GSFC/CP-1998-206850

Sixth Goddard Conference on Mass Storage Systems and Technologies

in cooperation with the

Fifteenth IEEE Symposium on Mass Storage Systems

Benjamin Kobler and P C Hariharan, Editors

Proceedings of a conference held at
The Inn and Conference Center
University of Maryland, University College
College Park, Maryland
March 23–26, 1998

National Aeronautics and
Space Administration

Goddard Space Flight Center
Greenbelt, Maryland 20771

March 1998

GSFC/CP-1998-206850

Sixth Goddard Conference on Mass Storage Systems and Technologies

in cooperation with the

Fifteenth IEEE Symposium on Mass Storage Systems

Edited by
Benjamin Kobler
NASA-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
March 23– 26, 1998

National Aeronautics and
Space Administration

Goddard Space Flight Center
Greenbelt, Maryland 20771

March 1998

Sixth Goddard Conference on Mass Storage Systems and Technologies

in cooperation with the

Fifteenth IEEE Symposium on Mass Storage Systems

Program Committee

Available from:

NASA Center for AeroSpace Information
800 Elkridge Landing Road
Linthicum Heights, MD 21090-2934
Price Code: A17

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

Preface

The Sixth NASA Goddard Space Flight Center Conference on Mass Storage Systems and Technologies is being held in cooperation with the Fifteenth IEEE Symposium on Mass Storage Systems.

The Conference begins with a day of tutorials, and is followed by three days of papers on Architecture, Tape Optimization, New Technology, Performance, Standards, Site Reports and Vendor Solutions, as well as an evening of poster paper presentations, and an evening banquet and after dinner talk. Because scalability is an important issue, a panel will debate the ability of currently available operating systems to handle increasing data volumes, and what the future holds in this regard. Time has also been set aside for an hour of extemporaneous talks, for attendees who feel that they have a statement to make or ideas to share with the audience.

Most of the papers presented during the Conference are published in these proceedings. These include papers that describe tutorial presentations on shared file systems and Fibre Channel, backup techniques, and the dynamics of obsolescence. Other papers published touch on the various aspects of data acquisition, storage, recording and management. Fibre Channel is helping to bring Network-attached storage (NAS) to the desktop. The inadequacies of traditional file systems vis-a-vis NAS are explored in a number of papers, and various solutions are proposed. Since the early 90's, areal densities on magnetic media have been growing at an annual rate of about 63%. Solid immersion lenses (SIL) promise an order-of-magnitude increase by the end of this year; RAID systems with multi-terabyte capacity in compact form factors should be just around the corner. Concern over loss of data from storage media has motivated the desire to monitor media health and performance, and AIIM Committee C21.3 is currently involved in drafting a standard for this.

As in the past, a CD-ROM that contains papers, viewgraphs, and audio/video of presentations, panel discussions, and the after dinner talk, will published and made available to registered attendees after the conference.

The preparation for this conference and this publication involved dedication and teamwork by the Conference Program Committee who reviewed submissions and worked with authors to improve papers and to ensure that they were complete. We are especially grateful to the following committee members who participated in most, if not all, of the numerous program committee meetings, and whose hard work made all this possible.

Jean-Jacques Bedet, *Raytheon STX*
John Berbert, *NASA Goddard Space Flight Center*
Bill Callicott, *Consultant*
Sam Coleman, *Lawrence Livermore National Laboratory*
Fynnette Eaton, *Office of Smithsonian Institution Archives*
Ethan Miller, *University of Maryland, Baltimore County*
Jim Ohler, *Department of Defense*
Bruce Rosen, *National Institute of Standards and Technology*
Don Sawyer, *NASA Goddard Space Flight Center*
Rodney Van Meter, *Quantum Corporation*

Special thanks also go out to Rodney Van Meter and Ethan Miller for helping with last minute edits of Postscript, LaTeX and FrameMaker documents.

Ben Kobler
P C Hariharan

Table of Contents

Shared File Systems and Fibre Channel

Matthew T. O'Keefe
Department of Electrical and Computer Engineering
and
Laboratory for Computational Science and Engineering
University of Minnesota
Minneapolis, MN 55455
okeefe@ece.umn.edu
+1-612-625-6306

Abstract: Shared file systems like Cray's SFS, DEC's VAXcluster file system, and Oracle's Parallel Server exploit network-attached storage by creating serverless distributed file systems that allow efficient, simultaneous access to shared, network-attached storage devices. In the past, these shared file system designs relied on proprietary network-attached storage like DEC's CI network or higher-cost interfaces like HiPPI, or used software to emulate shared storage networks. With the advent of Fibre Channel, a high-volume, open standard in network-attached storage interfaces is now available. In this paper we will review past work in traditional distributed file systems like NFS and AFS and work in shared file systems by Cray, DEC, CMU and others. New file system architectures for shared network storage will also be described[1].

We will give a brief overview of Fibre Channel technology, describing its protocol layers, how higher level protocols like IP and SCSI are mapped to this protocol, and the topologies supported. Finally we will sketch the future evolution of network-attached storage as it moves from high-end, high-capacity requirements into mid-range and lower-end desktop computing. We believe these trends may be affected by Fibre Channel's potential to function as both a network and storage interface.

1. Introduction

This paper reviews traditional client-server distributed file system designs and motivates shared disk file systems by describing Fibre Channel, a widely-used shared storage interconnect that will encourage the development of shared file systems. We define shared file systems and describe past and current work in this important research area. Finally, we sketch possible trends that may develop in the future as Fibre Channel and shared storage systems evolve.

2. A Short History of Distributed File Systems

In a traditional client-server distributed file system the server typically provides a name space, enforces file access permissions, maps names and file offsets to disk block addresses, and performs directory lookups. Clients send file system commands such as `create`, `read`, and `write` across a network to be executed on the server.

There are several advantages to the client-server approach for distributed file system design. Clients are to use access files transparently across a network using the same commands employed by the local file system, preserving binary compatibility. Network hardware and protocol independence are achieved by using standard network protocol stacks which include protocols such as RPC, XDR, and TCP/IP, as show in figure 1.

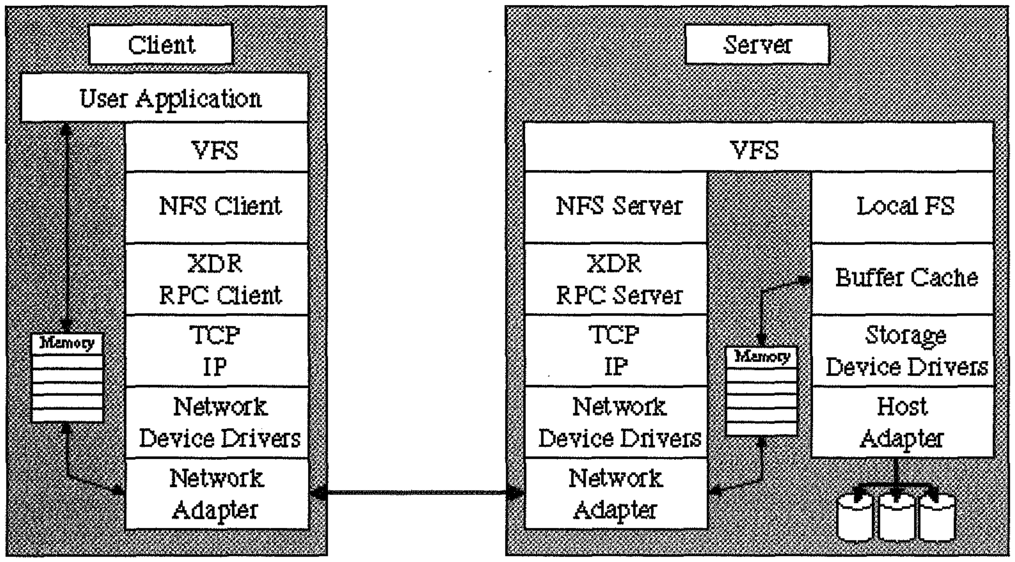


Figure 1: NFS Protocol Stack on Client and Server.

Network protocol independence helps insure portability by separating the network hardware from the distributed file system software. As we will show, this has implications for both performance and system complexity.

Important design criteria for distributed file systems include [Vah96]:

1. Name Space — how is the file system name space constructed on each client and does each file have the same pathname across all clients?
2. Statefull or stateless operations — do the clients and the server maintain state about previous operations and which files are open on which clients?
3. Semantics of sharing — when multiple clients share a file concurrently do they see file modifications on other clients immediately (UNIX sharing semantics) or is there a delay?
4. Server Callbacks — when a cached file is modified does the client discover this via polling the server or does the server *callback* other clients to inform them that the file has changed?

The original client-server distributed file system designs [SaG85], [Tuc96], [Vah96] relied on a single central server to service clients requests which simplified the design but also limited scalability and availability by introducing a single point-of-failure. To achieve better scalability distributed file systems have evolved in the following directions:

1. *Distributed servers* to balance the file request workload and provide redundancy [Sat90], [Ous88].
2. *Looser sharing semantics and client-based caching* to reduce client demands on the server [Sat90].
3. *Migrating functionality* from the server to the clients [Sat90], [AnD95].

Some examples of distributed file systems include NFS [SaG85], [Tuc96], RFS [Vah96], Sprite [Ous88], Coda and AFS [Sat90], and xFS [AnD95]. Here we summarize three distributed file system designs: NFS, RFS, and AFS.

2.1 Network File System (NFS)

The Network File System was designed by Sun Microsystems in the mid-80s to allow transparent file sharing among multiple UNIX clients [SaG95], [Vah96], [Woo95], [Koe95]. Sun also developed the *vnode/vfs* installable file system interface to allow a single UNIX kernel to support multiple, non-native file systems by providing an interface between file-system independent and file-system dependent portions of the kernel. This technique, though not the *vnode/vfs* interface itself, has become standard today as most modern operating systems provide an interface for installable file systems to be added to the kernel.

The original NFS architecture has been described as stateless but improving performance, failure recovery, and consistency with UNIX file system semantics has meant that most NFS implementations include state in actual implementations, especially on clients. NFS goals included operating-system-independent implementation, hardware and transport independence, and simple failure recovery. Sun aggressively licensed NFS and provided a reference implementation that allowed many other vendors to implement it; today it is the de facto standard for distributed file systems in UNIX and other heterogeneous environments.

Figure 1 shows the basic NFS protocol design: clients requests are routed through the client *vfs* layer onto a network via RPC, XDR, and TCP/IP protocols to the server *vfs* layer. The server translates these client requests into local file system requests; `read` request data is packaged and returned to the client whereas `write` data is synchronously written to disk (as is any metadata modified as a result of the `write` request). This means that the client application cannot proceed until the NFS server has written the data and metadata to the disk media. This limitation is a direct result of the state-less nature of the NFS protocol and results in other performance-inhibiting NFS “features”:

- when the NFS server crashes clients continue to send request packets, burdening the network with unnecessary load
- when clients interpret a heavily-loaded NFS server as having crashed due to long response latency they continue to send request packets to this server, loading it even further

2.2 The Remote File Sharing system (RFS)

RFS was developed by AT&T and introduced in SVR3 UNIX to provide remote access to files over a network. Unlike NFS, all UNIX semantics are preserved. The implementation is state-full to improve performance, provide UNIX file sharing semantics, and simplify implementation. RFS has never reached the popularity of NFS though in many ways it is a cleaner distributed file system design.

RFS uses a central name server to provide resource names for exported directories. In addition, its structure is hierarchical: a single RFS system may be composed of multiple domains each of which is an independent administrative structure. However, files may be accessed between domains relatively transparently. The RFS server maintains state about clients including which files have been opened by a client, incrementing the reference counts on their vnodes. In addition, file/record locks and readers/writers counts for named pipes are maintained as well as a table of all clients that have mounted file systems [Vah96].

Crash recovery in RFS is relatively straightforward. A virtual connection is set up when a mount operation is first performed between a client and server. This virtual connection is broken when either the client or server fails. When a client crashes, the server decrements the inode reference counts for files opened by that client and releases any file or record locks held by that client. When a server crashes, client processes waiting for server requests to complete are informed an error has occurred by the return of the `ENOLINK` error message. RFS inodes referring to files on the failed server are flagged so that later operations on these files return an error condition and a user-level process is started to clean up state related to this server.

