSes themselves.

16 Preservation Format

The above discussion of emulation as opposed to format conversion raises the question of what format to use for long-term preservation. Our approach is to preserve something close to the original digital object (which we have called the *underlying abstract form*), and to have representation information that gives access to the ability to convert yesterday's obsolete format into whatever form is appropriate for the required form of access.

We attempt not to pre-judge the nature of future accesses, and believe that retention of original data maximises the options open to future researchers. We believe that our benchmark of *recreation in principle of the significant properties of the original* addresses this need (see section 3 above).

We readily accept that this criterion is not the same as recreation of the original, but only its significant properties. The discipline of deciding at ingest time what these properties are provides a focus for selecting the preservation format, and aids in the generation of really meaningful representation information.

This accords well with our decision in 1991 that in preserving the data from VM/CMS on our Amdahl system we should retain the data in EBCDIC and provide a converter utility, rather than convert to ASCII as we moved onto UNIX and NetWare platforms. We can now render the full EBCDIC character set using UNICODE, an option that would have been lost by any conversion to ASCII.

17 Conclusion

We believe that we are well on the way to a functioning demonstrator, adhering to the overall architecture of the OAIS model, and with useful example material showing possible directions for Representation Information within such an archive.

Key to our approach is the identification of the significant properties and the correct underlying abstract form. We contend that representation information should contain reference to appropriate rendering software. On the other hand we suspect that data whose format is only described textually may well be ignored by future generations because of the labour involved in achieving meaningful access to its intellectual content.

18 Acknowledgements

This work was conducted as part of the Cedars project which is funded by JISC/e-Lib programme [11,12] and has three lead sites, the universities of Cambridge, Leeds, and Oxford.

Colleagues within the CEDARS project are addressing issues of Preservation Description Information.

References

[1] Consultative Committee for Space Data Systems. Reference Model for an Open Archival Information System (OAIS). (CCSDS 650.0-R-1, Red Book, 1999) (http://ssdoo.gsfc.nasa.gov/nost/isoas/ref_model.html),

[2] D Holdsworth. The Medium is NOT the message OR Indefinitely long-term file storage at Leeds University. (Published in: proceedings of Fifth NASA Goddard Conference on Mass Storage Systems and Technologies, NASA publication 3340, 1998)

[3] J Lindner. Toward a MediaLESS Archive. (proceedings of Paris 2000, 5th Joint Technical Symposium, Image and Sound Archiving and Access : the challenge of the 3rd Millennium).

[4] C Jenkins (editor). Collection Management in Academic Libraries. (Particularly chapter 9 by N Elkington) (1999)

[5] S Lee. Oxford, UK. (http://info.ox.ac.uk/localnet/dwp/priority.html) Answering the question of collection management decisions.

[6] N Beagrie and D Greenstein. A Strategic Policy Framework for Creating and Preserving Digital Collections. (1998) (http://ahds.ac.uk/manage/framework.htm)

[7] R Moats. URN Syntax. (RFC 2141 – 5/1997) (http://sunsite.icm.edu.pl/pub/doc/rfc/rfc2141.txt)

[8] A E Walsh. Draft : URN Namespace Definition Mechanisms. (10/1998) (http://www.web3d.org/WorkingGroups/media/hypermail/1998/0175.html)

[9] D Holdsworth. Draft: Proposed Architecture for CEDARS demonstrator (1998). (http://gps0.leeds.ac.uk/~ecldh/cedars/architecture.html)

[10] J Rothenberg. Avoiding Technological Quicksand: Finding a Viable Technical Foundation for Digital Preservation. (01/1999). (http://www.clir.org/pubs/reports/rothenberg/contents.html)

[11] Two web sites for the Cedars Project. (http://www.curl.ac.uk/projects.shtml) (http://www.leeds.ac.uk/cedars)

[12] K L Russell and D M Sergeant. The Cedars Project: Implementing a Model for Distributed Digital Archives. (RLG – Diginews 3:3, June 1999) (http://www.rlg.org/preserv/diginews/diginews3-3.html#feature)

[13] D Holdsworth and D M Sergeant. Representation Network Example. (http://www.leeds.ac.uk/cedars/blueprintAppendix.html)

Appendix

Here is an example of one of our AIPs [13], with the data object holding a PDF book and a colour GIF image of the original front cover. This AIP exists in our archive store and also has an access page on the gateway. The PDI component of the AIP (see fig 3 in the main text) is only a title and the object's CRID, which is the minimum necessary for our exploration of representation nets. The CRIDs are the arcs of the representation net.

The contents of Packaging Information are not shown.

PDI	RI	PDO

CRID=01trav
Title=Tigers
	StI=↑31fset
	SeI={ ↑51pdfr(tiger.pdf),↑52gifr(cover.gif),↑53webb }
		BitFile001=tiger.zip

The structure information (StI) in the RI is referenced indirectly (i.e. via a CRID for) and is an AIP which describes the data format (in this case a set of files) and facilities for interpreting it. The semantic information (SeI) in the RI is a list of CRIDs for RAEs that can give access to the intellectual content.

The StI has the following specialised fields in its data object: UAF, a description of the underlying abstract form; TOI, a *transformer object instance* (a special RAE which can generate the UAF from the preserved byte-stream); RAE, a list of render analyse engines that can generate other representations (or provide APIs).

PDI	RI	PDO

CRID=31fset
Title=FileSetUAF
	StI=↑32uafp
	SeI={ ↑50gate,↑54asciir }
		UAF=↑71fset RAE=↑80flist
		TOI= {↑81pczip,↑82maczip,↑83unixzip }

The representation net node (↑31fset) conveying information on how to deal with the UAF of the "Tigers" AIP also carries RI to interpret its own digital object component. The object referenced by ↑32uafp conveys this information, and only deals with the UAF specialisation of a DFD (see section 10). Although two choices are provided to render our fileset AIP in the SeI, while the Cedars gateway (↑50gate) can trace through the internal indirections of the representation net and present an overall decision framework, the ASCII renderer (↑54asciir) can only display the Java property file in the PDO verbatim.

PDI CRID=71fset Title=filesetDesc	RI StI=↑33asti Sel={ ↑54asciir }	PDO Fileset_description.txt

PDI CRID=81pczip Title=PCunzip	RI StI=↑34raep Sel={ ↑50gate,↑54asciir }	PDO Platform=↑91pc engine=winzip.exe Params=extract OutForm=↑72ftree

Examining the contents of the fileset AIP (↑31fset) the UAF field references an AIP containing an ASCII description of a fileset (↑71fset), whose RI shows how to render the ASCII file. The RAE field provides access to a tool (↑80flist) that displays the structure of the UAF, in this case by listing the filenames (tiger.pdf and cover.gif) which make up the set. The TOI list field provides methods for transforming the content file of the "Tigers" AIP into these two files on different computational platforms. Only the RAE for the PC platform has been shown (the others are similar). This RAE (↑81pczip) has a dedicated StI (similar to ↑32uafp). ↑91pc references an AIP containing an ASCII description of the PC platform, and ↑72ftree is the DFD for a filetree.

PDI CRID=32uafp Title=uafUAF	RI StI=↑32uafp Sel={ ↑50gate,↑54asciir }	PDO UAF=↑73uafpf RAE=↑80flist TOI= {↑84asciicp }

PDI CRID=51pdfr Title=PDFreader	RI StI=↑35rael Sel={ ↑50gate,↑54asciir }	PDO InForm=↑74pdf RAE={ ↑87pcpdfV,↑88macpdfV }

PDI CRID=87pcpdfV Title=pcPDFviewer	RI StI=↑34raep Sel={ ↑50gate,↑54asciir }	PDO Platform=↑91pc engine=acroread.exe Params=none OutForm=↑75gui

These last three nodes illustrate some important features of our representation net. ↑32uafp is a Gödel end for UAF nodes, ↑87pcpdfV is a Gödel end for the rendering platform. We show PDF as a platform (↑51pdfr), which is one of a number of choices offered in section 8. As the Gödel end platforms become obsolete this formalism offers the choice of adding PDF rendering capability on new platforms or emulation of the obsolete platform upon which the existing rendering software will run. The parameter, shown in brackets in ↑01trav, is passed through to the software engine of ↑87pcpdfV via the ↑51pdfr. If desired, (tiger.pdf) could be rendered via ↑88macpdfV instead. All new rendering software for pdf is added to the list of RAEs in ↑51pdfr.

Project 1244: IEEE Storage System Standards

John L. Cole
Computational and Information Sciences Directorate
United States Army Research Laboratory
Aberdeen Proving Ground, MD 21005-5067
jack.cole@ieee.org
tel +1-401-278-9276
fax +1-401-278-2694

Abstract

Approaching its tenth anniversary, the IEEE Storage System Standards effort is in the process of balloting Media Management System (MMS) standards. These represent the first standards for the IEEE Storage System Standards Working Group (SSSWG), and the first storage system standards for the world. In the early years, SSSWG produced the Mass Storage System Reference Model (MSSRM), directly influencing the design of many successful commercial products and the MMS standards themselves.

The IEEE Storage System Standards Committee (SSSC), sponsor of SSSWG, will work in the coming year to complete work on the suite of MMS standards, and begin work on new projects. New projects for SSSC in 2000 include tape standards, tape recommended practice, and a project to develop a Guide for Storage System Design. Existing and new collaborations with other groups developing storage-related standards will be fostered in 2000.

The SSSC is driven by the urgent need for interoperable storage system software, and storage systems that are highly scalable and functional in distributed, heterogeneous environments.

1 Background

1.1 In the Beginning

The IEEE Storage System Standards effort began unofficially with individual discussions in the 1980s to standardize and guide development of hierarchical storage management systems. In the summer of 1990, the IEEE approved the SSSWG charter and the first (and for a long time the only) project that resulted in the un-balloted MSSRM. Development of the MSSRM progressed through several versions until the last revision, version 5, was approved by an internal SSSWG vote in September 1994.

1.2 SSSWG Charter

The IEEE Storage System Standards Committee is chartered to model generic mass storage systems, and based on such modeling, to develop widely accepted, readily implemented standards with minimal licensing requirements.

In addition to working on standards, the SSSWG may develop recommended practices and guides. The SSSWG is primarily concerned with Distributed Storage System Design, and the SSSWG, without favor, includes Storage Systems of every scale in its studies. An object-oriented approach is desired in all SSSWG efforts, and net-attached storage is intrinsic to its model.

SSSWG must also consider promising emerging technologies in its modeling, even though standards for parts of the model may not be immediately practical as a result. The model may remain partly an abstraction expressing desirable features, and the associated standards expressing practical requirements relating to current technologies.

The purpose of these standards is to promote use of best technologies resulting in interoperable, fully-scalable systems permitting ready access of information throughout distributed, secure, heterogeneous net-attached storage systems.

1.3 MSSRM

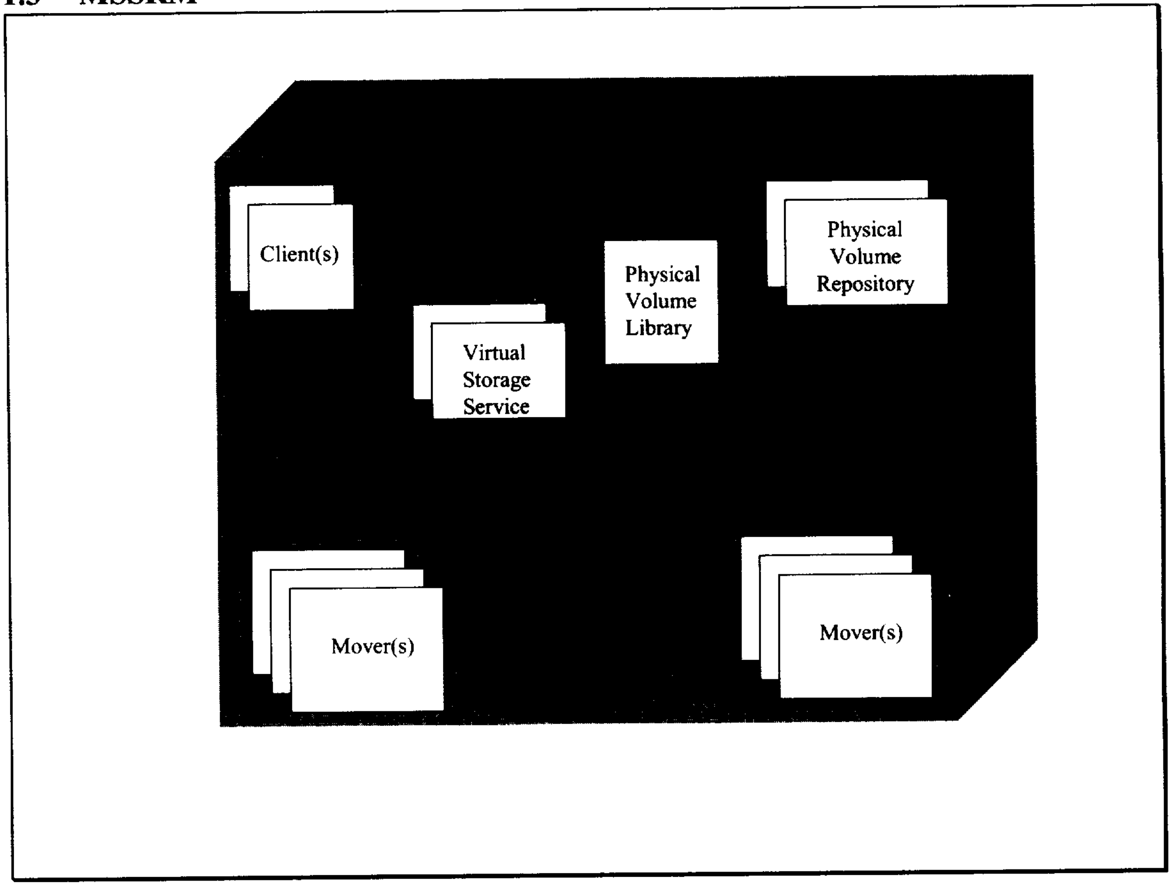


Figure 1. Components of the MSSRM

The IEEE Mass Storage System Reference Model (MSSRM), although not balloted as a standard, has been highly successful in service as a guide for development of many well-known storage systems and components of systems, commercially and otherwise, in use today.