Since RFS maintains state information on the server about which files each client has open, it is possible for the server to inform clients who have cached a given file that it has been modified and the current cached versions of this file are therefore invalid. This mechanism was also used by the Sprite operating system [Ous88] and the state-full NFS implementation Sprite inspired [Sri89]. Client caching is write-through so that all client writes must be immediately sent to the server which then informs clients to invalidate their cached versions of this file, providing strong consistency. A drawback of this approach is that if a client crashes or becomes unreachable across the network then it may take a long time for it to respond to cache invalidation requests which prevents other client operations from completing [Vah96].

2.3 The Andrew File System

AFS [Sat90] originated in 1983 as a joint project between Carnegie Mellon University and IBM to develop a campus-wide data sharing infrastructure that exploited desktop workstation technology but also provided the efficient data sharing capabilities of centralized machines. AFS evolved through three implementations — AFS-1, AFS-2, and AFS-3 — before the development was spun off to Transarc Corporation.

AFS shares some of the features of central server distributed file systems like NFS and RFS but was designed to scale to dozens of servers and thousands of clients potentially sharing data across a wide area network. Data is stored centrally in the collection of trusted AFS servers called "Vice" which are surrounded by untrusted AFS clients embedded in a surrounding network connected to Vice. A client workstations can access any file in the Andrew name space using the same name providing location transparency. (NFS file names may be different on each client depending on where the exported directory is mounted on each client.)

Like RFS, AFS-2 and AFS-3 rely on server *callbacks* when files cached in clients have been modified. AFS-1 and AFS-2 cached whole files, while AFS-3 caches 64-Kbyte chunks. AFS-2 uses session semantics: once a file is opened by a client, it reads and writes that file in isolation from other clients. File modifications are written to the server only when the file is closed. Clients cache entire directories and perform name lookups directly without accessing the server.

An excellent guide by Levy and Silberschatz to previous work in the area of distributed file systems can be found in [LeS90]. The Transarc DFS is a highly sophisticated and robust descendant of AFS-3 (see `www.transarc.com`). Fault-tolerance issues in distributed file systems are discussed in [KiS96]. Performance measurements for several distributed file systems are described in [Sat90] and [Bak91]. A classification scheme in the context of network-attached storage can be found in [SoE97a].

3. The Advent of Network Attached Storage: Fibre Channel Arrives

Recent advances in switching technology, fiber optics and the convergence of network and channel interfaces [SaL94], are allowing order-of-magnitude improvements in network latency and bandwidth through new technologies like Fibre Channel [Ben95]. Open standards and high-volume markets, combined with the constant increase in functionality and decrease in cost for microelectronic devices, continue to drive down network costs. The previous speed imbalance between disk drives and networks will be reversed: parallel drive designs will be needed to exploit switched network bandwidth and meet the requirements of tomorrow's demanding applications.

The Fibre Channel standard integrates both storage and networking capabilities into a single serial interface that currently has a speed of 100 Megabytes/second (and a growth path to 400 Megabytes/second), allows both low-cost loop connections (much like FDDI rings) with up to 126 devices at distances beyond 100s of meters and is scalable to 100s or 1000s of devices with Fibre Channel switches. Yet Fibre Channel is beginning to achieve widespread use with disk drives and adapters priced about the same as parallel SCSI technology [Dem94]. In contrast, today's parallel SCSI technology supports only about 8 devices per bus with each bus extending at most 25 meters making the technology effectively unscalable.

In this section we give an overview of the Fibre Channel standard which we break into four sections: interconnect topologies, physical characteristics, protocol layers, and industry support.

3.1 Interconnect Topologies

There are three basic topologies supported in Fibre Channel:

1. Point-to-point
2. Fabric (switch)
3. Arbitrated loop (ring)

Fibre Channel networks can be set up as a single point-to-point link between two "N-Ports", as a group of N-ports connected together through "F-ports" on a Fabric (switch), and by a group of "NL_ports" connected together on an arbitrated loop (ring) without the need of a switch. Each N_port resides on a Fibre Channel "node" which is typically either a computer, disk array, or disk drive. These three topologies allow system architects to use only as much bandwidth and interconnect capability as required. Hence, a single disk attached to a single computer uses a point-to-point connection, a group of disk drives attached to a single computer would likely use an arbitrated loop topology to reduce the port cost per disk drive, and a cluster of large, fast servers might share several fast disk arrays across a Fabric switch. These topologies and their interactions are all formally defined in the Fibre Channel standard (see `http://www.fibrechannel.com`).

3.2 Physical Characteristics

Though the physical Fibre Channel interface was originally designed to support single- and multi-mode optical connections, the standard has been broadened to include support for copper coax and twisted pair lines run over shorter distances. Single-mode fiber optic connections can be run up to 10 kilometers; cheaper multi-mode connections can be run up to 1 km. Each port includes both a transmitter and receiver: in point-to-point and Fabric topologies the remote connections for the transmitter and receiver end at the same port, while in the arbitrated loop topology these connections end up at different ports. Full-speed connections run at 1,062.5 Mbps (Megabits per second) which, after protocol overheads are factored in, gives a potential data transfer rate of 100 MBps (megabytes per second). The standard includes a well-defined path to "double-" and "quadruple-speed" which can achieve data transfer rates of 200 MBps and 400 MBps, respectively.

Fibre Channel provides scalability in both bandwidth and ports. Arbitrated loop topologies allow up to 126 NL_ports, far exceeding the 8 to 16 devices possible with parallel SCSI. Current Fabric switches contain 8 to 16 ports but can be cascaded to create large Fabrics with over 32 switches and 512-port Fabrics.

3.3 Protocol Layers

Fibre Channel functionality is implemented in 6 layers [Ben95]:

1.	FC-0:	Physical Interface and Media
2.	FC-1:	Transmission Protocol and Byte Encoding
3.	FC-AL:	Arbitrated Loop Functions
4.	FC-2:	Signaling Protocol and Link Service
5.	FC-3:	Common Services over Multiple N_ports
6.	FC-4:	Upper Level Protocol Mapping

FC-0 defines the optical and electronic cable plant, connectors, and transmitters and receivers. FC-1 describes the 8B/10B encoding used for byte and word alignment, ordered sets used for frame bounds, low-level flow control, and link management, port operational states and error monitoring. FC-2 defines the frames, sequences and exchanges used to transfer data and control information, buffer-to-buffer and end-to-end flow control, and Fabric and N_port login and logout. FC-3 includes common services implemented over multiple N_ports such as striping while FC-4 codifies how upper level protocols like IP and SCSI [Dem94] are mapped onto Fibre Channel.

New products are being introduced at an increasing rate and the Fibre Channel standard continues to evolve. The following Web sites provide current information on standards and products:

- `http://www.fcloop.org`
- `http://www.fibrechannel.com`

Van Meter provides an excellent overview of other interfaces (but including Fibre Channel) that allow direct attachment of devices to networks [Met96].

4. Shared File Systems

The client-server architecture for current distributed file systems described in section 2 was driven partly by the inability to attach storage devices directly to a network and by requiring that the file system be independent of the network transport medium employed. However, as new technologies like Fibre Channel that support network-attached devices become increasingly ubiquitous, it becomes reasonable to consider distributed file system architectures which exploit this network-attached hardware without deep protocol stacks like that found in NFS (see figure 1). These *shared file systems* are similar to local file systems in several ways but also must address new issues in synchronization, scalability and error recovery.

In this section we describe the key characteristics of shared file systems and give examples of this file system architecture. Shared file systems we are aware of include:

- IBM Sysplex [Pfi95]
- Oracle Parallel Database Server [Llo92]
- LLNL's High Performance File System [Wat95]
- DEC Vaxcluster file system [KrL86], [DEC87] (1984)
- High Performance File Server [ArB93] (1993)
- Cray's Shared File System [Mat95] (1994)
- IBM's Parallel Journaled File System [DeM95] (1995)
- NASD (Network Attached Secure Disk) File Systems [Gib96], [Gib97](1995)
- Global File System [SoE97a], [SoE97b], [SoR96] (1995)
- Veritas' Cluster File System (CFS) (1998)

Our definition of a shared file system is simple and therefore fairly broad: a *shared file system* allows direct data transfers between computers (clients) transferring data and the storage device that contains the data such that more than one client may access data from the same storage device. Hence, the device is *shared* between clients. This definition implies an interconnection network between multiple clients and the storage device. In addition, the file system software must recognize the existence of other clients accessing the same storage devices and file system data and metadata. This requirement precludes most *local file systems* from being considered as shared file systems since local file systems generally consider the storage devices as being owned and accessed by a single host computer.

Shared file systems provide a server-less alternative to traditional distributed file systems where the server is the focus of all data sharing. A shared file system approach based upon a shared network between storage devices and clients (often referred to as a *Storage Area Network or SAN*) offers several advantages:

1. *Availability* is increased since if a single client fails another client can continue processing its workload because it can continue to access the failed clients data from the shared disk.
2. *Load-balancing* a mixed workload among multiple clients sharing disks is simplified by the clients ability to quickly access any portion of the dataset on any of the disks.
3. *Pooling* of storage devices into a shared disk memory equally accessible to all clients in the system is possible.
4. *Scalability* in capacity, connectivity and bandwidth can be achieved without the limitations inherent in file systems designed with central servers.

Shared file systems can be classified using the following characteristics:

- Symmetric or asymmetric?
- Locking performed on clients or devices?
- Proprietary or open storage networking interface?
- Developed by modifying a local file system or writing new code?

A shared file system is *symmetric* if any client can perform any file system operation without interacting with another client. In *asymmetric* shared file systems a client must first make a request through a *file manager* executing on another client. The file manager typically manages the file system metadata, checks file access permissions, and provides the client with information necessary to access the data directly on disk via the storage area network. Asymmetric shared file systems are sometimes said to use *third-party transfers* because the file manager acts as a third-party in the transfer between the client and storage device. Once approved the data transfer between client and device is direct.

Asymmetric file systems have several advantages. The file manager can be designed by modifying the server in client-server distributed file system so that the control and data operations are separated [ArB93], [Wat95], [Gib97]. Since control transfers are much smaller than large data transfers a separate network for control packets can emphasize low latency and avoid interference with larger (and hence typically longer) data transfers. The predominance of packet-switching networks makes the latter advantage less important today. In addition, asymmetric designs do have several significant disadvantages, including:

- the file manager is a bottleneck and single point-of-failure
- both client and file manager code must be written and
- centralized locking and logging on the file manager limit scalability.

Since shared file systems allow multiple clients to access shared storage devices simultaneously a *locking mechanism* is necessary to insure mutual exclusion as file system metadata is modified. For example, if a write operation increases a file's size then additional file system blocks must be assigned to that file, changing the file system block allocation maps and the file's dinode (disk inode) structure so that it includes these additional blocks. The operations on both these data structures must be *atomic* to insure they are completed properly.

Locking in shared file systems is performed either in the clients or in the devices. In either case this locking may be either centralized or distributed. Asymmetric shared file systems exemplify centralized, client-based locking: all metadata locking is performed on the file manager [Wat95], [Gib96], [Gib97]. In contrast, shared file systems like the Vaxcluster [KrL86] and Oracle Parallel Server [Llo92] use a distributed, client-based locking scheme called the *distributed lock manager.* A distributed lock manager is not as vulnerable to file manager failure and can balance the lock manipulation overhead among many clients. However, distributed lock manager design in the context of potential client failures is notoriously difficult and may inhibit scalability to large numbers of clients. A key advantage to client-based locking, at least at the current time, is portability: all that is required is a network protocol that allows clients to communicate. Presently, there is no standard, widely-used fine-grained locking technique available on devices [2].

Locking in either storage or network devices usually can yield a simpler and faster lock protocol than client-based approaches [Pfi95], [Mat95], [SoE97a]. The lock mechanism can be placed in a centralized dedicated piece of hardware as in the IBM Sysplex Coupling Facility [Pfi95] and the Cray SFS HSMP [Mat95], in the network switch or hub, or on the storage device itself as in the University of Minnesota's Global File System [SoEb97]. These locks can be in one central location or spread among the devices or other network components. Since devices are less likely to fail than clients and can rely on techniques like RAID to ensure availability, lock protocol design is simplified. There must exist a pool of fine-grain locks that are fast and preferably distributed among the devices or network components to balance lock manipulation load.

IBM Sysplex, Cray SFS, and DEC Vaxcluster shared file systems used ESCON (Enterprise System Connection), HiPPI, and CI (Cluster Interconnect), respectively, as the underlying shared storage interconnect. These are proprietary or low-volume, high-end interfaces that are generally specific to a single vendor. In contrast, Fibre Channel is an open, industry-wide standard supported by every major workstation, disk, disk array, and component vendor and prominent OEMs like Compaq, HP, Sun, SGI, and Microsoft. Thus, Fibre Channel will provide the underlying shared storage interconnect for Storage Area Networks of the future enabling cross-platform shared file systems.