Along the way, the MSSRM has been revised a number of times, and renamed the IEEE Reference Model for Open Storage Systems Interconnection (OSSI). Version 5 of the OSSI Reference Model was approved for public release in September 1994.

There were seven IEEE-approved Project Authorization Requests (PARs) relating to the MSSRM — one PAR for the model itself and six for modules of the MSSRM representing sets of services identified by the SSSWG as those essential to the composition of viable storage systems. The PAR for the MSSRM has been revised and renewed as a different project, and these six PARs, which represent the modules into which the MSSRM was partitioned, are withdrawn:

SOID (1244.1) Storage Object Identifier
PVL (1244.2) Physical Volume Library
PVR (1244.3) Physical Volume Repository
MVR (1244.4) Data Mover
MGT (1244.5) Storage System Management
VSS (1244.6) Virtual Storage Service

In addition, other associated documents were created, such as the "Virtual Storage Architecture Guide"[1].

The Reference Model was intended to provide a framework for the coordination of standards development for storage systems interconnection and a common perspective for existing standards. Through development of this structured framework, the Model would expose areas where standards were necessary or in need of improvement.

The technology and application independence of the Model would accommodate descriptions of advanced technologies and expansion in user demands. This flexibility would also support the phased transition from existing implementations to storage system standards.

It was not the intent of the Model to serve as an implementation specification, to be the basis for appraising the conformance of actual implementations, or to define precisely the standards for services and protocols of the interconnection architecture. Rather, the Model was intended to provide a conceptual and functional framework allowing teams of experts to work productively and independently on the development of standards for storage systems. These remain, roughly, the intentions of the IEEE Model.

All who participated in drafting the MSSRM must be proud that this model was one of the earliest efforts to presage present concepts of storage objects and of net-attached storage.

1.4 Change of Direction

The present direction of the SSSWG is quite different.

Although work continued after 1994 to develop the PVR and PVL modules, it became apparent to the SSSWG that the approach taken to encompass all aspects of storage systems made it impossible to compose practical standards. To that end, the SSSWG began a device-driver level standard called "The Media Changer Service Standard", or "MCS".

Work on MMS began shortly after SSSWG came to the realization that writing standards for PVL and PVR was an intractable problem. These and other components of the MSSRM were and are very robust, very all encompassing, and very difficult to reduce to standards. In order to make progress and actually publish standards in the lifetimes of the SSSWG members, a different direction was needed.

As the SSSWG met with individuals interested in MCS, it became aware of an effort by some of the MCS participants to develop a minimalist media management system known as "OpenVault" (www.openvault.org).

After hearing presentations on the OpenVault effort, it seemed to the members of SSSWG that OpenVault embraced the same one, true reality of storage system needs which SSSWG saw, and that standards based on a more minimalist approach would be possible in a reasonable time. OpenVault was still developing; some of the people developing OpenVault were early members of SSSWG; OpenVault capitalized on ideas from the MSSRM; and OpenVault, it appeared, could co-evolve with an IEEE set of standards. Today, the IEEE MMS and OpenVault are close, although not equivalent, as a seesaw development of the two has progressed over more than two years.

1.5 MMS Architecture [reference 2]

This describes the motivations for an overall architecture of the IEEE Media Management System. Although the architecture may suggest a particular design or implementation, it is not the IEEE's intent to favor a specific implementation of the MMS. Indeed, it should be possible to implement the MMS in a number of ways, ranging from a "lightweight" implementation in a scripting language such as *perl*, or a full implementation written in a traditional programming language such as C, C++, or Java.
The MMS is a software system for managing physical media. The system has the following properties:

- It is ***media-neutral***, allowing the management of computer tapes, disk media, disks, optical disks, CD-ROMs, as well as non-computer media such as videotapes or reels of film.
- It is ***scalable***, being comfortable in environments as small as a single individual's office or home and as large as a multinational corporation, educational or scientific institution, or government archive.
- It is ***platform neutral and operating system independent***, working with existing computer systems from multiple vendors with varying degrees of media-handling sophistication.

- It is ***distributed***, allowing access to media and the devices that store and perform data transfer operations on the media by more than one system. A single MMS may manage devices that are connected to many host computer systems, including devices that are physically connected to multiple hosts. Connectivity between elements of the MMS requires the availability of standard TCP/IP.
- It provides a reasonable degree of ***security and protection*** for access to the media by ensuring that specific media may be mounted only by those applications which have authority to access that media. All parties are authenticated, and network communication is digitally signed so that it is extremely difficult to forge.
- It is ***content-neutral***, and does not have any inherent understanding of the content of the media; indeed, with some media, such as videotape or film, the MMS many not even have access to the content of the media.
- It is ***application independent***, providing appropriate media management functions for diverse applications ranging from backup and hierarchical storage management, to broadcast television automation. Media belonging to multiple applications may be managed by a single MMS; these applications may be multiple instances of the same program, or of different applications.
- It is designed to be ***modular*** to allow independent groups to work on components of the MMS independently; the modularity is provided by strong, flexible interfaces that can evolve over time.
- It is ***language-neutral***, permitting programmers to write applications that interact with the MMS in almost any programming language, and, indeed, to allow the MMS itself to be written in almost any programming language.
- It allows ***multiple implementations*** to interoperate seamlessly.

The key to the architecture of MMS is to clearly define the basic functionality that the MMS must provide, and to declare specific points in the functionality to provide defined interfaces that allow independent components to interoperate.

1.6 MMS Described

The IEEE Media Management System (MMS) suite of ten standards arguably could have been a single standard, although the first five total 304 pages in aggregate. The intent is that these five standards will evolve separately. Also, that builders of storage systems or components could comply with each standard individually.

A tutorial of MMS was given in March 1999, but to review the components briefly to see what has been balloted and what remains, some brief information about MMS is offered.

The following drawing depicts potential system boundaries (thick lines) between the core (or cores) of a media management system (or systems) and the protocols used to communicate among the components. And it suggests the distributability of a MMS. In this drawing "DM" is the drive manager; "dmp" is the drive management protocol; "LM" is the library manager; "lmp" is the library management protocol; and "mmp" is the media management protocol.

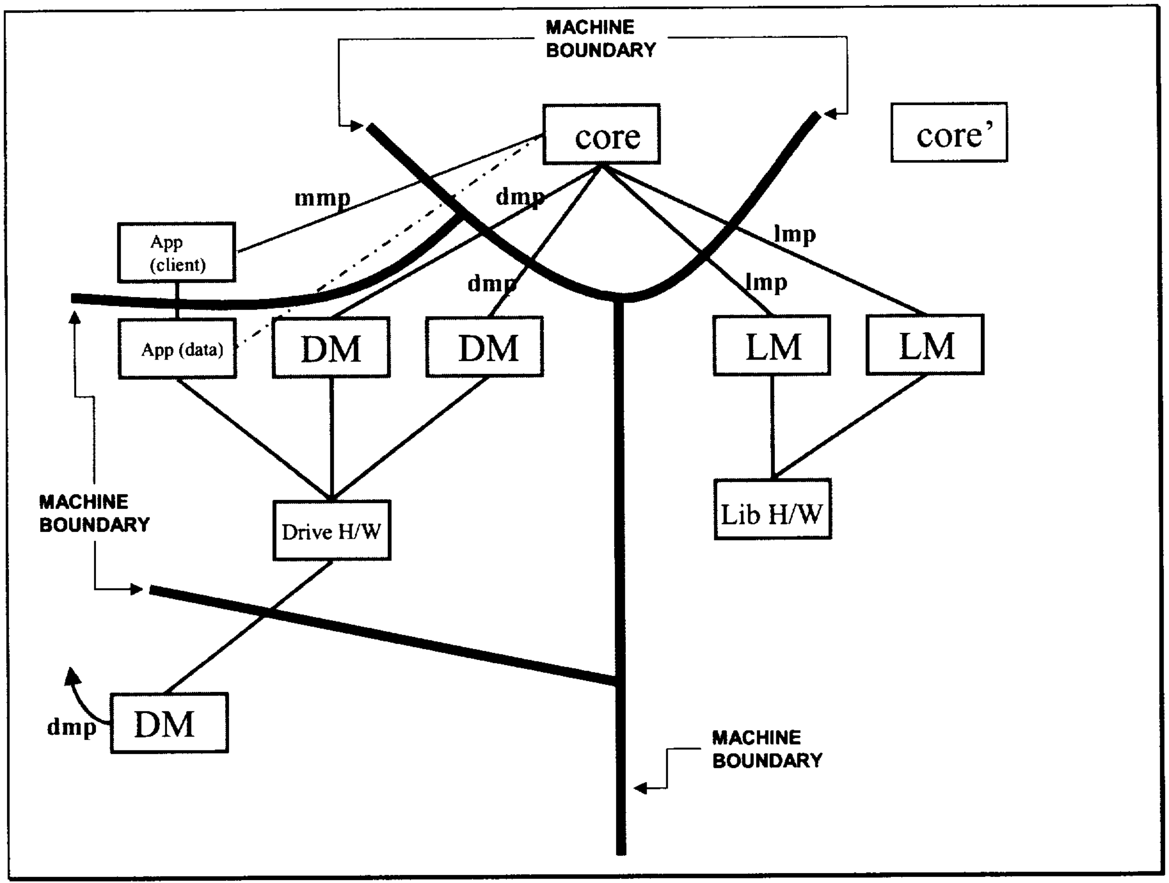


Figure 2. MMS Distribution of Components Across Machine Boundaries

There is a suite of ten MMS projects and a one project to develop an IEEE Data Mover under the 1244 series. The first five of the MMS projects are draft standards in balloting now. Work will proceed on the other five and MOVER, which is not part of the MMS, this year.

1244.1 - Media Management System (MMS) ***Architecture.***

Specifies the architecture of a distributed, platform-independent, system to manage removable media, including both disk and tape, using robotic and manual methods. The general schema for managing media, the expected components of the software system, and the data model to be supported by the software system for managing this media are described by this standard. Details of components of the Media Management System are specified by companion standards.

1244.2 - Session Security, Authentication, Initialization Protocol ***(SSAIP)***

is the initial "handshake" protocol used by components of the MMS to establish identity, authority, and initial communication.

1244.3 - Media Management Protocol ***(MMP)***

used by client and administrative applications to allocate, deallocate, mount, and dismount volumes, and to administer the system. The MMP includes levels of privilege so that, for example, a client application cannot perform administrative functions, or an operator console program cannot perform higher-level management functions.

1244.4 - Drive Management Protocol (***DMP***)
is used between two software components of the MMS: the central management core and a program that manages a drive which is used to access removable media.

1244.5 - Library Management Protocol (***LMP***)
is used between two software components of the MMS: the central management core and a program that manages an automated library or a vault. The minimum functionality required to implement an MMS is the SSAIP and MMP. Most practical implementations will include the DMP and LMP. Additional protocols are defined to extend the MMS to interoperate with other MMSes and with other media management systems:

1244.6 - The Media Manager Interchange Protocol (***MMIP***)
defines a protocol to allow interchange of information between autonomous Media Managers.

1244.7 - The Media Manager Control Interface Protocol (***MMCIP***)
defines a protocol which permits interfacing the data management component of the MMS with existing library management systems.