An important question when designing a shared file system is whether to re-use existing distributed or local file system code or to write a new file system. Most previous efforts. in shared file system design, including Cray's SFS [Mat95] and IBM's PJFS [DeM95], modified existing local file systems to work in a shared storage environment. Local file systems emphasize caching as much file system data and metadata as possible which improves performance when locality and data re-use exist in file accesses. However, this emphasis on caching complicates file sharing between clients when local file systems are modified to act as shared file systems [Mat95]. The Global File System [SoE97b] was designed to be a scalable, symmetric, shared disk file system that exploits shared storage devices on a network. The on-disk data structures and locks are partitioned to balance load between devices thereby enhancing scalability. Caching is used sparingly where necessary but is not allowed to interfere with fine-grained data sharing between clients.

We now give brief descriptions of four shared file system designs [3].

4.1 DEC Vaxcluster file system (1984)

DEC developed the Vaxcluster architecture to provide a highly available system that provided users with a single system image across a cluster of Vax workstations. The original implementations used the CI (Computer Interconnect), a custom 70 Mbps network that interconnects both computers and disk controllers. The Vaxcluster provides an elegant, symmetric shared file system for Vaxcluster nodes where locking is handled through the *distributed lock manager* [KrL86], [DEC87] (DLM). This client-based lock manager provides a generalized lock service for all resources in the Vaxcluster, including devices, print services, files, and any other resource the user or operating system might care to define.

The lock manager allows clients to request and release a lock. Each request specifies a locking mode which provides for varying levels of exclusive control on the associated resource, from exclusive access (no other host may read or write the resource), to concurrent read access where other clients may read or write to the shared resource. Lock requests may be queued so that once the resource becomes available the requesting client is so informed. The distributed lock manager is distributed across the clients in a Vaxcluster and locks are cached on the requesting client if possible. This distributes the lock manager load to all machines in the cluster, which aids scalability. It is designed to work in the presence of client failures.

The original VMS file system was modified to use the DLM to support shared, simultaneous access to files and associated file system metadata [DEC87]. This required that locks be associated with directories, file, volumes. Extensive caching was used to speed operations in the original file manager: this caching was preserved in the Vaxcluster implementation by exploiting version numbers associated with lock operations. Stale cache data could be detected as a disparity in version numbers caused by earlier file update operations [DEC87] that had occurred on other clients.

4.2 Cray's Shared File System (1994)

Cray developed a shared file system called SFS (Shared File System) for the UNICOS operating system [Mat95]. Originally developed as a custom implementation for a customer who required highly available, shared access to disks shared by multiple C90-class vector mainframes, SFS later became a supported product. Matthews [Mat95] describes how the implementation evolved over time but the basic architecture was the same for all implementations: a symmetric design with device-based locks. The UNICOS file system was modified to support parallel access to shared files on shared disks. The key improvement across the implementation phases was a reduction in either lock overhead or the number of lock operations.

Cray developed a "semaphore" operation to be executed at the device and developed a mapping of semaphores-to-metadata that allowed mutual exclusion for the operations on this metadata. The semaphore operation is actually a *test-and-set* primitive and is much simpler than the locks used in the Vaxcluster's distributed lock manager. Cray achieved good performance on large transfers for a single client [Mat95] but no multiple-client performance data is reported in this paper.

4.3 NASD (Network Attached Secure Disk) File Systems (1995)

Gibson [Gib96], [Gib97] has proposed *Network Attached Secure Disks* as a standard for shared storage devices[4]. NASD goes beyond previous shared disk storage systems in two key areas: security and objects. NASD-based file systems as currently proposed use a file manager for directory and certain other operations, but provide mechanisms to keep these overheads low. However, it is quite possible that NASD could support symmetric shared file system designs as well.

Secure communications between disks and clients is achieved using capabilities that have been cryptographically sealed by the file system manager. Support for these cryptographic operations is placed on the devices. NASD goes beyond all other previous shared file system approaches in that it dramatically raises the semantic level of disk drive operations, from fixed-size blocks to variable-sized objects. These objects can be files or directories and support for partitions, containers for separate groups of files, is provided. The higher semantic level used means that fewer disk commands need be sent over the network per file operation, reducing network overheads and improving scalability.

Gibson has modified NFS and AFS [Gib97] to exploit NASD drives simulated on DEC Alpha workstations, reporting relatively low overhead costs for NASD operations and good scalability across four clients.

4.4 Global File System (GFS)

In the Global File System design, clients service only local file system requests and act as file managers for their own requests; storage devices serve data directly to clients. No direct communication is necessary between clients to enable basic GFS operations so that client failures or bottlenecks do not in general affect other clients.

As shown in figure 2, in a GFS storage system the network-attached storage devices on the peripheral network form a global pool that we call the *Network Storage Pool* (NSP) that can be carved up into many *subpools.* This partitioning into subpools allows the system manager to configure separate subpools, each potentially with different characteristics.

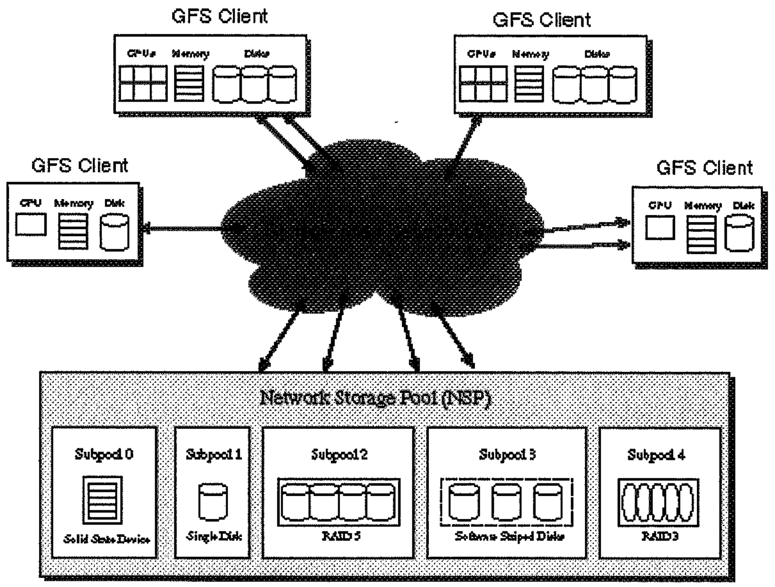


Figure 2: Global File System Distributed Environment.

GFS provides transparent parallel access to storage devices while maintaining standard UNIX file system semantics: user applications see only a single logical device via the standard *open, close, read, write* and *fcntl.* It is a symmetric design with device-based locks and was designed from the ground up as a shared file system implementation [SoE97b]. Current performance results show good scalability to four clients but also show the importance of reducing lock-related overheads and the need for better caching on devices.

Finally, contrast figure 1 with figure 3, which shows the protocol layers traversed by GFS (the diagram would be similar for other shared file systems). The dedicated storage interconnect transport layer obviates the need for protocol stacks that interconnect client to server.

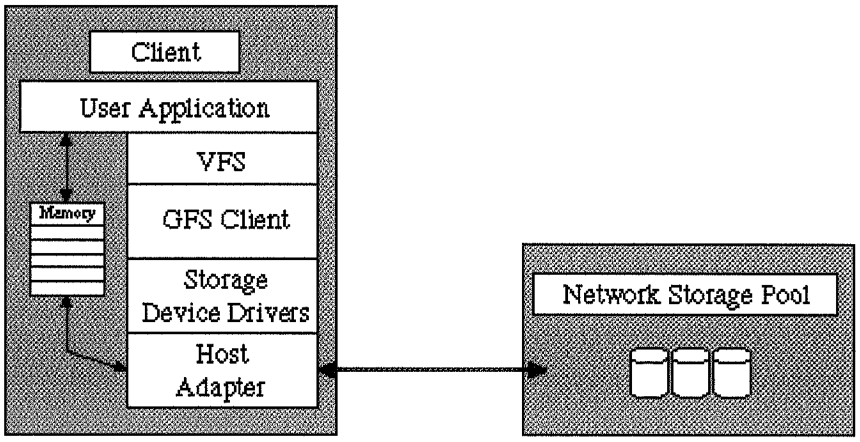


Figure 3: Global File System Protocol "Stack".

Several vendors including Mountaingate, Mercury Computer Systems, and Transoft are developing shared file system products based on Fibre Channel networks[5]. However, little or no published technical information is available on these systems.

5. Summary and Conclusions

Though network-attached storage and the information systems that exploit it are today changing rapidly, some trends are becoming apparent.

Fibre Channel is clearly the network-attached storage interface of choice. Though SSA (Serial Storage Architecture) is a worthy technical competitor to Fibre Channel, it does not have the wide industry support that FC enjoys. Fabrics were developed in the FC standard first but arbitrated loop implementations are growing increasingly sophisticated so that FC-AL (Fibre Channel Arbitrated Loop) appears to be mirroring the evolution of Ethernet. Products are now available that support logically shared FC-AL that can be partitioned into separate domains with smart hubs forwarding traffic between domains only when necessary. Switch-based fabrics are supporting increasing functionality including name servers, which provide hosts with a dynamic database of currently-attached devices and hosts. FC will continue to evolve with faster interfaces, first reaching 200 MBytes/sec by the year 2000 followed by 400 MBytes/sec not long after that.

FC will succeed first in high-end storage applications due to its improved physical interface and scalability compared to parallel SCSI. Disk vendors are currently putting Fibre Channel drives at the same price point as parallel SCSI drives. FC host adapters are not significantly more expensive than parallel SCSI adapters and are comparable in price to Gigabit Ethernet adapters. So Fibre Channel pricing will provide an opportunity for it to displace other networking technologies like Gigabit Ethernet and ATM, but its success strictly as a networking interface is not assured. The ability to have a single interface supporting both networking and storage connections is an appealing way to reduce system cost and complexity.

Shared file systems will continue to increase in popularity due to the availability of FC as a cost-effective, high-performance network-attached storage technology. Application requirements for high availability and performance that shared file systems and FC provide are driving system vendors to support these technologies. We expect most of the first implementations to be asymmetric but with migration paths towards symmetric designs. Some technical issues remain, but a cross-platform shared file system is possible in principal. The principal barriers to cross-platform, shared storage embedded in kernel-level file systems is the lack of a consistent, public, cross-platform installable file system interface; the lack of a standard metadata layout agreed on by all vendors; the tremendous effort required to debug and test kernel-level file systems that must support legacy applications and interfaces; and kernel dependencies built into many aspects of the storage subsystems.

Distributed file systems will continue to be popular since they provide network independence and portability. However, as Web technologies evolve and become more visible at the file system level, distributed file systems may be displaced because they exploit neither the spatial locality nor the performance of network-attached storage interfaces as do shared file systems.

Acknowledgements

I would like to acknowledge the current and past members of the GFS team at the University of Minnesota for insightful discussions about the work described in this paper, including Steve Soltis, Grant Erickson, Ken Preslan, Chris Sabol, Jon Brassow, and Erling Nygaard. Discussions with Tony Shoemaker of Rainmaker Imaging, Ted Fay of Santa Monica Studios, Andy Hendrickson of Industrial Light and Magic, Kevin Mullican and Chuck Spaulding at RFX, Inc., and Alan Poston at NASA Ames Research Center were very helpful in understanding the major benefits of shared file systems and important issues in their design.

References

[AnD95] T. Anderson, M. Dahlin, J. Neefe, D. Paterson, D. Roselli, and R. Wang, "A Serverless Network File System," *ACM Operating Systems Review,* vol. 29, no. 5, December 1995.

[ArB93] D. Arneson, S. Beth, T. Ruwart, and R. Tavakley, "A Testbed for a High Performance File Server," *Proceedings 12th Symposium on Mass Storage Systems,,* Monterey, CA, March 1993.

[Ben95] A. Benner, **Fibre Channel: Gigabit I/O and Communications for Computer Networks**. New York, NY: McGraw-Hill, 1996.

[Bak91] M Baker, *et al.,* "Measurements of a Distributed File System," *Proceedings 1991 Symposium on Operating System Principles,* pp. 198-212, 1991.

[Cha96] A. Chankhunthod, P. Danzig, C. Neerdaels, M. Schwartz, and K. Worrell, "A Hierarchical Internet Object Cache," *Proceedings of the 1996 USENIX Annual Technical Conference, pp. 153-164, January 1996.*

[DEC87] Digital Equipment Corporation, ***Special Issue on Vaxcluster Systems,*** *Digital Technical Journal,* No. 5, September 1987.