1244.8 - The C Language Procedural Interface
defines a set of standard programming interfaces which facilitate construction of components of the MMS, particularly client, administrative, and operational applications, library managers, and drive managers. The initial definition will be for the C programming language. The interface will be designed so that implementation in languages such as C++ or Java could be easily accomplished.

1244.9 - MMS User Mount Commands
defines a set of standard commands to allow a user to mount, unmount, acquire, and release media. These commands are specified as a part of a command line interface for systems that offer such interfaces, such as the UNIX shell or NT command line interface. Commands may be embedded in scripts to produce more complex or custom functions, or to allow an application program that is not written for MMS to be adapted for use with MMS.

1244.10 - MMS Standard Administrative and Operational Commands
defines a set of standard administration and operation commands of an MMS. The standard defines a command- line, minimally interactive interface for basic interaction with the MMS; these commands could be used to construct interactive interfaces using scripting-based systems such as web CGI scripting or tcl/tk.

1.7 MOVER

Very little thought or work has gone into developing a data mover standard since MSSRM version 5. If there are volunteers to develop this standard, and it is not overcome by trends in net-attached storage, the SSSC will ballot a draft standard before 2002. The present PAR for a data mover standard is:

1244.11 - MOVER
provides a standard storage system data mover architecture and interfaces for use by the IEEE Media Management System and other storage system software. MOVER transfers data between two endpoints in a distributed storage system." This deceptively simple statement belies the difficulty in describing MOVER as a standard.

The MSSRM concept of MOVER is synopsized in the Mover Architecture drawing and description that follows:

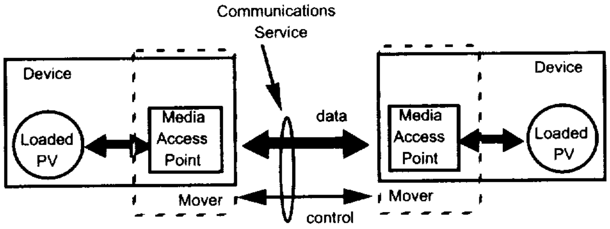


Figure 3. MSSRM Concept of Mover Architecture, "PV" is "Physical Volume"

"The Mover performs operations on media access points and affects data transfer. Media access points are the means of accessing physical volumes and sections of memory accessible to Mover clients."

"A Mover performs two distinct functions: 1) it changes or monitors the read/write state of a device (e.g., positioning within the physical volume, reporting status and errors, and loading and unloading physical volumes as necessary). 2) it transfers data and source/sink information (to effect the transfer of data) between devices, devices and memory, or from memory to memory."

2 Present and Future

2.1 Balloting Status, Note About Standards

The draft standards in ballot now are:

1244.1 - Media Management System (MMS) Architecture
1244.2 - Session Security, Authentication, Initialization Protocol (SSAIP)
1244.3 - Media Management Protocol (MMP)
1244.4 - Drive Management Protocol (DMP)
1244.5 - Library Management Protocol (LMP)

The SSSC could have held a ballot without involving the IEEE Standards Department Balloting Service, but chose to use that service. At this time balloting is carried out partly online and partly by old-fashioned methods. Registration of interested parties and balloting itself are carried out online, but the invitations are sent through the U.S. Postal system. Certain time periods are allowed for sending messages and receiving responses, so that balloting consumes months of time. In the case of the first five MMS draft standards, balloting is being accomplished online, and began December 15, 1999. This was an unfortunate timing with preparations for the holidays and concerns over Y2K effects. Roughly fifty individuals registered interest in balloting the MMS draft standards, and 31 responded to the official ballot invitations from IEEE. The first ballot of these standards ends on January 14, 2000. Depending on the response, revisions and re-balloting (re-circulation it is called) may be required. The intent is to complete all of the process in time for submission to the March 2000 IEEE meeting in which approval is sought. If this succeeds, these standards will be published by summer of 2000.

A note about the nature of standards: they are not fixed in stone. Standards are living documents, receiving modifications during their lives, being re-validated periodically, occasionally being withdrawn, and being re-balloted when the aggregate of modifications becomes too great. The work of groups like SSSWG to develop draft standards and initial balloting are just the beginning of the journey for standards. The anxiety by some that standards somehow bestow a lock-in on anything is founded on a misunderstanding of standards.

2.2 Remaining MMS and MOVER Standards

The remaining MMS standards to be developed are:

1244.6 - The Media Manager Interchange Protocol (MMIP)
1244.7 - The Media Manager Control Interface Protocol (MMCIP)
1244.8 - The C Language Procedural Interface
1244.9 - MMS User Mount Commands
1244.10 - MMS Standard Administrative and Operational Commands

The group believes that 1244.6 can be completed in 2-3 weeks, 30 pages of writing, and an additional 20 pages of XML DTD. One SSSWG member volunteered to serve as editor of 1244.8 starting with commercial work of another member in this area. The projects 1244.7 and 1244.11 were deemed too complex to grapple immediately. Project 1244.9 (user mount commands) will be accomplished for UNIX only, and will permit scripting. Project 1244.10 will be presented as a trial use standard only. In fact, discussion entertained the idea that all three of the Programming and Command Line Interfaces should be trial use standards (1244.8-1244.10).

Work will proceed on development of a 1244.11 MOVER draft standard subject to the normal limitations of interest, volunteerism, and time.

2.3 SSSC

Over the last several years the SSSC focus has been only on SSSWG and MMS standards. The new sponsor chairs for storage system standards will work to broaden the focus of SSSC to include other projects, working groups, study groups, and collaborations. The SSSC itself will become an actual committee of several people instead of just the sponsor chair.

2.4 Collaborations

During 1999, the chair and other members of SSSWG exchanged mail, held teleconferences and meetings with members of the Distributed Management Task Force (DMTF, www.dmtf.org) and members of the Storage Networking Industry Association (SNIA, www.snia.org) to provide the IEEE definitions of storage objects to the Common Information Model (CIM) being developed by DMTF. Major industrial entities are basing their Web-based Enterprise Management (WBEM, pronounced "web-um") products on CIM, and so this collaboration is very important. The SSSC will continue to pursue collaborations such as this in 2000 and beyond.

2.5 Tape Standards

Three tape standards were suggested by a member of SSSWG, and project authorizations requested. Approval for these projects is assumed, and SSSC intends to purse these in 2000:

Portable Tape Driver Architecture (1563.1, Recommended Practice) provides a reference model for tape driver architectures that is portable across multipleoperating system environments, fully featured, and high performance.

A fully realized architecture that industry can base their implementations on that will reduce the effort required to support a new tape device on a given platform and thereby increase the available choice of drives on any given platform. This will benefit the application vendor and the end customer.

Common Tape Driver Semantics (1563.2) defines a common set of operations and semantics for access to tape drives across multiple operating systems platforms.

Eases the task of porting and supporting applications that use tape storage across multiple operating system environments. This will enable application vendors to port to more platforms and thereby increase the end customer's available choices.

Common Format For Data On Tape (1563.3) defines a self-identifying format and record structure for the storage of data and meta-data on tapes, a structure that contains the key to understanding the format of the data stream as well the data itself. An analogue from the networking world would be the Document Type Definition (DTD) structure used to describe documents in XML (eXtended Markup Language).

Enables data written by one application to be accessible by other applications without those applications having to know how each other encodes data written to tape.

2.6 Guide for Storage System Design (P1600)

This project, a revision of the original P1244), will produce a clear, abstract, model exposing the design features required for storage systems to provide transparent, secure information access in highly distributed, heterogeneous computing environments.

The model produced will describe design alternatives and rationales applicable within the spectrum of valid storage system architectures.

Emphasis in the model will be placed on net-attached storage, object-oriented design, open source software, minimal licensing alternatives, and maximum scalability.

The work under this project will serve to revise the popular IEEE Mass Storage System Reference Model version 5 of 1994, and use it as a basis for related IEEE Recommended Practices and Standards.

Purpose. This Model will guide implementers toward interoperability in meeting such demands by suggesting best use of current and emerging technologies.

This Model will inspire commercial designs of storage systems and system components from a broad spectrum of implementers, resulting in a high level of interoperability throughout the world.

3 Observations, Motivations, Problems

The entire area dubbed "storage" suffers from a set of common problems.

First, the focus on storage is grossly misleading. Storing things, including data, can be very simple. It is the act of accessing data that makes it information. So the thrust of all efforts in "mass storage" is only for the sake of "information access". And, as a practical aspect, this is what is observed.

Second, storage system technology urgently needs to advance much more rapidly than it is advancing now to meet the challenge of information access. Perhaps the test of when storage system technology sufficiently advances is that "Any sufficiently advanced technology is indistinguishable from magic"[3]. That is, when information appears magically on request, success will have been achieved.

There are many symptoms and documented aspects of this urgent need, more than this paper can accommodate and stay on topic. As example, one aspect cited by Jim Gray is that storage capacities are increasing at the rate of 100x/decade while storage throughput is improving at only 10x/decade [4].

Third, raw storage capacity is seldom sized properly with computing capabilities, even though there are direct relationships between processing power, memory, and storage. The Dept. of Energy's Accelerated Strategic Computing Initiative suggests that you need 300 bytes of archival data for each sustained megaFLOPS. This points to one of the major factors in growth of storage (information access) demands: the growth in processing power. Processing today goes on with processors of 70 to 100 million transistors, and yet single processors of more than a billion transistors is forecasted for the next decade [5]. And "QuBit" processors are contemplated with speeds "millions" of times greater [6].

And while information is being generated at rates in step with the rapidly improving technologies of processors, and storage media and hardware are improving rapidly as well, storage system software and architecture lag dangerously behind. This is in part due to the overall crisis in software [7,8]

At the same time the specialized demands for secure access to information in highly distributed and heterogeneous environments are growing.

Traditional approaches are not sufficient to meet these needs, and revolutionary or rapid evolutionary changes are needed in storage system design and software. The information which will need to be stored and accessed in the next year or two will equal twice all the data ever stored before now.

Far from leading the target, in this case the "disaster recovery" with which we should be concerned is the loss, perhaps permanent, to access of information purchased with the expense of computing and human resources. To quote John Carlin, the U.S. National Archivist, will the country lose its memory [9]?

4 Promising Trends

The IEEE is re-inventing itself in several respects, including embracing techniques that speed up every step of the standards process. In addition, the IEEE Industry Standards and Technology Organization (ISTO), affiliated with the IEEE and the IEEE Standards Association (IEEE-SA) and just formed in 1999, is moving away from the sales of standards reprints to the intelligent position of just making them publicly reviewable. The ISTO embraces corporate entities as participants, and attempts to act much like consortia in industry.

The cooperation among groups developing "standards" in storage is very promising. This is truly an area in which we must all hang together or... The greatest competitor for products and solutions against storage efforts is not within the arena of storage at all. It is the preponderance of interest in faster processors, wider and faster networks.

The trend away from "SAD" storage (server-attached disks [10]) toward Storage Area Networks (SANs) and onward to Net Attached Storage (NAS) carries great hope with it. Using idle embedded cycles on computationally rich storage devices for the remote execution of some applications serves to reduce the motion of data and fits better in highly distributed environments.

The extended abstraction of the present storage object called "file" to even more abstract storage objects will both serve the user's need for magical transparency and the global need for distributed computing [11,12].

Finally, the movement toward opensource and away from restrictive licensing will greatly aid the need for interoperability, and foster an economic bonanza for the storage industry.

5 Summary

The SSSC is balloting the first storage system standards in the world, and these are the first five of ten Media Management System (MMS) standards. These standards and the future work of the IEEE will help bring a consistent approach to building interoperable storage systems, and to address the emergency need for improvements in the systems we all use to access information.

References

[1] Many documents, such as versions 4 and 5 of the MSSRM are available for public reading at http://www.ssswg.org, and serve only as working materials since the SSSC does not intend to standardize these.

[2] G. Peck, editor, IEEE Draft P1244.1/1.15 Standard for Media Management System (MMS) Architecture, December 1999

[3] Arthur C. Clarke's Third Law.

[4] J. Gray, P. Shenoy, "Rules of Thumb in Data Engineering", IEEE Conference on Data Engineering, San Diego, April 2000.

[5] N. Ranganathan, "TCVLSI Activities: A Message from the TC Chair", IEEE Computer Society TCVLSI Technical Bulletin, Fall 1998.

[6] L. Grover, "Quantum Computing". *The Sciences*, July/August 1999, pp. 24-30.

[7] T. Lewis, "Software Architectures: Divine plan or digital Darwinism?", IEEE Computer, August 1996.

[8] D. Taylor, "Beating the Software Crisis", *Object Oriented Technology: A Manager's Guide*. Addison-Wesley, Reading, Massachussets, 1994.

[9] "Will the Country Lose Its Memory?", USA Today, September 25, 1998.

[10] G. Gibson, et al., http://www.pdl.cs.cmu.edu

[11] G. Peck, Storage Object Presentation at NASA/Goddard Mass Storage Conference, March 1998. http://www.peck.com/~geoff/storage-peck-980325.pdf

[12] J. Nielsen, "The Impending Demise of the File System," IEEE Software, March 1996.

Compact Holographic Read/Write Memory

Wenhai Liu and Demetri Psaltis
Department of Electrical Engineering
California Institute of Technology
Pasadena, CA 91125
Phone: +1-626-395-4843
Fax: +1-626-568-8437
{ wliu, psaltis }@sunoptics.caltech.edu

Abstract— We examine the primary challenges for building a practical and competitive holographic random access memory (HRAM) system, specifically size, speed, and cost. We show that a fast HRAM system can be implemented with a compact architecture by incorporating conjugate readout, a smart-pixel array, and a linear array of laser diodes. It provides faster random access time than hard disk (100 microseconds or less) and similar bandwidth as silicon storage with lower cost. Preliminary experimental results support the feasibility of this architecture. Our analysis shows that in order for the HRAM to become competitive, the principal tasks will be to reduce spatial light modulator (SLM) and detector pixel sizes to 1 μm, increase the output power of compact visible-wavelength lasers to several hundred milliwatts, and develop ways to raise the sensitivity of holographic media to the order of 1 cm/J.

1. Introduction

Holography memory is a potential technology that can provide very large storage density and high speed. The theoretical storage capacity of this technology is on the order of V/λ^3 [1] (where V is the volume of the holographic medium and λ is the wavelength of light), or equivalently, a storage density limit of about one bit per cubic wavelength. Furthermore, holography has the inherent advantage of massive parallelism. Unlike conventional storage media such as magnetic hard disks and CD-ROMs, which access only one bit at a time, each access of a holographic memory yields an entire data page -- potentially megabits at a time.

Figure 1 shows a typical angle-multiplexed holographic memory in the 90^0 geometry. Information is recorded in the holographic medium through the interference of two coherent beams of light. The information-carrying signal beam and the interfering reference beam cause an index grating (the hologram) to be written in the material through the electro-optic effect. If the hologram is subsequently illuminated with one of the original writing beams, light is diffracted from the grating in such a way that the second beam is reproduced.

Due to Bragg effects, many holograms can be multiplexed within the same volume of material by slightly changing the angle of the reference beam with each new data page. Thousands of holograms can be multiplexed this way in a small volume of crystal, offering the potential of very high storage densities.

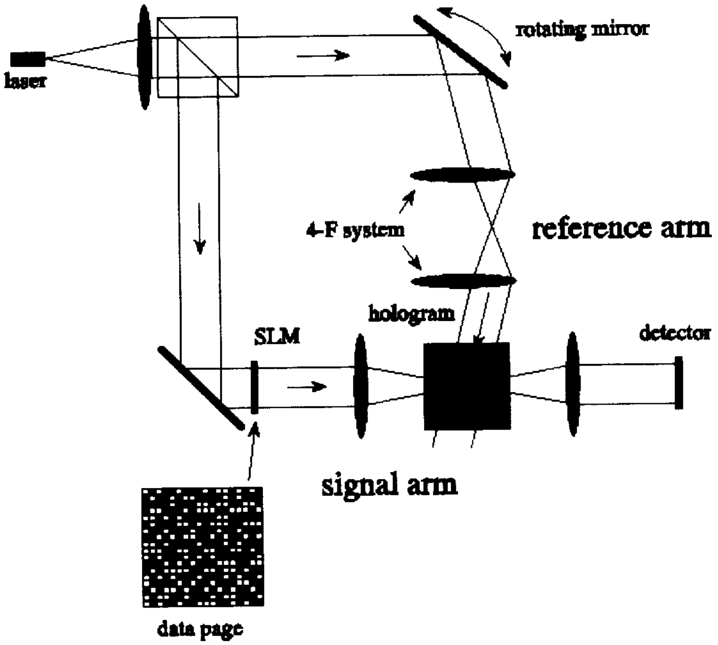


Figure 1. Typical angle-multiplexed holographic memory.

Silicon (1×1 cm^2)	$125
LiNbO$_3$ (1×1×1 cm^3)	$ 10
Liquid Crystal	$ 5
Beamsplitters and lens	$ 6
LD array (500)	$ 25 - 100
Total	$ 171 - 246

Table 1. Estimated cost of components in the holographic memory module, assuming production in large quantities

In the figure shown, the signal path consists of a spatial light modulator (SLM) and detector array with a 4-F imaging system between them, and the reference path uses another 4-F lens system in combination with a rotating mirror to provide the angular tilt to the reference beam. Recent work has shown the ability to store and retrieve many thousands of holograms [2,3]. Much of the progress that has been made can be attributed to advancements in our understanding of ways to take advantage of the Bragg selectivity of 3-D recording to multiplex holograms, as well as continued research in holographic material properties and dynamics.

In this paper, we describe a holographic random access memory (HRAM) with phase conjugate reconstruction and present experimental results from this architecture. It has the potential of faster random access time than hard disk (100 microseconds or less) and similar bandwidth as silicon storage with lower cost. The phase conjugation leads to high-resolution signal image recovery with a compact and inexpensive optical system. And we believe that HRAM can be a competitive memory technology if optoelectronics technology can achieve the following three milestones in the next few years:

1). small SLM and detector pixel sizes on the order of 1 μm;

2). high recording sensitivity of the holographic material with no more than 1 J/cm^2 to reach saturation;

3). inexpensive high spatial density laser diodes with at least 500 mW of output power in the near-infrared or visible wavelength.

2. Conjugate Readout Method

Despite the high theoretical limit on the storage density of volume holographic storage (one bit per cubic wavelength of material), the practical implementation of holographic systems is often bulky due to the large space occupied by the various components that are necessary to provide the recording and readout mechanisms for the crystal. The system of Figure 1 is fairly simple with a relatively small number of components, however the spacing requirements of the imaging lenses imposes constraints on how closely these components can be placed. For example, assuming SLM and detector array dimensions of 1cm and high quality lenses with F/#=1, the focal distance between the arrays, lenses, and crystal must also be at least 1cm. The system of Figure 1 would then occupy a volume of approximately 6cmx5cmx1cm, which is 30 times larger than the volume of the recording material.

The reason we normally need to place lenses within the signal path is to undo the effects of diffraction. When we record a hologram of the signal beam diverging from the input SLM and reconstruct it with the original reference beam, we produce a virtual image of the input data page and thus require a lens to refocus it onto the detector array. We can eliminate the lens system between the SLM and detector array if we reconstruct a real image instead of a virtual one. One way to do this is to use phase conjugate readout [4-6] as illustrated in Figure 2. Using this method, a hologram is recorded in the usual manner between the signal and reference beams, but the hologram is read out with the phase conjugate of the reference beam, propagating in the opposite direction as the one used for recording. This causes the signal reconstruction from the hologram to propagate back along the direction from which it originally came, reversing the original signal diffraction, and refocusing exactly at the plane of the SLM array. To generate the conjugate reference we may use a phase-conjugate mirror [5], or in the case of a planar reference beam, we may simply use a counter-propagating plane wave at the each angle.

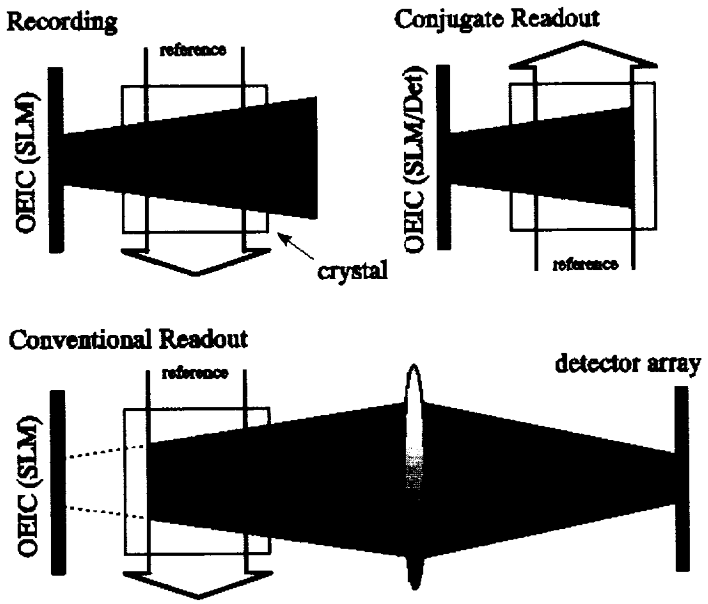


Figure 2. Comparison of phase conjugate readout method with conventional readout using imaging lenses.

Experimentally, we compared the reconstructed image fidelity that can be obtained with conventional reconstruction using high-quality, custom-designed lenses to the image fidelity we get with the conjugate readout method of planar reference beams. An SLM and detector array each with pixel spacing of 24μm were used for these tests, allowing one-to-one matching of the SLM and detector pixels. Both methods yielded SNR (signal-to-noise ratio) values ranging from about 3.8 to 4.5, verifying that the conjugate readout method produces results that can only be achieved with quality lenses, while using a much more compact and inexpensive optical system.

Phase conjugation read-out not only eliminates the lenses and associated path lengths that are normally required in the signal path, it also provides a possibility to record and reconstruction signal beams with high spatial frequencies. The holographic recording and reconstruction possesses a basic spatial frequency bandwidth for the holograms, which limits the smallest feature size to be record and reconstructed. The theoretical calculation and experimental measurements indicates a width bandwidth for holographic recording and reconstruction in the photorefractive $LiNbO_3$. This makes it possible to record and reconstruction holograms with very small pixel sizes, which has important effects on the system storage density and cost efficiency as discussed in section 3. Figure 3 shows the theoretical simulation of the holographic recording and reconstruction bandwidth inside a $LiNbO_3$ with 90^0 geometry, with the consideration of the interface losses. The hologram strength is a function of the spatial frequency, or the incident angle of the signal beams due to the different grating period, orientation and interference modulation depth [7,8]. The experimental measurement of the bandwidth confirms the theoretical prediction as shown in figure 3. Holograms with sub-micron pixels were recorded and conjugate

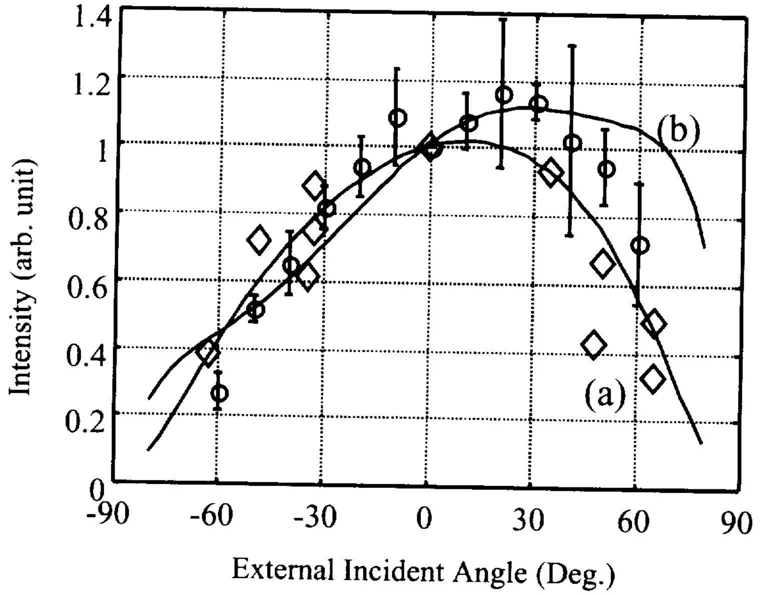


Figure 3. (a) The experimental data (diamond) and the theoretical calculation of holographic efficiency in the signal reference plane; (b) the experimental data (circle) and the theoretical calculation of the holographic efficiency out of signal reference plane.

reconstructed, which further demonstrated the resolving power of the phase conjugate reconstruction. Figure 4 shows the mask image and the phase conjugation. There are no image degradation detected for the hologram reconstruction from the direct image of the mask.