[Dem94] D. Deming. **The SCSI Tutor**. Saratoga, CA: ENDL Publishing, 1994.

[DeM95] M. Devarakonda, A. Mohindra, J. Simoneaux, W. Tetzlaff, "Evaluation of Design Alternatives for a Cluster File System," *1995 USENIX Technical Conference,* January 1995.

[Gib96] G. Gibson *et al.,* "A Case for Network-Attached Secure Disks," Technical Report CMU-CS-96-142, Carnegie Mellon University, June 1996.

[Gib97] G. Gibson *et al.,* "File Serving Scaling with Network-Attached Secure Disks," *Proceedings of the ACM Int. Conf. on Measurements and Modeling of Computer Systems (SIGMETRICs '97),* Seattle, WA, June 15-18, 1997.

[Koe95] S. Koegler, "SPANStor Adds on Network Storage with Ease and Convenience," *Network Computing,* November 1, 1995.

[KaG89] R. Katz, G. Gibson, and D. Patterson, "Disk System Architectures for High Performance Computing," *Proceedings of the IEEE,* vol. 77, pp.1842-1858, 1989.

[KiS96] S. Kittur, D. Steel, F. Armand, and J. Lipkis, "Fault Tolerance in a Distributed CHORUS/MiX System," *Proceedings of the USENIX 1996 Annual Technical Conference*, pp. 219-228, January 1996.

[KrL86] N. Kronenberg, H. Levy, W. Strecker, "VAXClusters: A Closely-coupled Distributed System," *ACM Transactions on Computer Systems*, vol. 4, no. 3, pp. 130-146, May 1986.

[LeS90] E. Levy and A. Silberschatz, "Distributed File Systems: Concepts and Examples," *ACM Computing Surveys*, vol. 22, no. 4, pp. 321-374, December 1990.

[Llo92] I. Lloyd, "The Oracle Parallel Server Architecture," *Proceedings of Supercomputing-Europe 92*, pp. 5-7, 1992.

[Mat95] K. Matthews, "Implementing a Shared File System on a HiPPI Disk Array," *Fourtheenth IEEE Symposium on Mass Storage Systems*, pp. 77-88, September 1995.

[Met96] R. Meter, "A Brief Survey on Current Work on Network Attached Peripherals," *ACM Operating Systems Review*, pp. 63-70, January 1996.

[Ous88] J. Ousterhout, A. Cherenson, F. Douglis, M. Nelson, and B. Welch, "The Sprite Network Operating System," *IEEE Computer*, pp. 23-36, February 1988.

[Par94] B. Parwlowshi *et al.*, "NFS Version 3: Design and Implementation," *Proceedings of the Summer USENIX Conference*, 1994.

[Pfi95] G. Pfister, **In Search of Clusters**. Upper Saddle River,NJ: Prentice-Hall.

[RuO95] T. Ruwart and M. O'Keefe, "A 500 Megabyte/Second Disk Array," *Fourth Nasa/Goddard Conference on Mass Storage Systems and Technologies*, College Park, Maryland, March 1995.

[SaL94] M. Sachs, A. Leff, and D. Sevigny, "LAN and I/O Convergence: A Survey of the Issues," *IEEE Computer*, vol. 27, no. 12, pp. 24-33, December 1994.

[SaG85] R. Sandberg, D. Goldberg, S. Kleiman, D. Walsh, and B. Lyon, "Design and Implementation of the Sun Network File System," *Proceedings of the Summer USENIX Conference*, pp. 119-130, 1985.

[Sat90] M. Satyanarayanan, "Scalable, Secure, and Highly Available Distributed File Access," *IEEE Computer*, pp. 9-20, May 1990.

[Sch94] C. Schimmel, **UNIX Systems for Modern Architectures.** Addison-Wesley: Reading, MA, 1995.

[Sea97] D. Seachrist, R. Kay, and A. Gallant, "Wolfpack Howls Its Arrival," BYTE Magazine, pp. 126-130, vol. 22, no. 8, August 1997.

[SoE97a] S. Soltis, G. Erickson, K. Preslan, M. O'Keefe, and T. Ruwart, "The Global File System: A File System for Shared Disk Storage," submitted to the *IEEE Transactions on Parallel and Distributed Systems*, October 1997.

[SoE97b] S. Soltis, G. Erickson, K. Preslan, M. O'Keefe, and T. Ruwart, "The Design and Performance of a Shared Disk File System for IRIX," to appear in the *1997 Joint IEEE and NASA Mass Storage Conference* (these proceedings), College Park, MD, March 1997.

[SoR96] S. Soltis, T. Ruwart, and M. O'Keefe, "The Global File System," *Fifth NASA Goddard Conference on Mass Storage Systems and Technologies,* College Park, MD, September 1996.

[Sri89] V. Srinivasan and J. Mogul, "Spritely NFS: Experiments with Cache-Consistency Protocols," *Proceedings of the 12th ACM Symposium on Operating Systems Principles,* pp. 45-57, 1989.

[Tuc96] M. Tucker, "NFS Accelerators," *SunExpert Magazine,* pp. 59-64, August 1996.

[SwD96] A. Sweeney, D. Doucette, W. Hu, C. Anderson, M. Nishimoto, G. Peck, "Scalability in the XFS File System," *1996 USENIX Technical Conference*, January 1996.

[Vah96] U. Vahalia, **UNIX Internals: The New Frontiers.** Prentice-Hall, Upper Saddle River, NJ, 1996.

[Val93] P. Valduriez, "Parallel Database Systems: the Case for Shared-something," *Proceedings of the Ninth International Conference on Data Engineering,* pp. 460-465, 1993.

[Wat95] R. Watson and R. Coyne, "The Parallel I/O Architecture of the High Performance Storage System (HPSS)," *14th IEEE Symposium on Mass Storage Systems,* pp. 27-44, September 1995.

[Woo95] C. Wood, "Client/Server Data Serving for High Performance Computing," *Fourteenth IEEE Synposium on Mass Storage Systems,* pp. 107-119, Monterey, CA, September 1995.

---

[1] This work was supported by the Office of Naval Research under grant no. N00014-94-1-0846, by the National Science Foundation under grant no. CDA-9414015 and no. ASC-9523480, by NASA through grant no. NAG2-1151 and by equipment grants from Seagate Technology, Brocade Communications, Silicon Graphics Inc. and Ciprico.

[2] See reference [SoE97b] which describes DLOCK, a fine-grain device-based locking mechanism implemented as a SCSI command , as an example. The University of Minnesota team intends to pursue the standardization of this command in the X3T10 (SCSI) committee in collaboration with our industrial partners.

[3] We do not claim these systems are either the most important or the most widely used; rather, they provide a view of the spectrum of design choices and tradeoffs possible. The interested reader is urged to explore the references to learn more about the other shared file systems listed earlier.

[4] More information on the National Storage Industry Consortium, a group of companies and universities involved in standards for network-attached storage devices (NASDs), can be found at http://www.nsic.org/nasd.

[5] See, respectively, the following WWW sites: www.mountaingate.com, www.mc.com, and www.transoft.net.

Protecting File Systems: A Survey of Backup Techniques

Ann L. Chervenak
Vivekanand Vellanki
Zachary Kurmas
{annc,vivek,kurmasz}@cc.gatech.edu
tel +1-404-894-8591
fax +1-404-894-9486

Abstract

This paper presents a survey of backup techniques for protecting file systems. These include such choices as device-based or file-based backup schemes, full vs. incremental backups, and optional data compression. Next, we discuss techniques for on-line backup (backups performed while users continue to access the file system); these techniques include file system locking and creating instantaneous, copy-on-write "snapshots" of the file system. Last, we discuss protecting data from site disasters and media deterioration. The paper then classifies several research and commercial backup systems according to the parameters already described. The classified systems include the UNIX *dump* and *tar* utilities; hierarchical storage managers such as IBM's ADSTAR Distributed Storage Manager, Legato's NetWorker, and UniTree; and several research and academic systems. We conclude with measurements of full and incremental backups over a one-year period in a large networked UNIX environment at Georgia Tech's College of Computing.

1 Introduction

This paper is a survey of backup techniques for file systems. Backups protect file systems from user errors, disk or other hardware failures, software errors that may corrupt the file system and natural disasters. The most common uses of backups are to restore files accidentally deleted by users and to recover from disk failures.

As the capacities of new magnetic disk drives continue to increase at a rate of over 50% per year for the next decade, networked computing environments will grow to include multiple terabytes of disk storage. Meanwhile, the access times and data rates of disk and tape drives will increase at slower rates, about 20% per year. These trends indicate that it will take increasingly long to read the contents of a disk drive and write them to a backup device. Over time, traditional backup schemes are likely to prove too slow.

This paper surveys and classifies existing backup techniques as a prelude to identifying the needs of future systems. The next section characterizes design issues for backup software. Section 3 describes the features of various academic and commercial backup schemes. Section 4 presents measurements of full and incremental backups at the Georgia Institute of Technology's College of Computing. Finally, Section 5 discusses desirable properties of future backup systems.

2 Design Issues for Backup Software

In the following sections, we characterize several features of backup systems: full vs. incremental backups; file-based vs. device-based schemes; support for on-line backups; the use of snapshots and copy-on-write mechanisms; concurrent backups; data compression; and support for tape management and disaster recovery.

2.1 Full vs. Incremental

The simplest way to protect a file system against disk failures or file corruption is to copy the entire contents of the file system to a backup device. The resulting archive is called a **full backup**. If a file system is later lost due to a disk failure, it can be reconstructed from the full backup onto a replacement disk. Individual lost files can also be retrieved. Full backups have two disadvantages: reading and writing the entire file system is slow, and storing a copy of the file system consumes significant capacity on the backup medium.

Faster and smaller backups can be achieved using an **incremental backup** scheme, which copies only those files that have been created or modified since a previous backup. Incremental schemes reduce the size of backups, since only a small percentage of files change on a given day. A typical incremental scheme performs occasional full backups supplemented by frequent incremental backups. Restoring a deleted file or an entire file system is slower in an incremental backup system; recovery may require consulting a chain of backup files, beginning with the last full backup and applying changes recorded in one or more incremental backups.

We characterize several variations of incremental backup techniques. We call the traditional scheme, where occasional full backups are supplemented with frequent incremental backups, a "full+incremental" or simply an "incremental" scheme. Recent systems include variations of incremental backup schemes. An "incremental-only" scheme is used in the IBM Adstar ADSM system (Section 3.3); full backups are eliminated, and files are only written to the backup medium when they change. The UniTree hierarchical storage manager (Section 3.5) can be characterized as a "continuous incremental backup" scheme; the system makes copies of all newly-written data within a few minutes, rather than doing a traditional incremental backup once per day.

2.2 File-Based vs. Device-Based

Files consist of logical blocks. These blocks, also called pages, are typically of fixed size of approximately 8 kilobytes. Each logical file block is stored on a contiguous physical disk block. However, different logical blocks of a file may not be stored contiguously on disk. UNIX uses an index node or *inode* structure to map logical block addresses to the corresponding physical addresses on disk. An inode contains pointers to physical disk blocks; for large files, a single inode may be too small to map all the logical blocks, and is supplemented with indirect blocks that contain additional pointers.

Backup software can operate either on files or on physical disk blocks. **File-based backup** systems understand file structure and copy entire files and directories to backup devices. These systems traverse the pointers stored in each inode and read the physical blocks of each file sequentially. Then backup software writes each file contiguously to the backup medium; this facilitates fast individual file recovery. However, storing files contiguously slows down backups, since extra disk seek operations are required when a file is not stored contiguously on disk. These extra seek operations increase disk overhead and decrease disk throughput. Another disadvantage of file-based incremental backup schemes is that even a small change to a file requires the entire file to be backed up. Tools that use file-based backup include the UNIX *tar* program.

By contrast, **physical** or **device-based backup** systems ignore file structure when copying disk blocks onto the backup medium. This improves backup performance, since backup software performs fewer costly seek operations. However, this approach complicates and slows file restores, since files may not be stored contiguously on the backup medium. The UNIX *dump* program is sometimes called device-based, although it is more accurately considered a hybrid between a pure device-based and a file-based scheme (Section 3.1).

To allow file recovery, device-based backups must include information on how files and directories are organized on disks [23] to correlate blocks on the backup medium with particular files. Thus, device-based programs are likely to be specific to a particular file system implementation and not easily portable. File-based schemes like *tar* are more portable, since the backup file contains contiguous files, and the notion of files is fairly universal. Another disadvantage of device-based backup schemes is that they may introduce data inconsistencies. An operating system kernel may buffer write data before writing the disk; device-based backup schemes that traverse disk blocks in order typically ignore this file cache data and back up older versions of files. By contrast, file-based backup schemes consult the file cache and back up current versions of the files.

2.3 On-line Backup

While many backup programs require that the file system remain quiescent during backup, on-line or active backup systems allow users to continue accessing files during backup. On-line backup systems offer higher availability but introduce consistency problems.

Shumway [23] discusses many of the difficulties with on-line backups when using programs like UNIX *tar* and *dump*. The most serious problems occur when directories are moved during backup, changing the file system hierarchy. If a user moves a directory during backup, then depending on where the backup program is in its traversal of the file system hierarchy, the backup program may either not encounter the directory and therefore fail to copy it to the backup medium or may encounter and copy the directory multiple times in different locations. Most backup programs perform a depth-first traversal of the file system hierarchy. UNIX *tar* operates in a single phase, copying files to the backup medium as it traverses the file system hierarchy. By contrast, *dump* operates in multiple phases, with a scan phase that traverses and records the structure of the file hierarchy, followed by a dump phase that copies files. Single-phase programs like *tar* have the longest vulnerability to file movement; the multiple-phase *dump* program is vulnerable only during the scan phase as it constructs an image of the file system structure.

Other problems for on-line backup include file transformations, deletions and modifications during backup. An example of a file transformation is compression, in which the contents of a file are compressed and written to a new file name (possibly in a new directory), and the original file is deleted. The new compressed file will not be backed up if the backup program has already either completed its scan phase (e.g., *dump*) or traversed that portion of the file system hierarchy (e.g., *tar*); the original file will not be backed up either, since it has been deleted. In general, if files are deleted before the backup program reaches them, they will not be copied to the backup medium. If a file is modified during backup, the results are unpredictable. The backup may contain a mix of old and new data.

Two strategies for overcoming these problems with on-line backup are locking and detection. (The next section discusses a third scheme, copy-on-write.) To prevent directory movement or file modification, some systems lock directories and files to disallow *move*, *write*, *truncate* and *link* commands. Locking limits the availability of the file system, so it is important to minimize the locking period. Alternatively, a backup system can detect file and directory movements and modifications by comparing the old and new state of the file system; if a moved or modified file was left out of the previous backup, the software ensures it is included in the next one.

IBM's ADSM hierarchical storage system (Section 3.3) offers on-line backup with various levels of consistency. At one extreme, the system may be configured not to back up any file that is being modified. At the other extreme, the system copies files to the backup medium regardless of whether they are being modified. Between these extremes, the system will retry copy operations in an attempt to back up a stable and consistent version of the file.

2.4 Snapshots and Copy-on-Write

Another alternative for on-line backup is to create a **snapshot** or frozen, read-only copy of the current state of the file system. The contents of the snapshot may then be copied to a backup device without danger of the file system hierarchy changing from subsequent accesses. Systems such as Andrew (Section 3.9), Petal (Section 3.6) and Spiralog (Section 3.10) can make either full or incremental backups from the frozen snapshot. The system can maintain any number of snapshots, thus providing read-only access to earlier versions of files and directories.

A **copy-on-write** scheme is often is used along with snapshots. Once a snapshot is created, any subsequent modifications to files or directories are applied to newly-created copies of the original data. Blocks are copied only if they are modified, which conserves disk space. The Plan9 (Section 3.8), Petal, and Spiralog backup systems use copy-on-write.

2.5 Concurrent Backups

When performing backups for a large collection of networked machines, it is often desirable to perform backups of multiple file systems in parallel, either to one or to multiple tape drives. Most of the systems discussed in Section 3 allow concurrent backups. The

Amanda Backup Manager (Section 3.7) uses a temporary staging disk to hold backups of several file systems; eventually, these backups are written to a tape drive at high bandwidth. Legato NetWorker (Section 3.4) interleaves concurrent backups from several machines onto a single tape drive. IBM's ADSM (Section 3.3) sends concurrent backups from multiple machines to separate tape drives. Georgia Tech's College of Computing performs backups to up to 25 tape drives in parallel (Section 4).

2.6 Compression

To reduce tape storage and network bandwidth requirements, many backup systems give administrators the option to compress files before backup. The backup system may compress data at the client or file server using compression hardware or software. IBM's ADSM (Section 3.3) and Legato's NetWorker (Section 3.4) include optional file compression. Alternatively, storage devices may compress data immediately before writing files to the backup medium. Many I/O devices, including tape drives, are equipped with compression hardware.

2.7 File Restores

We have mentioned several issues affecting the speed of restoring individual files and entire file systems. Restores will be slower in an incremental backup system, which must begin with the last full backup and apply changes from subsequent incremental backups. Device-based backup schemes may slow down restores, since blocks of a file may not be stored contiguously on the backup medium. An additional concern when restoring entire file systems is that files deleted since the previous backup will reappear in the restored file system.

Performance of individual file restores improves dramatically in systems that maintain online snapshots of earlier versions of the file system, as described in Section 2.4. UniTree (Section 3.5) supports a "trash can" feature for temporary storage of recently deleted files.

The next two sections discuss tape management and disaster recovery issues, many of which can affect the system's ability to restore files.

2.8 Tape Management

Over time, a backup system generates a large number of backup tapes. These tapes are labeled both internally and externally to avoid losing backup data. Internally, each tape begins with a file or label that identifies the file system backups stored on the tape. This information is repeated on the tape's external physical label. Before a backup system writes a backup tape or reads from a tape on a restore operation, it reads the internal tape label to verify that the correct tape has been loaded. Many systems include a relational database that correlates filenames and dates with backup tapes.

Magnetic tape systems face difficult reliability challenges, including errors that are not correctable by error correction codes (ECC), tape wear, head wear and long-term tape storage. A tape error is detected when previously written data cannot be successfully read. Most of these errors are caused by debris that become embedded in the tape surface [17]. Because the rate of such errors is high, all magnetic tape drives incorporate large amounts of error-correction code. Despite this, tapes occasionally encounter errors that cannot be rectified by the ECC. Tape wear is another reliability issue. Magnetic tapes that are frequently read or written eventually wear out. Tapes last on average several hundred passes [2], [12], [16]. However, they wear out sooner if a particular segment of the tape is accessed repeatedly. Frequent stops and starts on the tape cause excessive wear. Tape heads also wear out, typically after a few hundred or thousand hours of use. Another set of reliability concerns involves the long-term storage of data on tape. Over time, tapes are subject to corrosion and to mechanical changes including tape shrinkage, creasing of the edges, and peeling of the magnetic layer [12]. To avoid losing data due to wear, mechanical problems, or tape deterioration, backup systems must maintain information on tape and drive usage, and must replace old tapes and schedule drive maintenance.

Nemeth et al. offer additional practical advice for performing backups in UNIX environments [19]. The correctness of backup procedures must be monitored; backup software should attempt to re-read tapes after dumps are complete. To guard against backup tapes that become unreadable by the tape drives that wrote them because the drive heads drift out of alignment over time, system administrators should periodically attempt to restore files from various tapes, including tapes that are months and years old. Administrators should also attempt to read backup tapes on different drives from those on which they were written to guarantee that if one tape drive is destroyed, other drives will be able to read the tapes.

Since particular tapes may be lost, damaged, or contain errors, it is important that the loss of one tape does not render the entire backup useless. One scheme used at Georgia Tech's College of Computing (Section 4) avoids this problem by doing full backups every week and daily incremental backups with respect to the previous week's full backup. This allows files to be recovered in the event of a loss of either the current week's or previous week's full backup. A scheme developed at Ohio State University maintains multiple redundant backup "chains", with independent sets of full and incremental backups [21]. Such a scheme requires twice as many tapes and drives, but offers a high degree of reliability.

2.9 Disaster Recovery

Nemeth et al. [19] describe techniques that ensure that recovery will be possible after natural disasters. Dump tapes that contain full backups should be stored off-site to prevent data loss in the event of natural disaster or fire. Backup tapes should be secured against access by unauthorized persons, since they contain all of an installation's data. IBM's ADSM system (Section 3.3) automatically generates a disaster recovery plan that contains recovery instructions and scripts; the plan includes a list of necessary backup tapes and tape drives and their physical locations. Legato's NetWorker (Section 3.4) plans to add support for automatically writing off-site copies of data to guard against data loss from local disasters.

3 Backup Systems

Next, we examine several commercial and research backup systems and classify them according to the issues discussed in the last section. Table 1 summarizes the features of these backup programs and systems.

system	incremental scheme	file-based or device-based	on-line backup	concurrent backup
dump	full+incremental	hybrid	No	Yes (with scripts)
tar	full+incremental	file	No	Yes (with scripts)
IBM ADSM	incremental-only	file	Yes	Yes
Legato Networker		file	Yes	Yes (interleaved)
UniTree	continuous incremental		Yes	N/A
Petal and Frangipani	full+incremental	file (uses tar)	Yes (snapshots)	
Amanda	full+incremental	(uses tar or dump)	No	Yes
Plan 9	permanent file storage	N/A	Yes (snapshots)	N/A
Spiralog (LFS)	full+incremental	device	Yes (snapshots)	

Table 1: Classification of backup programs and systems. Blank spaces in the table indicate that information was not available. N/A indicates the category does not apply.

3.1 UNIX dump and rdump

The most common UNIX backup program is *dump* [26], which operates in several passes. *dump* maintains character arrays for directories and inodes. On the first pass of the algorithm, *dump* "marks" each directory by setting a bit in a character array. *dump* also examines each inode, setting an array bit for those inodes that have blocks modified since the last *dump*. On the second pass of the algorithm, *dump* looks at all bits for inodes under a particular directory. If any of the inodes are marked or any subdirectories of the directory are marked, then *dump* leaves the directory marked. Otherwise, *dump* unmarks the directory. This pass continues until all inodes have been examined. On the third pass of the algorithm, *dump* writes the marked directories to the tape or other archive device. Finally, on the fourth pass, *dump* writes the marked inodes along with the modified data blocks.

dump is used to create backups on a local tape drive; the variation *rdump* writes backups to a remote tape drive. *dump* performs incremental backups at different levels, from 0 to 9. A level 0 dump corresponds to a full backup. A level N dump backs up all files that have been modified since the last dump of level less than N.

Different sites devise schedules for *dump* based on the activity of the file systems, the capacity and number of tapes, and the amount of redundancy desired [19]. A simple but expensive schedule is to do level zero dumps (full backups) every day. A more moderate schedule would perform different levels of backup on a daily, weekly, monthly, and annual basis. For example, daily level three backups could be supplemented with weekly level two backups, monthly level one backups, and level zero backups at least once a year. Backup schemes like the Towers of Hanoi sequence [19] alternate backups of many different levels to achieve high redundancy with relatively few tapes; such schemes are complex and may perform poorly on restores, since they access many tapes.

The *dump* program includes options for specifying tape characteristics including capacity, tape density, blocking factor, length, and number of tracks. These parameters are used between the third and fourth passes of the algorithm to estimate tape length and the number of tapes required for the backup. The goal is to avoid over-running the end of a tape.

Full backups (level 0 dumps) must be performed on a quiescent file system. In addition, *dump* offers a verification option that checks the correctness of each volume as it is dumped.

3.2 UNIX tar

The UNIX *tar* [27] program is a file-based archival program. *tar* traverses the file system hierarchy in a single pass. It uses standard UNIX file system read calls to read a file sequentially and then copies it sequentially to the backup medium. Because it stores files contiguously, tar offers efficient restores of individual files. Also, because *tar* writes files and the semantics of files are widely accepted, tar archives are more portable than dump files. Therefore, *tar* is widely used for transferring files and directories as well as for backups.

3.3 IBM ADSM

One commercial storage management system that includes flexible backup functionality is IBM's ADSM, or ADSTAR Distributed Storage Manager [9] [10]. ADSM supports automated backup of multiple clients concurrently to separate backup devices, optionally compresses data before backup, and can write multiple copies of a backup file. The storage manager also maintains multiple versions of files for a specified length of time. ADSM allows selective and incremental backups. Selective backups copy only specified files and directories. The incremental backup scheme, called "progressive incremental" or "incremental-only", duplicates the file the first time it is backed up; thereafter, only changes to the file are recorded, and further full copies are not required.