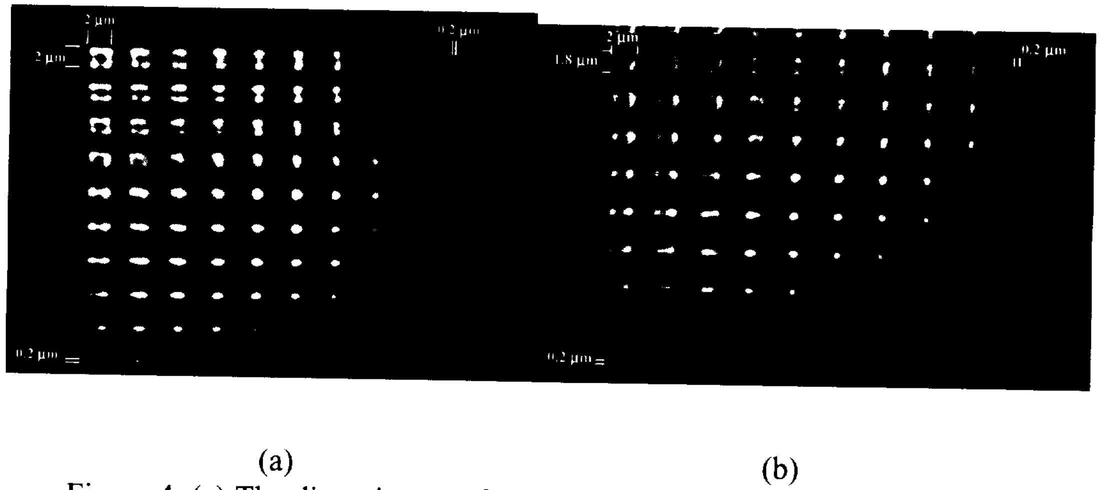


Figure 4. (a) The direct image of a resolution photo mask with pixels from 2x2 μm^2 down to 0.2x0.2 μm^2. (b) The holographic phase conjugate reconstruction of the photo mask. Both images were magnified by a Nikon objective lens with NA=0.65.

3. Compact Fast Access Architecture

While conjugate readout eliminates the lenses in the signal path of the memory system, we still require a compact design to rapidly deflect the reference beam for multiplexing purposes. The 4-F system shown in Figure 1, while reliable, is bulky and slow due to the limited mechanical speed of the rotating mirror.

With the recent development of compact laser emitters, such as laser diodes and Vertical-Cavity Surface-Emitting Laser (VCSEL) devices [9,10], it has become feasible to consider the possibility of incorporating arrays of hundreds of microscopic laser sources in a holographic memory. We can then design a system in which each angle multiplexed hologram is addressed by a dedicated laser source. This architecture is shown in Figure 5. A Fourier transforming lens is used to convert the spatial shifts between the laser elements into angularly offset plane waves incident on the crystal. In this implementation, the time it takes to produce the proper read-out reference beam is determined by the switching time of the laser sources, which is in the nanosecond regime. Using a 1cm-thick crystal and a wavelength of 630nm, the first null of the angular selectivity function occurs at an angular spacing of 0.0036°. Using a lens with a focal length of 2cm would require the laser elements to be placed only 1.3 μm apart to produce this angular separation. In practice, we would separate them by 10 μm or more in order to reduce interpage crosstalk while also making the array easier to fabricate.

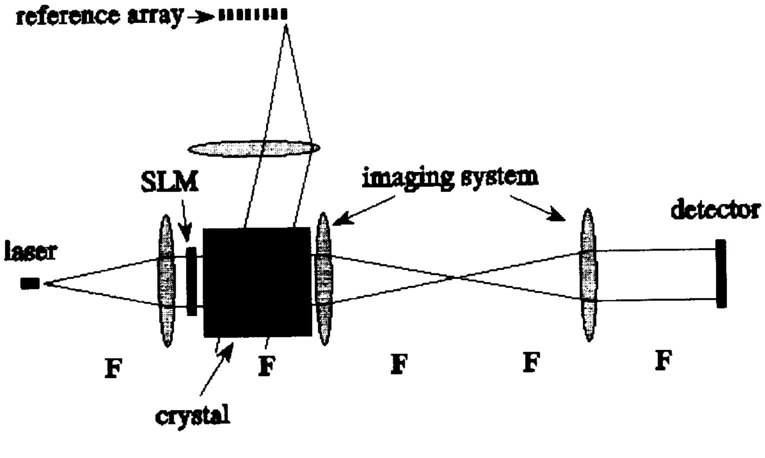


Figure 5. Use of a laser array in the reference arm of an angle multiplexed memory for fast page access.

This approach is also compatible with the conjugate readout method as shown in Figure 6. With a properly aligned laser array and a mirror placed on the opposite face of the crystal such that it lies at the focus of the Fourier transforming lens, the proper conjugate beam can be generated with the symmetrically opposite laser source. A beamsplitter must also be introduced to accommodate both the SLM and detector devices. The combination of conjugate read-out in the signal beam path and laser diode arrays in the reference beam path results in a very compact holographic memory module with fast access. It is not completely lensless, since one lens still remains in the system, but such a lens would be required to collimate the laser source in any optical system that uses plane waves.

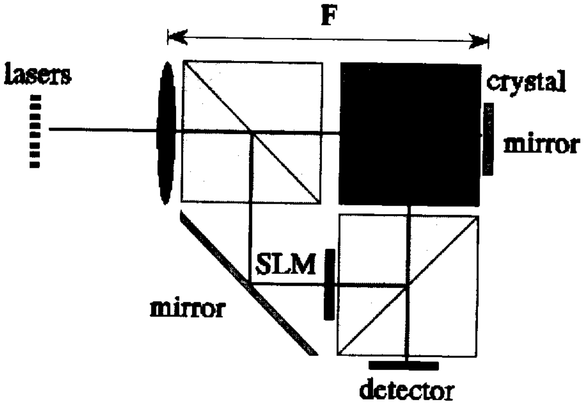


Figure 6. Compact memory module with phase conjugation incorporating separate SLM and detector devices.

1. Cost

The cost is perhaps the most important metric for accessing the commercialization prospects of HRAM. We will compare the costs of HRAM and DRAM with reference to Figure 7. We can think of HRAM as a holographic module that sits on top of a page of DRAM. The ability of the HRAM to multiplex holograms essentially allows us to store M DRAM data page, hence saving us the cost of fabricating M-1 additional DRAM pages in silicon. However, it is not quite that simple. First, the silicon device in the HRAM is not really a DRAM page, but rather the DHR chip described earlier or and SLM/detector pair. Because of the necessity of fabricating SLM and detector pixels (either in the same optoelectronic device or in two separate devices), the page density of the DHR will be less than that of a true DRAM. We call this ration of the page densities R>1. Moreover, the cost of the holographic module also includes the optical elements C_{Opt}, and laser diode array C_{LD}, in addition to the cost of the silicon C_{Si}. The projected costs of the optical elements (assuming production in large quantities) are summarized in Table 1. We assume the silicon cost to be purely based on area, and therefore will be identical to that for an equal-sized DRAM. The cost of the laser array is not well known at this time, since large arrays have not yet been produced for visible wavelengths; however, we estimate the cost to be in the range of $25-$100 per array.

The cost ratio per megabyte CR of holographic memory to the silicon storage will be:

$$CR = \frac{C_{Si} + C_{Opt} + C_{VCSEL}}{C_{Si}} \cdot \frac{R}{M} \qquad (1)$$

where the R is the pixel area ratio of the SLM and detector to the silicon area of each bit on DRAM, M is the number of holograms multiplexed in the crystal on top of the silicon. With the fixed cost of silicon area C_{Si}, optical elements C_{Opt}, and LD array C_{LD}, the key to have a small cost ratio CR is to have small R and large M, which means a high storage density in holographic memory comparing with the DRAM.

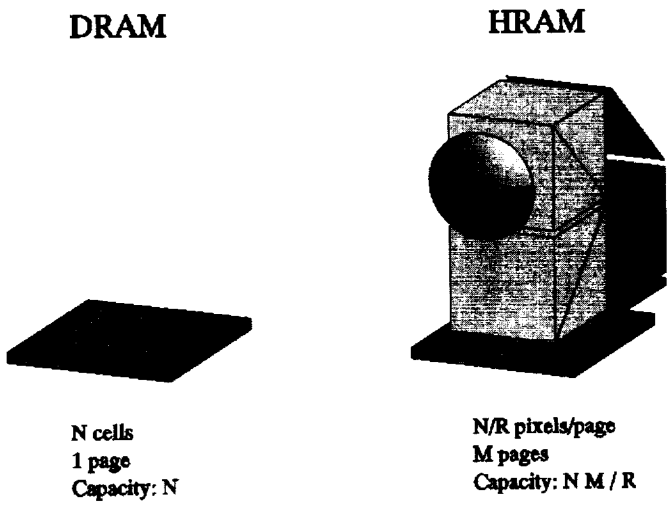


Figure 7. Model for cost comparison between HRAM and DRAM.

The number of holograms to be recorded and readout with reasonable bit error rate, is limited by the dynamic range and sensitivity, or the M/# of the material. Recording and reading 10,000 holograms at one location of a $LiNbO_3$ crystal was demonstrated with a similar system. However limited by the material M/# [11], the LD array number and power, and reasonable recording/readout rates, it is practical to keep M below 1000.

For current commercial SLM and detector array, the pixel area is typically $4x4\mu m^2$. And the current commercial DRAM is 1 μm^2/bit, which leads R=16. With typical M=1000, we have R/M=1.6%, which leads to a small and promising CR. However if the DRAM keeps the history trend as the NTRS97 [12] projected, the DRAM cell will be $0.04\mu m^2$/bit in 2006. To keep the R around 25, the pixel size of the holographic data pages has to be $1x1\mu m^2$ or even smaller, which is previous proved achievable for the holographic memory system.

Figure 8 shows the experimental demonstration of conjugate hologram reconstruction of a 1x1 μm^2 random pixel mask as SLM, which gives Bit Error Rate (BER) at $7x10^{-5}$. This finite BER indicates the requirement for error correction coding for the holographic memory.

Comparing the cost per megabyte for the DRAM projection of 42 cents/Mbyte in 2006, we have the cost estimation for the holographic module in table 1, where we assume the same cost per area for silicon usage. With the R=25 for $1x1\mu m^2$ pixel size and M=500, the cost for holographic memory is around 4 cents/Mbyte, one order of magnitude lower than the DRAM in 2006.

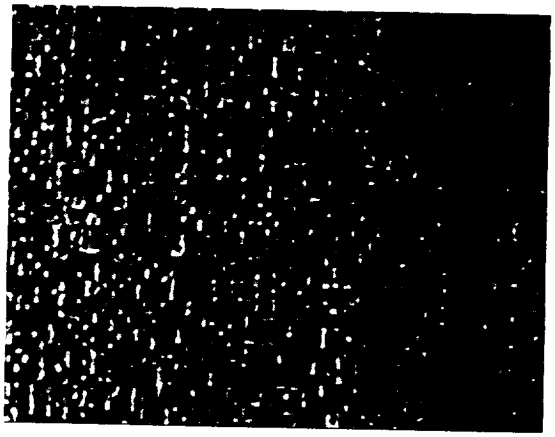


Figure 8. The phase conjugate reconstruction of 1x1 μm^2 random data mask holograms.

2. System volume density

An analysis of the system storage density of the holographic memory module (including the recording medium and all the optical components) in Figure 6 shows that the module storage density peaks at about 40Mb/cm^3 for an optimum pixel size of 5μm. There is an optimum pixel size because as the pixel size decreases the light in the signal path spreads more due to diffraction, causing us to use larger apertures for the crystal and beamsplitters.

A more aggressive concept for minimizing the volume is shown in Figure 9. This design relies on total internal reflection to contain the beam diffraction within the boundaries of the module, so that the optical elements can be made the same size as the SLM array. Preliminary experiments indicate that accurate recordings are obtained using the internally reflected light. In this case, the system density can be raised to the order of 2Gb/cm^3, if SLM pixel sizes fall to 1μm. At this density, a gigabyte of data could be stored in a single module with a volume of 1x2x2 cm^3. The challenges in achieving such high densities are several: Development of SLM and detectors with 1 micron pixels, designing the optical system so that we have uniform illumination throughout, and further characterization of the performance of the module when the light is allowed to undergo total internal reflection.

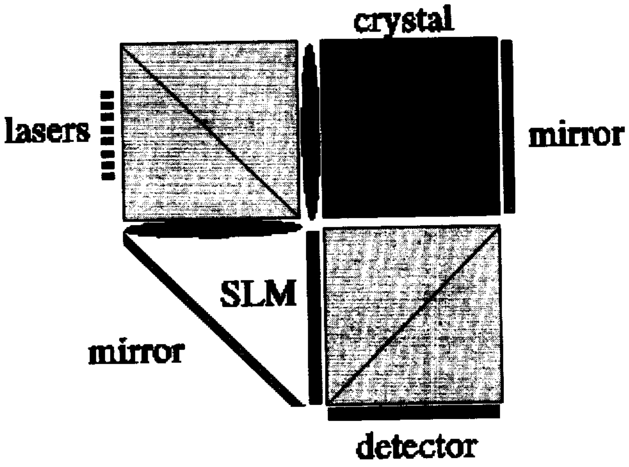


Figure 9. Variation of compact memory module for minimum volume.