ADSM supports on-line backup with four different schemes or "modes of serialization." In *static* mode, ADSM will not back up any file that is being modified, hereafter referred to as an *open file*. In *shared static* mode, the system retries backup of an open file a specified number of times or until a static copy of the file is made. *Shared dynamic* mode retries backup of an open file some specified number of times; on the last try, if the file is still open, the system backs up the file regardless of whether it may be modified during backup. Finally, *dynamic* mode backs up all files regardless of whether they are open.

ADSM is a hierarchical storage manager that organizes storage devices as *pools*. Primary pools of magnetic disk drives provide fast access to files. Files that are unlikely to be soon accessed may automatically migrate to lower levels of the storage hierarchy based on file size and age. Lower levels of the hierarchy may include magnetic disk, magnetic tape and optical disk drives. *Copy storage pools* are lower levels of the storage hierarchy that are used specifically for backup. The ADSM backup software copies files in primary pools to one or more copy storage pools. The hierarchical storage manager is tightly integrated with the backup functions; typically, a file is replicated to a copy storage pool before it is migrated to a lower level of the hierarchy.

There are several options for file recovery in ADSM. The system supports *Point-in-Time* recovery, which restores the file system or database to its state at a particular full or incremental backup. For database systems, ADSM can maintain a log of transactions; after a failure, the system is first restored to its state at the last backup, and then the log is used to roll forward the state of the database to the point of the system crash.

ADSM accelerates recovery of files for a single client using "collocation." The system consolidates the data for each client on as few sequential volumes as possible. Clients can initiate their own file recovery.

To facilitate site recovery after a disaster, ADSM automatically generates an updated disaster recovery plan. This plan includes instructions for recovering the server, the list of relevant backups and copy pool volumes needed for recovery, the off-site locations of these volumes, the devices needed to read backups, the amount of space required to restore the file system, system configuration files and shell scripts needed to initiate recovery.

3.4 Legato NetWorker

Legato NetWorker is another commercial storage manager that performs automated network backup [4]. NetWorker allows up to 64 file systems to send data to up to 32 backup storage devices simultaneously. To achieve optimal transfer rates for backup tape drives, NetWorker interleaves backup data sent simultaneously by multiple clients onto a single tape. Clients may optionally compress files before sending them over the network for backup. NetWorker includes application-specific routines that obey the locking conventions of particular applications such as relational databases, and it can backup data from actively-running applications. The software also writes electronic labels on tapes; these labels are checked before new writes are allowed.

NetWorker writes files to the backup medium using a portable, machine-independent file format called NetWorker Open Tape Format. Using this machine-independent format allows clients with different operating systems to write to the same backup tape and allows a client to restore files backed up on another platform.

A server process initiates a NetWorker backup session by setting up a connection between a *save* process on a client and a *media manager* process on a server. The save process reads backup data from a client file system or database and forwards it to the media manager, which writes it to the backup device.

3.5 UniTree

UniTree [18] is a hierarchical mass storage system for UNIX environments originally developed at Lawrence Livermore National Laboratory. UniTree allows automatic migration of files between levels of a storage hierarchy.

UniTree does not perform traditional full or incremental backups, but rather provides *continuous backup* [24]. Upon creation, files are stored in a magnetic disk cache. Within a short period, typically 3 to 30 minutes, UniTree copies newly-created files to one or more lower levels of the storage hierarchy. Thus, files are protected more quickly than in a daily incremental backup. The user can make up to 15 copies of a file on physically distinct media [11]. Future versions of UniTree will provide disaster protection by creating automatic remote copies of files over a wide area network. Because a copy of every file exists at a lower level of the hierarchy, if UniTree needs space in the disk cache for new files, it can quickly purge existing files. For good performance, UniTree maintains all metadata on magnetic disk.

After data loss, files may be recovered from any valid copy in the hierarchy. In addition, UniTree supports a "trash can" feature, in which directories for each user retain recently discarded files. This feature allows users to quickly restore files they have accidentally deleted without requiring reference to backup tapes [11]. UniTree also offers database recovery, including transaction logs.

3.6 Petal and Frangipani

The Petal system [15] from the DEC Systems Research Center allows a collection of network-connected servers to cooperatively manage a pool of physical disks. The pool of disks appears to the servers as a single large virtual disk. Petal provides a copy-on-write snapshot mechanism. When creating a snapshot, Petal pauses applications briefly (for less than one second). Snapshots may be kept online for quick access to previous versions of data. To create a virtual disk backup, Petal simply copies a snapshot to an archive device using a utility such as *tar*. Frangipani [25] is a distributed file system built on top of Petal virtual disks that uses the Petal snapshot facility to perform file system backups.

3.7 Amanda

The Amanda Network Backup Manager [5] [6] was developed at the University of Maryland at College Park to facilitate network backup of many UNIX workstations in parallel. Amanda uses standard UNIX backup programs like *dump* and *tar*. For the best performance, Amanda writes multiple backups in parallel to a temporary holding disk, which later streams the data to a tape drive at its maximum transfer rate.

Amanda backups are controlled by a central backup server host that initiates backups in off-peak hours. The backup server host dictates the backup level for each file system each night. Amanda provides backup history for individual file systems to facilitate restores. The system also allows optional compression for each file system.

Amanda performs checks both before and after backups are run. Before backups, it verifies that the correct tapes are loaded into tape drives and that there is sufficient temporary storage on the holding disk. After backups, Amanda reports any problems that occurred, including disk errors, backup program problems such as permission errors or software crashes, and client hosts that were down.

3.8 Plan 9

The Plan 9 computing environment [20] used a write-once optical disk jukebox for *permanent file storage*. The system also included a magnetic disk file cache for faster file access. Daily on-line backups were made by creating snapshots of the file system. Plan 9 used a copy-on-write scheme; it froze the state of the file system, and made subsequent modifications to a copy of the frozen data. Since old files were not deleted, users could restore individual files or entire file systems to their state on an earlier date. File access permissions were maintained on snapshot copies of files.

3.9 Andrew File System

The Andrew File System (AFS) organizes files into *volumes*, which are the focus of system management, including backup and restore [8] [14]. A volume is a collection of files and directories. Typically, a user's home directory resides in a different volume, as do different binary directories. A volume resides entirely on one magnetic disk partition, but a disk partition may hold multiple volumes. Andrew allows *cloning* or copying of volumes.

Backup in AFS begins by creating a clone called a *backup volume* that contains hard links to all the files in the original volume. The backup volume is then dumped to tape using either a full or incremental backup scheme. After writing the backup to tape, Andrew may delete the backup volume or may keep it for on-line read-only access to previous versions of data, the equivalent of a snapshot.

One or more machines in the networked file system are designated as Backup Machines. These machines coordinate tape drive operations and maintain a database with information about tapes and dump schedules for individual volumes and sets of volumes.

3.10 Log-Structured File System Backup

The Spiralog backup system from DEC [7] provides on-line backup of a log-structured file system or LFS [22]. A log-structured file system writes all modifications to disk sequentially in an append-only log [22]. The log is divided into segments, which are composed of disk blocks. Because the log is written sequentially, segments appear in the log in temporal order. Over time, as portions of files are edited and deleted, portions of segments contain obsolete data. A segment cleaner periodically traverses the log and compresses valid information into new segments, freeing up the original segments. Periodically, LFS writes a checkpoint to the log, which records the mapping between all logical file blocks in the file system and their location in the log. LFS also puts a pointer to the end of the log at a fixed location on the disk. Upon crash recovery, LFS quickly restores the file system by reading the last checkpoint and using the log to roll forward from the checkpoint. LFS can create snapshots, also called "time travel", of the file system by creating checkpoints and turning off the segment cleaner so that space is not reclaimed after files are deleted.

Spiralog exploits several features of LFS in its backup system [7]. It uses LFS's checkpoint capability to create a snapshot of the file system that is then copied to the backup medium. Spiralog uses a physical or device-based backup scheme; it copies LFS segments rather than files, and therefore can read data sequentially from disk and write it to the backup medium at a high transfer rate. The temporal ordering of segments in LFS allows Spiralog to perform simple incremental backup, copying only those segments to the backup medium that were modified since the last backup. To facilitate restores of individual files, Spiralog writes a copy of the directory tree before writing segments to the backup medium. These directories are consulted to determine which segments contain data for the requested file, and only these segments need be read from the backup medium. Restoring an entire volume requires copying the last incremental backup and any preceding backups on which

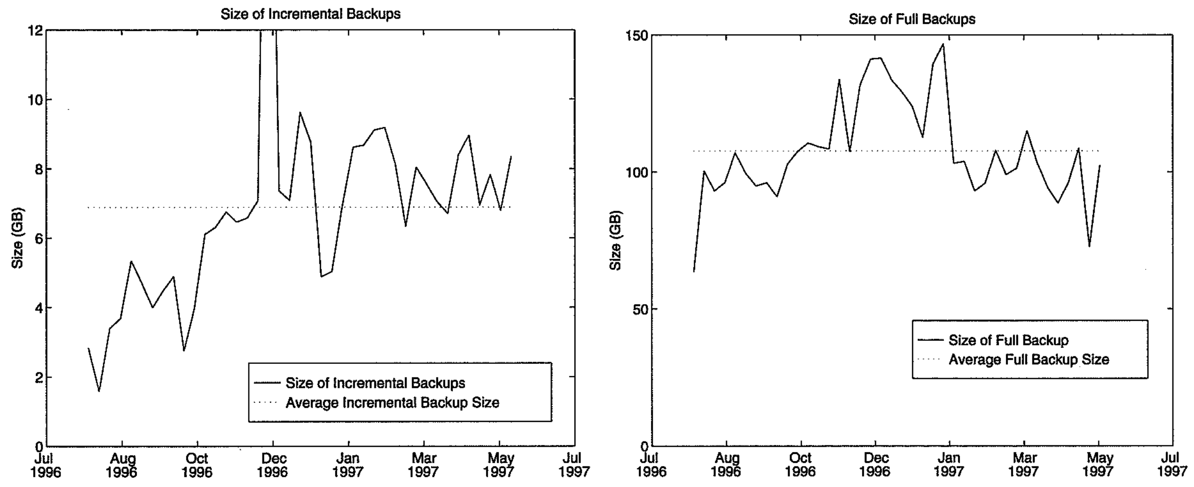


Figure 1: *Size of incremental and full backups in Georgia Tech's College of Computing from August 1996 to June 1997.*

it depends to disk.

3.11 SGI Terabyte Backup

Silicon Graphics implemented a system that includes an Origin 2000 server, 138 disks, 66 UltraSCSI channels, and 38 IBM 3590 tape drives. This system achieves a backup rate of over a terabyte of data per hour [3] using both Legato NetWorker and Spectra Logic Alexandria hierarchical storage manager software. This performance is achieved while running a TPC-C database benchmark at an applied load of 4500 transactions per minute.

4 Georgia Tech Backup Measurements

The College of Computing at Georgia Tech has more than 200 UNIX workstations, 150 Macintoshes and a smaller number of Intel personal computers running Windows/NT. Until recently, these machines were backed up using a locally-defined collection of backup scripts that performed full backups once a week and incremental backups daily. Backups were performed to 25 tape drives simultaneously. Each tape drive could write approximately 2.5 gigabytes of compressed data per tape.

Figure 1 shows the average sizes of incremental and full backups, respectively, over a period of approximately a year. The incremental backups show a steady increase in the average data produced per week; around December 1996, there was a large increase in data, probably due to temporary storage of image and video files from graphics researchers. Incremental backups typically consume less than 10 gigabytes, while full backups consume over 100 gigabytes per week. Again, the full backup sizes increase considerably around December 1996, and drop down again around March 1997, indicating that large files were created and later removed from the system.

Backup times represent the sum of time spent on all 25 tape drives. The aggregate time for incremental backups was approximately 20 hours, while full backups took approximately 200 hours. Distributed over 25 tape drives, this means that full backups consumed about 8 hours of real time per week per drive, typically spread over several days.

More recently, the college has switched to using Legato Networker software to backup the UNIX machines and Retrospect to backup the Macintosh and Intel machines. In addition, the college has acquired two tape libraries, each containing two drives. Each tape library holds 800 gigabytes of compressed data. These libraries use Exabyte Mammoth tape drives, each capable of writing 20 gigabytes of compressed data per tape. Under the new scheme, full backups of the UNIX machines are performed once per quarter, Level 5 backups are performed once a month, and Level 7 backups are done once a week. Finally, the scheme includes daily incremental backups.