3. Readout and recording rate

Since the laser diode array discussed in the previous section allows us to switch between multiplexed data pages with negligible delay (on the order of nanoseconds), the random access time and the readout rate become limited by the required integration time of the detector. We can write the integration time as

$$\textbf{Detector integration time} = \frac{\mathrm{Neh}\nu\mathrm{N}^2}{\left(\frac{\mathrm{M}/\#}{\mathrm{M}}\right)^2 \mathrm{P_i}} \qquad (2)$$

where Ne is the number of electrons per pixel that we need to integrate for the given detector sensitivity and level of background noise, h is Planck's constant (6.63x10^{-34} J•s), v is the light frequency, N^2 is the total number of pixels in the detector array, M/# is the system metric [11] of the holographic medium, M is the number of multiplexed holograms, and P_i is the incident readout power. For example, if we use a crystal of M/#=10 to record 500 holograms of a 1000x1000 pixel array, and we read out with 100mW of laser power, requiring 300 electrons per pixel, the integration time, and hence the random access time, would be 2.4 μs. This corresponds to a sustained readout transfer rate, from the hologram to the silicon detectors, of 53GB/s.

We can write the recording rate of the memory module as

$$\text{Recording rate} = \frac{N^2 ISLp}{(M/\#)/M} \qquad (3)$$

where N^2 is the total number of pixels per data page, I is the incident recording intensity, S is the sensitivity per unit length of the recording medium, L is the crystal thickness, and p is the light efficiency of the SLM. Again assuming a crystal of M/#=10 to record 500 holograms of a 1000x 1000 pixel array, with I=100mW/cm^2, S=0.1cm/J, L=1cm, and p=50%, we obtain a recording rate of 31kB/s. This is typical for experiments currently performed. Increasing the recording rate to make it comparable to the read-out rate is highly desirable for a practical system. We will discuss possible methods for achieving this goal later on.

4. Roadmap for A Competitive HRAM Technology

From the preceding discussion, we can summarize that it would be commercially competitive for a holographic memory system with parameters: M/#=10, S=1 cm/J, laser diode array with output 500 mW/cm^2 for each element and 1000 holograms storage of 10,000x10,000 pixels each page. This module expect to deliver a recording rate >100 Mbyte/sec, access time <100 μsec, and cost <$0.04 /Mbyte. For comparison, the DRAM is projected to be $0.40/MB in 2006[12].

Presently, the greatest challenge for the HRAM is to raise its recording rate by several orders of magnitude. To achieve this, we must rely in part on improvements in SLM technology to bring the pixel sizes down to 1 μm. This will allow us to increase the size of each data page to 10,000x10,000 pixels while still holding the array size to about 1cm^2. By increasing the page size in this way, we immediately gain two orders of magnitude in the sustained recording rate due to the increased parallelism. Experimentally, we have used a mask fabricated with e-beam, lithography to record and reconstruct data pages with 1 μm pixels holographically with good image fidelity. Figure 10 shows an experimental measurement of the SNR for various pixel size holograms. The reconstruction for 1 μm pixels gives SNR =4.

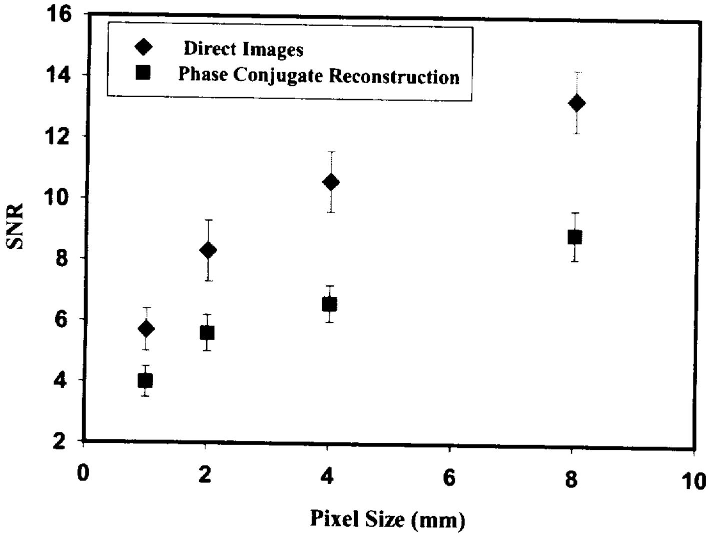


Figure 10. The SNR for the direct images and the holographic phase conjugate reconstruction of random binary data of pixel size from 8x8 down to 1x1 μm^2.

Reducing the pixel sizes to 1 μm is not only necessary for raising the recording rate, but also for maintaining the cost advantage of HRAM over DRAM. By 2006, the DRAM cell pitch is expected to fall to 0.2 μm [12]. By bringing the SLM pixel pitch down to 1 μm, we can hold the factor R in Equation (1) at 25, and beat the cost of DRAM by an order of magnitude.

Because the HRAM readout rate is limited by the electronic transfer rate out of the detector chip, we can afford to give up some readout speed in favor of increasing the recording speed. We do this by intentionally reducing the strength of the holograms so that we can record with shorter exposures, at the cost of increasing the detector integration time. In Equations (2) and (3), this is equivalent to recording in a medium with lower M/#, but without sacrificing sensitivity. Unfortunately, as we increase the required integration time we increase at the same time the random access time of the memory. In order to maintain an advantage of at least an order of magnitude over magnetic disks in random access time, we can only afford to increase the integration time to several hundreds of microseconds.

Other opportunities for increasing the recording rate can arise from improvements in laser output powers or from improving the sensitivity of the recording materials. Compact laser arrays with outputs of 500mW per emitter may be possible by 2006, or if not, we may consider sharing a larger, more powerful tunable laser among multiple HRAM modules. Increasing material sensitivity presents more of a challenge. The sensitivity of $LiNbO_3$:Fe, by far the most commonly used recording material today, Is typically around 0.02cm/J in the 90-degree geometry. In order to get recording rates on the order of 100 MB/s, we must find ways to boost the material sensitivity to about 1cm/J by improving lithium niobate's properties. For instance, switching to transmission geometry and increasing the doping level result in large increases in M/# which can be traded for better sensitivity as we discussed previously. Alternatively, we can switch to alternative materials such as doubly doped $LiNbO_3$, in which sensitivity $S > 1$ cm/J was measured in the transmission geometry. However, this is a relatively new material and much more expensive at present.

5. Conclusion

In order to develop a competing HRAM technology, three main challenges must be met: reducing pixel size to 1 μm, producing arrays of high-power laser diodes, and increasing the sensitivity of holographic recording media. Each of these tasks is difficult, but if they can be achieved by 2006, then the projected HRAM performance levels shown in previous section become feasible. Attaining these goals will position the HRAM as a viable alternative memory technology to magnetic storage, offering performance that is at least one order of magnitude better in terms of random access and transfer rate than magnetic hard disks, and at least one tenth the cost compared to fabricating an equivalent memory in DRAM.

Acknowledgment:
The authors would like to thank Drs. E. Chuang, J-J. Drolet, G. Barbastathis, and A. Adibi for helpful discussions.

References:

[1] P.J. van Heerden, "Theory of optical information storage in solids," *Applied Optics*, vol.2, no.4, pp.393--400, 1963.

[2] F. H. Mok, "Angle-multiplexed storage of 5000 holograms in Lithium Niobate," *Optics Letters*, vol. 18, no. 11, pp. 915-917, June 1993.

[3] X. An, D. Psaltis, G. W. Burr, "Thermal fixing of 10,000 holograms in LiNbO3 : Fe," *APPL OPTICS* 38: (2) 386-393 JAN 10 1999.

[4] J. Drolet, G. Barbastathis, J. Patel, and D. Psaltis, "Liquid crystal devices for volume holographic memories," in *Proc. OSA Annu. Meeting,* Portland, OR, Sept. 1995.

[5] J. Drolet, E. Chuang, G. Barbastathis, D. Psaltis, "Compact, integrated dynamic holographic memory with refreshed holograms," *OPTICS LETTERS*, 22: (8) 552-554 APR 15 1997

[6] Z. O. Feng and K. Sayano, "Compact read-only memory with lensless phase-conjugate holograms," *Optics Letters*, vol. 21, pp.1295--1297, August 1996

[7] H. Zhou, F. Zhao, and F. Yu, "Angle-dependent diffraction efficiency in a thick photorefractive hologram", *Applied Optics*, Vol. 34, No. 8, pp. 1303-1309, Mar. 1995.

[8] W. Liu, and D. Psaltis, "Pixel size limit in holographic memories", *optics letters,* vol. 24: pp. 1340-1342 Oct. 1999.

[9] J. L. Jewell, K. F. Huang, K. Tai, Y.H. Lee, R. J. Fischer, S. L. Mccall, and A. Y. Cho, "Vertical cavity single quantum well laser," *Applied Physics Letters*, vol.55, no.5, pp.424--426, 1989.

[10] W. W. Chow, K. D. Choquette, M. H. Crawford, K. L. Lear, and G. R. Hadley, "Design, fabrication, and performance of infrared and visible vertical-cavity surface-emitting lasers," *IEEE Journal of Quantum Electronics*, vol.33, pp.1810-1824, October 1997.

[11] F.H. Mok, G.W. Burr, and D. Psaltis, "System metric for holographic memory systems," *Optics Letters,* 21: (12) 896-898 JUN 15 1996.

[12] "The national technology roadmap for semiconductors," Semiconductor Industry Association, Tech. Rep., San Jose, Ca 95110, 1997.

High Density Holographic Data Storage

Lisa Dhar, Kevin Curtis, Arturo Hale, Melinda Schnoes, William Wilson, Michael Tackitt, Adrian Hill, Marcia Schilling, and Howard Katz
Bell Laboratories, Lucent Technologies
600 Mountain Avenue, Murray Hill NJ 07974
ldhar@lucent.com, kevincurtis@lucent.com
tel. +1-908-582-5035
fax +1-908-582-3958

Abstract

The demand for increases in the capacity and speed of data storage tests the limits of conventional technologies and drives the search for new approaches. Optical holography has long held the promise of storage densities and data transfer rates far greater than those of traditional magnetic and optical systems. In the past, its realization has been frustrated by the lack of availability of suitable system components, the complexity of holographic multiplexing strategies, and perhaps most importantly, the absence of recording materials that satisfied the stringent requirements of holographic data storage. Here we report on the design and development of a high-performance photopolymer recording medium and on advances in the design of a holographic storage system that have enabled demonstrations of storage densities as high as 31.5 channel Gbits/in^2. We believe these results will provide the foundation for a practically realizable, high capacity storage system with fast transfer rates and low-cost, removable recording media.

1. Introduction: How Holographic Data Storage Works

The compelling features of volume holography, rapid transfer rates and ultrahigh storage densities, arise from two basic properties: (i) the writing and reading of bits of data occur in a parallel, page-wise fashion, unlike the serial read-write processes of most storage technologies; (ii) the three-dimensional nature of holography enables the storage of many of these pages of data within the same volume of a recording medium, thereby enabling densities far beyond the diffraction limit of conventional optical technologies.

In holographic storage, light from a coherent laser source is split into two beams, signal (data-carrying) and reference beams. These two beams are spatially overlapped through the volume of a photosensitive storage medium producing an optical interference pattern that is imaged within the medium. This process records information contained in the phase and amplitude of the two beams. The optical interference pattern typically induces modulations in the refractive index of the recording material yielding diffractive volume gratings. A schematic of a typical holographic storage system is shown in Figure 1.

The reference beam is used during readout to diffract off of the recorded grating and reconstruct the information that was contained in the signal beam. The readout of data depends sensitively upon the characteristics of the reference beam. By varying the reference beam, for example by changing its angle of incidence or wavelength, different holograms can be recorded in the same volume of material and read out by applying a reference beam identical to that used during writing. The number of holograms that can be overlapped or multiplexed within a volume typically depends on the thickness of the material – the thicker the material, the higher the selectivity of the material and therefore the greater the number of holograms that can be multiplexed.

Information to be stored is digitized with appropriate error correction and channel modulation. The digital data are arranged into pages or large arrays of bits. The 0's and 1's of the data pages are translated into pixels of a spatial light modulator that either block or transmit light. The light of the signal beam traversing through the modulator is therefore encoded with the "checkerboard" pattern of the data page. Each of the pages of data is recorded as the signal and reference beams interfere through the volume of the storage material. When the appropriate reference beam diffracts off of stored volume gratings within the material, it recreates the array of bits which is projected onto a pixelated detector that reads the data in parallel. The recovered data pages are then processed using the channel and error correction codes to reconstruct the original information.

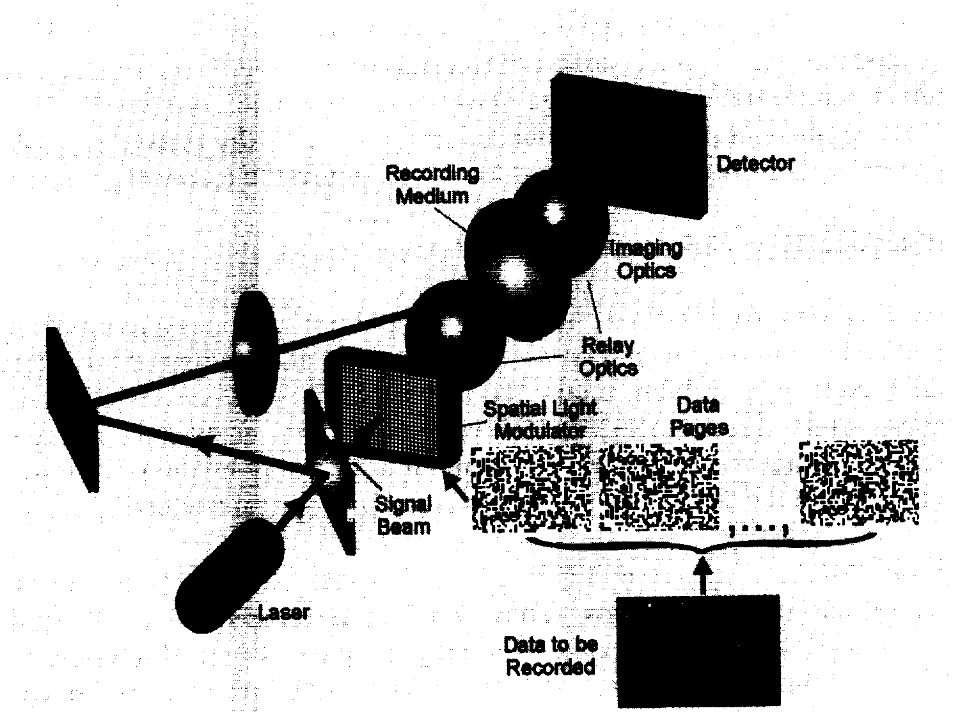


Figure 1. Schematic of a holographic storage system. Light from the source laser is split between the reference and signal arm. The signal arm is encoded with the data to be stored. The signal and reference arms overlap in the recording medium to produce diffractive gratings that are readout by the reference arm. The readout data is imaged onto a pixelated detector.

2. Holographic Recording Media – Photopolymer Materials

One of the major challenges in the area of holographic data storage has been the development of suitable storage materials. Holographic media must satisfy stringent criteria, including high dynamic range, high photosensitivity, dimensional stability, optical clarity and flatness, nondestructive readout, millimeter thickness, and environmental and thermal stability.

While many materials have been considered as media for holographic storage, most suffer from disadvantages that preclude their use in practical systems. Lithium niobate, the traditional choice for holography, exhibits the dimensional stability required for digital data storage, yet suffers from low dynamic range and poor photosensitivity and typically exhibits volatile readout. More recently, photorefractive polymers have shown promise as holographic media, but require the application of electric fields which become prohibitively large for thick media. Photochromic materials can be used for rewritable applications but are characterized by low photosensitivity and limited dynamic range.

Photopolymer materials are attractive candidates for write-once-read-many (WORM) times data storage applications because they can be designed to have large modulations in their refractive index and high photosensitivity, record permanent holograms, and be easily processed. Most of the currently available holographic photopolymers, however, have been optimized for display applications. Typically, these materials can be used only as thin (≤100-200 μm) layers and often exhibit significant dimensional and bulk refractive index changes due to the polymerization of the photosensitive species that occurs during recording.

To better meet the needs of holographic data storage, we designed a new type of polymer system that is composed of two independently polymerizable and compatible chemical systems: low refractive index matrix precursors and high refractive index photopolymerizable monomers [1]. The matrix of our media is formed by an *in-situ* polymerization to yield a cross-linked network in the presence of the photopolymerizable monomers, which remain dissolved and unreacted. Recording of holograms occurs through a spatial pattern of polymerization of the photosensitive species that mimics the optical interference pattern generated during writing – polymerization is induced in the light intensity maxima of the interference pattern while no polymerization occurs in the nulls. The concentration gradient that results from this patterned polymerization leads to diffusion of the unpolymerized species which creates a refractive index modulation that is determined by the difference between the refractive indices of the photosensitive component and the matrix. The most important aspects of this strategy which yield high performance holographic storage media are (i) preforming the matrix *in-situ* which allows media to be shaped into the required thick and flat formats, (ii) the creation of a cross-linked matrix as a support structure for stable holographic gratings, (iii) the choice of compatible matrix and monomer systems to yield media with good optical clarity and low levels of light scattering, and (iv) the design of independent matrix and monomer systems so as to avoid cross reactions that dilute the refractive index contrast. The fourth point ensures that the low refractive index of the "background" matrix is not detrimentally raised by the premature polymerization of the high refractive index monomer. Media with high refractive index contrast can be fabricated using small amounts of the high index monomer thereby minimizing the photopolymerization-induced dimensional and bulk refractive index changes that occur during recording.

To prepare samples for holographic recording, resins, consisting of the matrix precursors, photopolymerizable monomers, and a visible light sensitive photoinitiator, were dispensed between two optically flat glass slides. The *in-situ* room temperature formation of the matrix allowed routine fabrication of high optical quality media with polymer thicknesses between 200 μm to 1.5 mm. A transmission interferogram of a typical 1 mm thick (polymer thickness) is shown in Fig. 2(a) with a surface plot of the variation in the optical flatness shown in Fig. 2(b); the data show flatness within 500 Å (λ/10) over the three inch diameter sample.

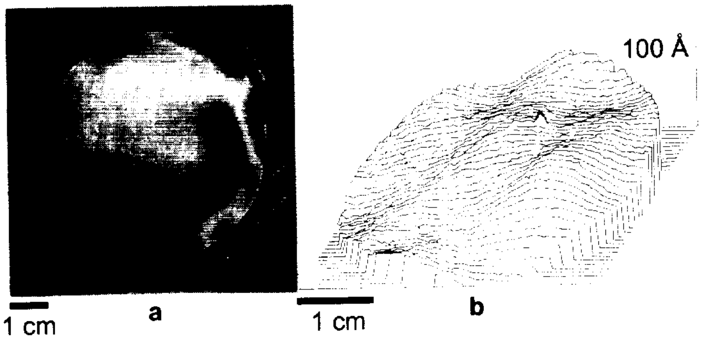


Figure 2(a) Transmission interferogram of a typical 1 mm thick photopolymer medium. **(b)** Surface plot of the variation in the optical thickness of the inner 4 cm of the sample. The optical flatness of the sample varies less than λ/10/cm (~500 A) over the entire area of the medium.

In order to enable simultaneously high densities and rapid recovery rates, a material must have the dynamic range to support large numbers of holograms with sufficiently high diffraction efficiencies. The dynamic range of a medium, which depends both on the magnitude of the modulation in its refractive index and its thickness, is typically characterized by a parameter, M/# [2]. The M/# is defined to be the number of $\sum_{i=1}^{N}\sqrt{\eta_i}$, where N is the maximum number of holograms that can be stored in a volume of the material and η is the diffraction efficiency of each hologram. The M/# of iron-doped lithium niobate is typically 1-1.5 for 1 cm thick crystals. It is commonly believed that M/#'s at least an order of magnitude higher will be required to achieve compelling storage densities and transfer rates. Polymer systems developed at Bell Laboratories have yielded M/#'s as high as 42 in ~1 mm thick formats. The high dynamic range of our polymer media is achieved while controlling the dimensional and bulk refractive index changes that accompany the recording-induced polymerization of the photosensitive species. Recording media must undergo only limited changes in their dimensions and bulk refractive index as these changes can degrade the fidelity of data recovery and ultimately limit the storage density of a material. Our design strategy enables us to optimize the response by minimizing the concentration of the reactive material and simply tuning the refractive index difference between it and the matrix. This approach allows us flexibility in tailoring the media to the particular needs of high density holographic data storage.

In Figure 3(a), we show the M/#'s of a series of 200 μm thick media that were fabricated using the same matrix but writing monomers of varying refractive index and varying size where the concentrations of the monomers were adjusted to yield equivalent levels of recording-induced changes in dimension and bulk refractive index. (Each of the samples underwent changes equivalent to ~0.35% in thickness and ~2.1x10^{-3} in the bulk refractive index for the complete reaction of the photoactive monomers.) Increases in M/# from 2 to 11 were realized while maintaining the same level of effective dimensional stability of the media.

Media with high M/# were obtained by fabricating thick samples of our photopolymer materials. In Figure 3(b), we show how the M/# scales with thickness in media fabricated with a typical writing monomer. Data from three sets of samples are shown, with each set formulated with a different concentration of the monomer and therefore exhibiting different levels of effective dimensional stability. The gains in M/# with increasing thickness are possible because of the low levels of light absorption and light scatter and the high level of effective dimensional stability of the media.

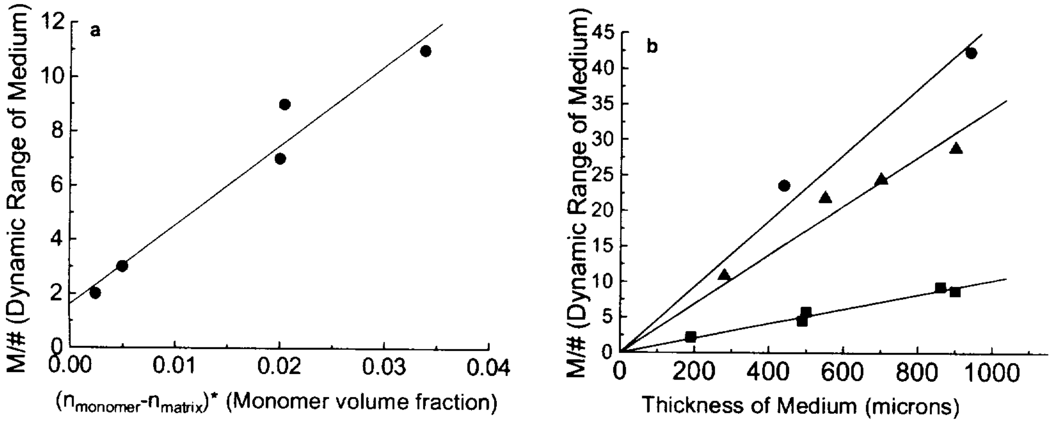


Figure 3(a) M/# versus the product of the difference between refractive indices of the photosensitive monomer and the matrix and the volume fraction of the monomer for five different resins. The molar number of reactive groups of the photoactive monomer in each of the media is adjusted to yield equivalent amounts of recording-induced rotations in the Bragg angle upon recovery. The media are all 200 μm thick. **(b)** M/# versus thickness for photopolymer media fabricated with a typical writing monomer: closed squares (triangles, circles), media exhibit ~0.1% (0.35%, 0.5%) change in thickness upon recording.

The optical quality of the photopolymer media is demonstrated by the "straight-through" image shown in Figure 4. Here, an 800x600 pixel chrome on glass amplitude mask is imaged through the media onto a charge-coupled device detector as shown in Figure 4(a). The transmitted image and image statistics are shown in Figure 4(b).