The College supports 700 active users. System administrators typically receive one request per day to restore a file that has been accidentally deleted by a user. Restores of entire file systems due to disk failures are less frequent, occurring approximately once per month.

5 Summary

We have developed a characterization of backup systems and used it to categorize a number of commercial and research backup schemes. First, we classified incremental backup schemes. The *full+incremental* scheme supplements occasional full backups with frequent incremental backups. An *incremental-only* scheme avoids full backups of a file system and only writes incremental changes. A *continuous incremental* system copies new data within a few minutes of its being written.

Next, we differentiated between systems that copy data to the backup medium as contiguous files vs. those that copy physical blocks regardless of file structure. *File-based* backups facilitate restores and are more portable, while *device-based* backups access the disk more efficiently.

We discussed problems associated with providing on-line backup and considered several schemes. The file system can disallow writes during backup, or it can create instantaneous *"snapshots"* or read-only copies of the file system, which may then be copied to the backup medium. Many systems that create snapshots provide a *copy-on-write* scheme that replicates data only when it must be modified, thus saving substantial disk space.

We discussed several schemes for providing *concurrent* backup of multiple file systems over a network. We also discussed the ability of file systems to compress files before they are written to the backup device. Many systems also provide support for managing tapes and tape drives, and some provide automatic protection from site disasters by creating and managing remote copies of data.

As file systems grow to multiple terabytes, it is likely that new backup strategies will be required to protect them. The most promising techniques for handling very large file systems appear to be incremental-only backup schemes, device-based backup to use disk bandwidth efficiently and to avoid writing entire files based on small file changes, snapshots and copy-on-write for on-line backup, compression of data before backup, and automated creation of off-site backup files.

6 Acknowledgements

The authors are grateful to Dan Forsyth for his generous assistance in gathering statistics on Georgia Tech's backup environment. Vikas Gupta did initial work in identifying papers used in this study. This work was funded in part by NSF grant CCR-9702609.

References

[1] K. A. Anderson and B. H. Kirouac. A Simple and Free System for Automated Network Backups. In *The Third Annual System Adminstration, Networking and Security Conference (SANS III)*, pages 63–68. Open Systems Conference Board, April 1994.

[2] B. Bhushan. *Tribology and Mechanics of Magnetic Storage Devices*. Springer-Verlag, New York, 1990.

[3] S. Cariapa, R. Clark, and B. Cox. Origin2000 One-Terabyte Per Hour Backup White Paper. http://www.sgi.com/Technology/teraback/teraback.html.

[4] L. Corp. Legato Networker Documentation. http://www.legato.com.

[5] J. da Silva, O. Gudmundsson, and D. Mosse. Performance of a Parallel Network Backup Manager. In *USENIX Conference Proceedings*, pages 17 – 26, 1992.

[6] 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.

[7] R. Green, A. Baird, and C. Davies. Designing a Fast, On-line Backup System for a Log-structured File System. *Digital Technical Journal*, October 1996.

[8] S. Hecht. Andrew Backup System. In *USENIX Proceedings of the Workshop on Large Installation Systems Administration*, pages pp. 35–38, November 1988.

[9] I.B.M. ADSTAR Distributed Storage Manager (ADSM) – Distributed Data Recovery White Paper. http://www.storage.ibm.com/storage/software/adsm/adwhddr.htm.

[10] I.B.M. ADSTAR Distributed Storage Manager (ADSM) – Frequently Asked Questions. http://www.storage.ibm.com/storage/software/adsm/adfaq.htm.

[11] F. Kim. UniTree: A Closer Look at Solving the Data Storage Problem. UniTree Software Inc., http://www.unitree.com/newpage/wpaper.htm.

[12] T. Kitahara. On the Long Term Storage of Metal Tapes. Fuji Photo Film Co. Magnetic Recording Lab; FIAT/IFTA Technical Commission in Hilversum (The Netherlands), June 14th 1988.

[13] R. Kolstad. A Next Step in Backup and Restore Technology. In *USENIX Proceedings of the 5th Conference on Large Installation Systems Administration*, pages 73–78, September 1991.

[14] S. Lammert. The AFS 3.0 Backup System. In *USENIX Proceedings of the 4th Conference on Large Installation Systems Administration*, pages 143–147, October 1990.

[15] E. K. Lee and C. A. Thekkath. Petal: Distributed Virtual Disks. In *Seventh International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS-VII)*, October 1996.

[16] J. C. Mallinson. Magnetic Tape Recording: Archival Considerations. In *Digest of Papers*. Tenth IEEE Symposium on Mass Storage Systems, May 1990.

[17] C. D. Mee and E. D. Daniel, editors. *Magnetic Recording, Volume II: Computer Data Storage*. McGraw-Hill, New York, 1988.

[18] NCSA. UniTree Mass Storage System Frequently Asked Questions. http://consult.ncsa.uiuc.edu/docs/unitree.

[19] E. Nemeth, G. Snyder, S. Seebass, and T. R. Hein. *UNIX System Administration Handbook*. Prentice Hall, Inc., Upper Saddle River, New Jersey, 1995.

[20] R. Pike, D. Presotto, K. Thompson, and H. Trickey. "Plan 9 from Bell Labs".

[21] S. M. Romig. Backup at Ohio State, Take 2. In *USENIX Proceedings of the 4th Conference on Large Installation Systems Administration*, pages 137 – 142, October 1990.

[22] M. Rosenblum and J. Ousterhout. Log-Structured File System. In *Proceedings of the 13th ACM Symposium on Operating Systems Principles*, pages 1–15, June 1991.

[23] S. Shumway. Issues in On-line Backup. In *USENIX Proceedings of the 5th Conference on Large Installation Systems Administration*, pages 81–88, September 1991.

[24] U. Software. Technical Overview. http://www.unitree.com/newpage/techover.htm.

[25] C. A. Thekkath, T. Mann, and E. K. Lee. Frangipani: A Scalable Distributed File System. In *Sixteenth ACM Symposium on Operating System Principles (SOSP-16)*, October 5-8 1997.

[26] DUMP(8). Unix System V man page.

[27] TAR(1). Unix System V man page.

[28] E. D. Zwicky. Torture-testing Backup and Archive Programs: Things You Ought to Know But Probably Would Rather Not. In *USENIX Proceedings of the 5th Conference on Large Installation Systems Administration*, pages 181–190, September 1991.

Page intentionally left blank

The Dynamics of Obsolescence and the Challenges of Legacy Data Storage

Gordon E. MacKinnon
DataSure Services
P.O. Box 42016, 2200 Oak Bay Avenue
Victoria, BC CANADA
V8R 6T4
E-mail: mackinnon@datasure.com
Tel: +1-250-598-6831
Fax: +1-250-598-6841

Abstract

The constant improvement in mass storage technology is not without risks. The dynamic nature of the storage technology and market demands for improvements in price and performance put at risk the long-term viability of data stored on legacy media.

The risks come from a number of factors at work in the market and industry. Despite the efforts of groups such as AIIM, IEEE and THIC toward the adoption of open standards there is still a tendency of manufacturers and developers to create proprietary solutions, ostensibly to enhance performance through unique advances.

The problems arise on a number of levels; in media, hardware, software and operating system to varying degrees depending on the media, form factor, format and maturity of the technology. One of the few constants is that the market itself is by far the most significant influence. What we will cover in this paper are specific examples of these kinds of problems, and some suggested solutions.

Issues

Our examination will cover two areas. We will look at the comparative performance of some of the storage options and their strengths and weaknesses. We can then develop strategies to minimize exposure to obsolescence, and minimize the impact of technological change to best maintain the data and its accessibility.

In order to examine the problems we must start with some of the variables of long term data storage choices currently available in the hardware / media. There are five major competing objectives: access time, throughput, capacity, reliability and price. These are competing in as much as the two main technologies, tape and removable disks have specific advantages and disadvantages in these areas. The solution of best fit for the capacity and performance may not provide the best value in the longer term.

Access Time

The random access performance of the optical, magneto-optical CD-R, and robotics with multiple drives can improve access time.

- Kodak ODW25 14" WORM 25GB - 700ms
- Kodak OD2000 series14" WORM jukebox - 6.5 second Disk exchange
- Magneto-Optical 2.6 GB - 19ms access time

- Pioneer DRM1004X CD-R Jukebox - 110ms
- DLT7000 - 60 second average access time per tape
- Advanced Metal Evaporated Tape / AIT - 28 second average access time per tape

Throughput

Similarly throughput can also be addressed through arrays with multiple drives to multiply throughput. There are two measurements of throughput, burst and sustained transfer rates, and we will look at sustained transfer rates for single drives.

- SONY AIT - 6 MB/sec
- DLT 7000 - 5 MB/sec
- DDS-3 - 1.2 MB/sec
- Magneto-Optical 2.6GB - 6 MB/sec
- Pioneer DRM1004X 12x CD-R Jukebox - 1.8 MB/sec

Capacity

The most obvious issue regarding mass storage is capacity. This can be addressed in two ways, through capacity of the individual media pieces, and capacity of robotic arrays. Individual drive capacities are are shown below.

- SONY DIR-1000 96 GB - 82 GB formatted
- 14" Optical WORM from Kodak provides - 25 GB per disk
- DLT7000 - 35 GB
- SONY AIT SDX-300C - 25GB
- DDS-3 - 12GB
- Magneto-Optical - 2.6GB, 2 sides
- Magneto-Optical - 5.2GB 2 sides
- CD-R - 650 MB

In both areas, tape typically has the advantage.

Reliability

Regarding longevity, disk media comes out ahead in reliability, with a lifespan estimated in the range of 30 to 100 years. Older tape media suffers from degradation much earlier.

- Advanced Metal Evaporated Tape (SONY AIT, EXABYTE Mammoth) - 30 years
- DLT - 30 years
- Magneto-Optical - 30 years
- CD-R - 100 years

The real cost of reliability is hidden in the maintenance costs as the data is migrated more frequently. This includes the cost of additional media over the life cycle of the data and the cost of additional resources required for migration. Depending on the amount of data and the frequency of migration, the advantages of less expensive initial media cost could be offset by the cost of maintenance.

Price

There are three factors in the cost of technology: media costs, drive cost and less obviously, cost of ownership. In terms of cost per GB, tape has a significant lead over MO, WORM, and CD-R. Similarly, the cost of drives also favors tape. However if one is to factor in the cost of ownership, the advantage is less clear. The reason for this is that longevity of tape media is considerably less than MO, WORM, and CD-R, and as such, the issue of duplication and migration costs is introduced. Although this issue is difficult to quantify it could become a significant factor over long periods. Also influencing the cost of ownership is the physical storage space required. With the fullness of time and the cost of real estate, a premium for high storage density can be offset by lower storage space requirements and costs.

- ADIC SCALAR 458 DLT7000 48 cart 2 drive 1680 GB - $47,000, $27.97/GB
- EXABYTE EXB 480 80 cart 1 drive 1600GB - $45,000, $28.12/GB
- ADIC SCALAR 218 DLT7000 18 cart 630 GB - $20,000, $31.75/GB
- Pioneer DRM5004X CD-R jukebox, 500 disc, 325 GB - $20,000, $61.00/GB

To be considered also would be the strategy of planning for limited life cycles of systems, with migrations to new products in the not to distant future. Such a strategy presumes improved price, performance and capacity, so as to lower long term storage cost with increased density and productivity.

Critical Path and Point of Failure

Given the choice of particular technologies, we will now examine the points of failure, their causal influences and proactive measures to minimize their impact. We will look at how both physical and market issues influence the obsolescence of technology.

Physical Issues

The physical issues affecting storage have improved in recent years, as there are higher capacity devices, and many of the drives are backward compatible.

Media Degradation

This is the most obvious failure in long term storage. It can be countered by multiple archive copies, a pre-emptive migration strategy, optimum storage environments, and a choice of media influenced by the anticipated life cycle of the resident data.

Although tape is less expensive per GB, keep in mind the cost of migration and vigilance in terms of systems and personnel costs over the life cycle of the data. Modern optical and tape robotics and software can assist in the maintenance of data.

Drive Failure

As a function of their contact with media, tape drive heads are subject to replacement more frequently than the heads in optical drives. All drives are subject to wear, and maintaining critical parts inventory can assist in maintaining drives and improving drive availability in the future.

Maintaining multiple drives can mitigate this. In the case of long term archives that may not be migrated and are resident on legacy media, care must be taken to ensure that compatible and operational drives, as well as suitable systems, are available. Tests should be executed subsequent to major upgrades to ensure continued access to the data, as new systems and software may not recognize legacy drives and formats.