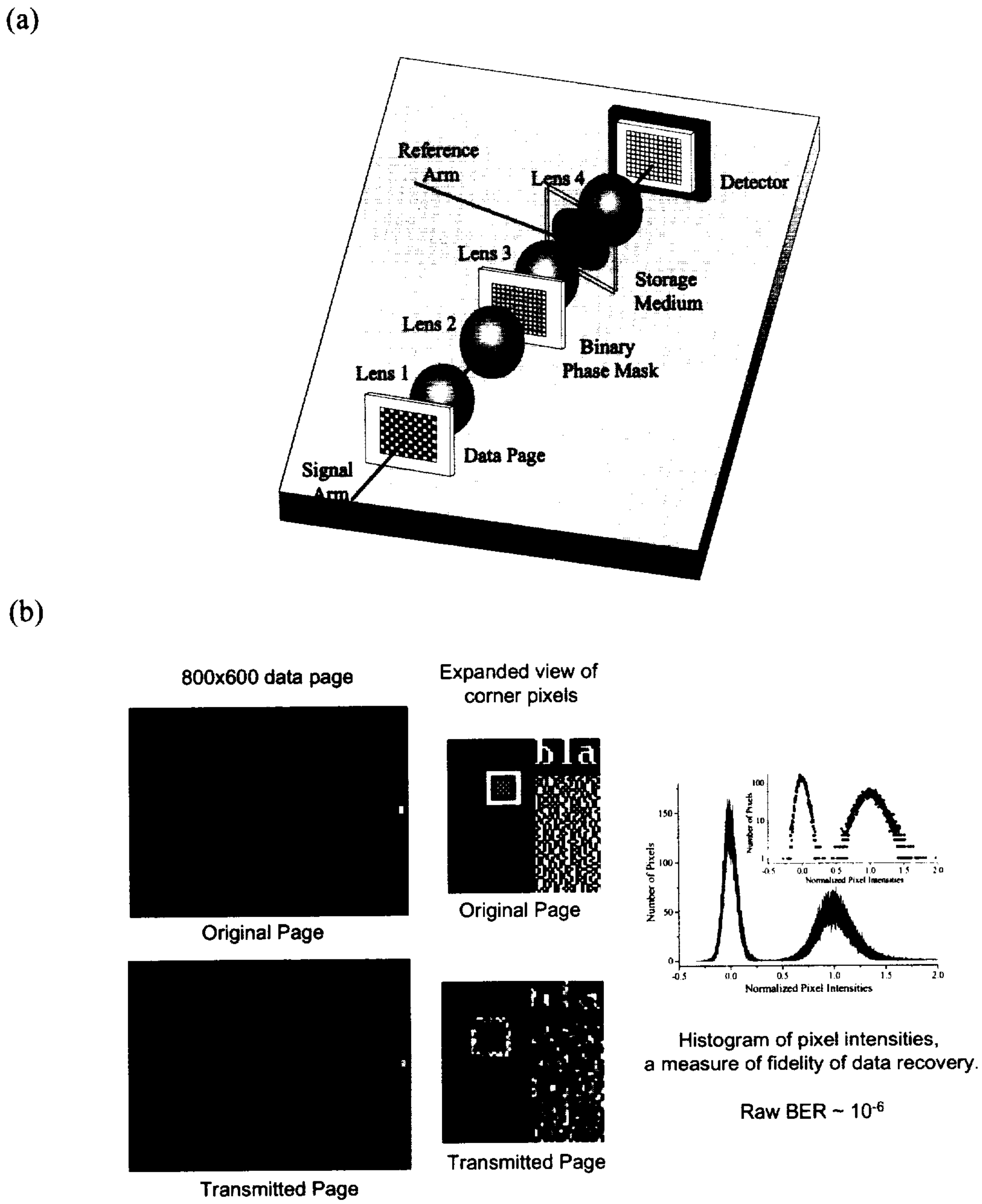


Figure 4(a) Optical system used for digital holographic data storage. A spatial light modulator (the data page), encoded with an array transmitting and opaque pixels, is illuminated with a plane wave. The data page is imaged through lenses 1 and 2 onto a random binary phase mask which serves to randomize the information of the data page at the recording plane. The data page is then Fourier transformed through the storage medium at the recording plane by lens 3 and imaged through lens 4 onto a charge-coupled device detector. For high fidelity data recovery, each pixel of the data page must be precisely mapped through the optical train of the signal arm onto a corresponding pixel of the detector array requiring the storage medium to be of high optical quality. The reference arm is spatially overlapped with the signal arm at the storage medium during the recording process and is used alone to reconstruct the data page during the recovery process. **(b)** Straight-through image of a 480 kbit data page. The intensities were digitized using a Princeton Instruments ST138 CCD camera with 16 bit resolution. The calculated raw bit error rate calculated from a histogram of the pixel intensities is 2.6 x 10^{-6}. (Before calculating the histogram, the intensities of the pixels were normalized so that the local averages of the on and off bits equaled the global averages.) The inset shows the data plotted on a logarithmic scale.

The materials described here represent substantial advances in the development of recording media for holographic data storage. Ongoing work is focused on the temperature sensitivity of the polymer media and the long-term archival life of stored data are also under investigation.

3. New Multiplexing Methods

The methods used to overlap or multiplex holograms determine the complexity and architecture of the recording system. Two recently developed multiplexing methods have led to system designs in which accessing different holograms requires only motion of the media.

Both approaches used a fixed set of optics to create the reference beam. In shift multiplexing, the reference beam consists of a collection of plane waves or a spherical wave [3]. Holograms are multiplexed through spatial translations on the order of microns of the media relative to the reference beam. Large numbers of holograms can therefore be overlapped in essentially the same volume of the media. In correlation multiplexing, a complex reference beam encodes the position of the hologram in the recording medium and again micron-scale translations allow many holograms to be multiplexed [4]. These "fixed optics" methods enable a holographic storage system based on a spinning disk architecture used throughout much of the storage industry. A schematic of a holographic drive based on these fixed optics methods is shown in Figure 5.

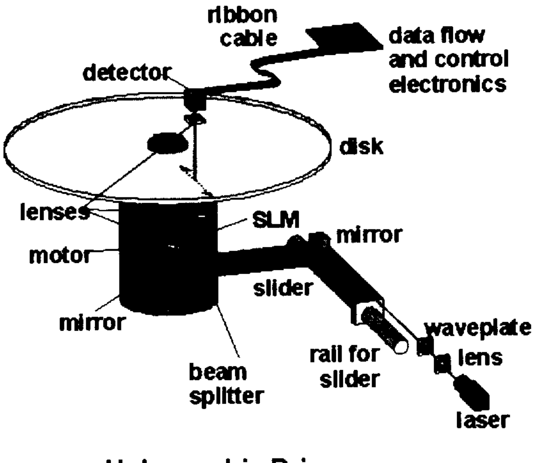


Figure 5. Schematic of a rotating holographic drive based on multiplexing methods that use fixed optics.

4. Demonstrations of High Storage Densities

In early experiments [5], we demonstrated the feasibility of digital storage in thick photopolymer systems by recording and recovering with low bit error rates multiple high capacity (480kbit) digital data pages in media up to 500 μm thick. The results established that (i) polymer media could be fabricated with the high optical quality and low level of light scatter required for high density data storage applications, and (ii) optical components and the media could be integrated to yield a high performance storage system.

More recently, storage densities as high as 31.5 channel Gbits/in^2 have been attained by recording and retrieving >3000 data pages (each of capacity 480 kbit) using shift multiplexing in a 750 μm thick photopolymer medium [6]. Each of the 3000 data pages were fully recovered with signal-to-noise levels above that required for error-free readout. These experiments provide further evidence of the storage capabilities of the photopolymer media. In addition, the data indicate that even greater storage densities are possible with in-house, higher response versions of the materials used here.

5. Prototype Holographic Recording System

The development of practical components for holographic systems has been accomplished largely in fields outside the storage industry. For example, the frequency-doubled, diode-pumped Nd:YAG laser, used in medical, cable TV, and printing industries, is an attractive recording source due to its small size, ruggedness, and low cost. Digital micro-mirror devices appearing in display applications are ideal spatial light modulators with their large numbers of pixels (~ 1 million), fast frame rates (2000 Hz), and high optical contrast. The CMOS active pixel detector arrays emerging in digital photography exhibit the rapid access and data transfer properties required for holography. The volume of these non-storage markets is expected to lead to low-cost, reliable components.

A prototype digital storage system assembled from the components described above and readily available optics is shown in the accompanying photograph in Figure 6. The system occupies an approximately 1x2 foot area and can be considerably reduced in size with the use of custom-designed optics.

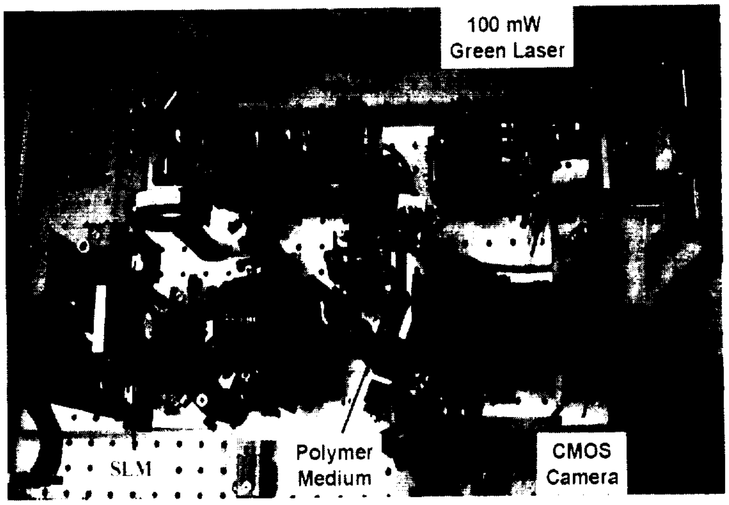


Figure 6. Prototype holographic storage system.

6. Outlook for Holographic Data Storage

The recent commercial availability of optical components such as spatial light modulators, CMOS detectors, and compact visible wavelength lasers has removed many of the obstacles that previously prevented the practical consideration of holographic data storage. We believe the advances in media, recording methods, and the demonstrated densities of >30 channel Gbits/in^2 described here further enhance the prospects for holography to become a realizable, next-generation storage technology.

References

[1] L. Dhar, A. Hale, H. E. Katz, M. L. Schilling, M. G. Schnoes, and F. C. Schilling, “Recording media that exhibit high dynamic range for digital holographic data storage”, *Opt. Lett.* **24** (1999) 487.

[2] F. H. Mok, G. W. Burr, and D. Psaltis, “System metric for holographic memory systems”, *Opt. Lett.* **21** (1996) 896 .

[3] D. Psaltis, M. Levene, A. Pu, G. Barbastathis, and K Curtis, “Holographic storage using shift multiplexing”, *Opt. Lett.* **20** (1995) 782.

[4] K. Curtis, presented at OSA Annual Meeting, Long Beach, CA (1997).

[5] L. Dhar, K. Curtis, M. Tackitt, M. Schilling, S. Campbell, W. Wilson, A. Hill, C. Boyd, N. Levinos, and A. Harris, “Holographic Storage of Multiple High Capacity Digital Data Pages in Thick Photopolymer Systems”, *Opt. Lett.* **23,** (1998) 1710.

[6] K. Curtis, W. Wilson, M. Tackitt, A. Hill, and T. Richardson, presented at International Conference on Photonics, Taipei, Taiwan (1998).

Index of Authors

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*

2. REPORT DATE
March 2000

3. REPORT TYPE AND DATES COVERED
Conference Publication

4. TITLE AND SUBTITLE
Eighth Goddard Conference on Mass Storage Systems and Technologies in Cooperation with the Seventeenth IEEE Symposium on Mass Storage Systems

5. FUNDING NUMBERS
423

6. AUTHOR(S)
B. Kobler and P C Hariharan, Eds.

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS (ES)
Goddard Space Flight Center
Greenbelt, Maryland 20771

8. PEFORMING ORGANIZATION REPORT NUMBER
2000-01419-0

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS (ES)
National Aeronautics and Space Administration
Washington, DC 20546-0001

10. SPONSORING / MONITORING AGENCY REPORT NUMBER
NASA/CP—2000–209888

11. SUPPLEMENTARY NOTES
P C Hariharan, Systems Engineering and Security, Inc., Greenbelt, Maryland

12a. DISTRIBUTION / AVAILABILITY STATEMENT
Unclassified–Unlimited
Subject Category: 82
Report available from the NASA Center for AeroSpace Information,
7121 Standard Drive, Hanover, MD 21076-1320. (301) 621-0390.

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

This document contains copies of those technical papers received in time for publication prior to the Eighth Goddard Conference on Mass Storage Systems and Technologies which is being held in cooperation with the Seventeenth IEEE Symposium on Mass Storage Systems at the University of Maryland University College Inn and Conference Center March 27–30, 2000. As one of an ongoing series, this Conference continues to provide a forum for discussion of issues relevant to the management of large volumes of data. The Conference encourages all interested organizations to discuss long term mass storage requirements and experiences in fielding solutions. Emphasis is on current and future practical solutions addressing issues in data management, storage systems and media, data acquisition, long term retention of data, and data distribution. This year's discussion topics include architecture, future of current technology, new technology with a special emphasis on holographic storage, performance, standards, site reports, vendor solutions. Tutorials will be available on stability of optical media, disk subsystem performance evaluation, I/O and storage tuning, functionality and performance evaluation of file systems for storage area networks.

14. SUBJECT TERMS Magnetic tape, magnetic disk, optical data storage, mass storage, archive storage, file storage management system, hierarchical storage management software, data backup, network attached storage, archive performance, media life expectancy, archive scalability, tertiary storage, data warehousing, holographic storage.

15. NUMBER OF PAGES
471

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	**18. SECURITY CLASSIFICATION OF THIS PAGE**	**19. SECURITY CLASSIFICATION OF ABSTRACT**	**20. LIMITATION OF ABSTRACT**
Unclassified	Unclassified	Unclassified	UL