Operational Problems

There is the chance that through oversight or changes, legacy data is unavailable due to operational problems such as insufficient documentation and legacy procedures. Consider the consequences of password protected backup tapes without passwords or legacy applications that lack documentation. Some older systems use proprietary hardware.

Market Dynamics

Beyond the physical problems of technology, market dynamics play a big role in determining the viability of technologies. The market dynamics we will look at will examine the impact of changes in the marketplace that influence the critical areas of media, hardware, archival software, and operating systems. Over the life of the data, the changes induced by the marketplace are most likely to influence the availability and prevalence of various technologies. The fast pace of change and innovation combined with the vagaries

Media

As storage demands grow we find capacity demands continue to increase. One of the consequences of the growth of capacity is the increasingly shorter production cycle for media of lesser capacity. This is not of itself a crucial problem because of the usual backward compatibility of newer drives and software. Aggressive price performance demands of the newer technology limit compatibility to not so distant generations. The limitations of past technologies may be surpassed by the demands of the marketplace, causing a migration to new form factors and formats. This can lead to a proliferation of legacy media form factor and formats.

This problem can be compounded by reorganization within firms and agencies where legacy formats and systems are orphaned by standardization, leaving media without the systems that created them.

Hardware / Software / Operating Systems

In the case of manufacturers and developers the issues of profitability and market-share/mind-share in relation to the industry as a whole are most critical. The first concern regarding profitability is of course the long-term survival of the manufacturers and developers, and their continuing ability to support and maintain migration paths and future iterations of technology. Their failure, poor performance, or perceived poor performance can have irrevocable effects on the market share and mind-share of the firms. With a lower profile in the marketplace, the technologies of the affected firms are less likely to attract broad support and compatibility from other industry players, and less support further marginalizes the technologies. Poor profit performance of some products can lead to refocusing of corporate resources

With the increasing scrutiny of the corporate world by investment fund managers, the fiscal performance of the individual firms is examined in great detail on a quarterly basis. The consequences of meeting or not meeting performance expectations can have a large impact on share values. This puts pressure on firms to deliver high profit margins.

Poor fiscal performance can lead to rumors of takeovers and uncertainty, resulting in erosion of confidence in the market, the marketplace, and sometimes the loss of valuable human resources within the organization. Pressures of the financial market can make firms attractive takeover targets, with the inevitable result that competing product lines are rationalized.

Dramatic response to poor financial performance through restructuring can lead to the loss of key personnel and morale, short-term market share attrition, which is balanced against the benefit of longer term stability.

Technological obsolescence can happen quickly as new storage requirements surpass capacities. This is most true with nonstandard technology. Without aggressive marketing and production to gain marketshare and create an installed base early, nonstandard technology often remains a niche market until superseded by capacity demands.

Production

Within manufacturers, products compete for limited production facilities: the premature discontinuation of older technology facing intense competition and falling profit margins, allowing them to concentrate resources on newer and more profitable technologies. This in itself is not as much of a short term risk because the newer products are usually backwards compatible. In the long term, however, this leads to shorter product cycles and the proliferation of legacy equipment for the consumers. Low inventory production methods and shortages of key components from time to time cause some products to be unavailable in mid production cycle. This can cause problems for clients by causing them to wait or choose a competing technology and increase proliferation if the need is more immediate.

Similarly the increased scrutiny also causes software development firms to focus their critical human resources on the largest market segments, ignoring or delaying the development of products and drivers of the marginalized technology, perpetuating the cycle of decline of these products. We are led to examine the corporate structure of the manufacturers and developers and its role in the marketplace.

Niche Software

These are companies that specialize in specific market segments. Their small areas of expertise promote broad support of hardware within their area of specialization. They add to the variety of solutions available for the end user. With mergers and acquisitions, some of the smaller firms are vulnerable to larger rivals, or often the intellectual properties of some firms are of enough value to warrant a takeover. The purchasing firm saves time and development costs, and is given an instant entry into new market segments, or expanded product lines.

The challenge of supporting a range of underlying OSs and diverse automated hardware brings different approaches to file storage, introducing the issue of file storage management systems. This eventually creates a problem in migration to other file formats, and the potential for orphaned legacy file systems, no longer supported, or no longer supporting critical hardware.

Strategies

Although there are no perfect solutions to these issues, there are strategies that can help to limit the impact of the risks.

Considering the longer term costs of ownership, the premium price of AIT, DLT 7000 or 5.2 GB M/O drives are more than offset by the longevity and density of the media and the youth of the technology. Depending on your speed and access demands, higher capacity solutions deliver better value in the long term.

One of the big concerns would be for lessening importance of the long-term interests of the storage users in the decision processes of the manufacturers and developers. The larger financial and corporate interests of the manufacturers and developers ultimately determine further development or discontinuation of technology.

The promotion and wide spread adoption of open standards lessens the risk of being limited by proprietary technologies.

The adoption of technology standards within organizations can lead to more homogenous technological environment. The homogenous environment offers the advantage of hardware redundancies and greater flexibility to restructure within the organization. The standardization occurs at the expense of users' ability to access specific non-standard technology that may be better suited to specific tasks, and the individual requirements are compromised for the standards.

At the File Storage Management System level, there have been first some efforts to adapt a standard tape format to include File Level Metadata in a tape format [1]. This has evolved into the work of File-Level Metadata for Portability of Sequential Storage (FMP) Study Group [2], which has culminated in the Association for Information and Image Management International ANSI/AIIM MS66 Proposed standard, Metadata for Interchange of Files on Sequential Storage Media Between File Storage Management Systems (FSMS) [3].

To this end public and private sector customers can specify that proposals require adherence to these standards. Early support from large clients in this manner can help to establish market momentum and an installed base for the standards, reinforcing the commitment of vendors to the standards, and attracting new participants. This in turn creates a broader installed base and helps to build momentum.

Continuous monitoring and migration can help to ensure the viability of legacy data, and its compatibility with current technology.

The maintenance of hardware, software, O/S and operations manual archives can help to ensure the accessibility of obsolete technologies; this is best complemented with ongoing testing to ensure the viability of legacy data.

Conclusion

With an explosion of on-line data driven by low cost disk storage, capacity and performance demands for removable media will grow and find new technologies.

Manufacturers and developers will evolve with the market place, and technologies will be superseded.

Most of all, continuous vigilance of the technologies and industry is essential to best deliver proactive responses to inevitable change, and the long-term commitment of appropriate budget resources to accomplish these ends. As a user of the technology you are not just buying the technology but also a migration path to the future.

References

[1] A Straw Man Proposal for a Standard Tape Format, Ben Kobler, Goddard Space Flight Center, and Joel Williams, Systems Engineering and Security.
http://ses2.ses-inc.com/~joelw/TAPE_FMT/Straw_Man.html

[2] Progress in Defining a Standard for File-Level Metadata, Joel Williams, Systems Engineering and Security.
http://ses2.ses-inc.com/~joelw/TAPE_FMT/Format_Progress.html

[3] Metadata for Interchange of Files on Sequential Storage Media Between File Storage Management Systems (FSMS) Association for Information and Image Management International ANSI/AIIM MS66 Proposed.
http://ses2.ses-inc.com/~joelw/TAPE_FMT/RSD011098.html

The Design and Performance of a Shared Disk File System for IRIX

Steve Soltis, Grant Erickson, Ken Preslan, Matthew O'Keefe, and Tom Ruwart
Department of Electrical and Computer Engineering
and
Laboratory for Computational Science and Engineering
University of Minnesota
Minneapolis, MN 55455
okeefe@ece.umn.edu
+1-612-625-6306

Abstract: In this paper we present a new storage architecture for clusters that creates a shared memory of disk storage that is uniformly accessible to all cluster clients, scales to large capacity, and provides very high performance and connectivity. The cluster structure resembles a symmetric multiprocessor (SMP) in that clients (processors) can access disk data (memory) across a local area network like Fibre Channel (a bus or other interconnection network). All clients can see and access the same disk data with perfect consistency. Our approach avoids buffer copy overheads and server bottlenecks found in traditional file systems while scaling to potentially large numbers of clients and large capacity disk systems.

We provide a description of the basic file system design, our current implementation on SGI IRIX operating system, and detailed benchmarking and performance analysis results. Good speedup and throughput is achieved for large files across 3 clients and 4 high performance disk arrays and on other client-array configurations[1] .

1. Introduction

Computer architects have for many years struggled with the problem of fast and efficient transfer of data between main memory and external storage devices, primarily disks. Until recently, it had been convenient (and certainly accurate) to point out that disk drives were three or four orders of magnitude slower than main memory and that their rate of capacity increase (25% per year), access time decrease (1/3 over 10 years), and bandwidth increase (20% per year) were below the corresponding rates for both IC logic for processors and DRAM for main memory trends. However, significant innovation in nearly all aspects of disk technology have accelerated these disk technology improvement curves so that since about 1992:

- capacity increases for disks at 60%/year are now roughly equal to those found in main memory
- bandwidth off the media is now increasing on average at about 40%/year
- access times are greatly reduced for some accesses that exploit on-board disk cache.

As was pointed out in the original RAID (Redundant Arrays of Inexpensive Disks) paper [6], a commodity market in SCSI (Small Computer Systems Interface) disk drives encouraged single-chip integration of SCSI controllers. Coupled with an on-board microprocessor and recent advances in high speed serial interfaces, disks now can communicate with clients over intelligent, fast, and highly functional interfaces such as Fibre Channel. This interface technology combines both network and storage features and provides an industry-standard, high-bandwidth, switched interconnection network between clients and drives. These dramatic developments have encouraged us to propose a revolutionary rather than evolutionary approach to designing future storage architectures, one that assumes disks are highly-capable peer devices available directly on a network.

Traditional client/server distributed computing is limited in that it simply provides a mechanism for a client machine to transparently access data on a remote server through the client's local file system. Though useful, this approach limits the potential storage efficiency and speed that can be realized by distributed systems. The server is a potential bottleneck and single point of failure[2]. To avoid both problems requires expensive redundancy (e.g., a Tandem system) or specialized hardware (for example, hardware built by Auspex or Maximum Strategies) tuned for the network file system protocols such as NFS. An alternative and increasing popular approach is *clustering,* which physically integrates stand-alone computers using fast networks, shared disk storage, and a single system image to create scalable compute and data servers [11], [7], [1].

An important component in several cluster designs is a *shared file system* that allows cluster machines to directly access shared disk devices across a network [7], [9], [4], [3], [11], [8] instead of through a server, increasing cluster performance and availability. In addition, we will show that such a shared file system also makes each node in a *storage area network* (SAN) more effective. A SAN consists of a local area network that allows storage devices to be directly attached to the network.

Disk drive and LAN speeds have increased gradually over the last ten years. Hard disk drives have had transfer rates from 1 to 3 Megabytes/second directly from media, though recently these rates have increased to 5 and in the next year will likely be over 20 Megabytes/second. During that period, Ethernet bandwidth has been limited by its shared physical media to less than 1 MB/sec. Hence, though direct disk attachment to networks is now possible [5], the network bandwidth has generally been too low to make it possible to exploit all the aggregate disk bandwidth actually available on the network. This imbalance between disk drive and Ethernet speed has become even more pronounced with RAID devices and multi-level RAID hierarchies which have aggregate bandwidths of several hundred Megabytes/second or more [12].

Fortunately, recent advances in switching technology, fiber optics and the convergence of network and channel interfaces [13], [10] are allowing order-of-magnitude improvements in network latency and bandwidth through new technologies like Fibre Channel and Gigabit Ethernet. Open standards and high-volume markets, combined with the constant increase in functionality and decrease in cost for microelectronic devices, will drive down network costs. The previous speed imbalance between disk drives and networks will be reversed: parallel drive designs will be needed to exploit switched network bandwidth and meet the requirements of tomorrow's demanding applications.

For example, the new Fibre Channel standard integrates both storage and networking capabilities into a single interface that currently has a speed of 100 Megabytes/second (and a growth path to 400 Megabytes/second), allows both low-cost loop connections (much like FDDI rings) with up to 126 devices at distances beyond 100s of meters and is scalable to 100s or 1000s of devices with Fibre Channel switches. Yet Fibre Channel will achieve very widespread use with disk drives and adapters priced about the same as parallel SCSI technology [2]. In contrast, today's parallel SCSI technology supports only about 8 devices per bus with each bus extending at most 25 meters making the technology effectively unscalable.

These new network technologies will certainly improve the performance of today's client-server networks. However, the advances in network-attached storage interfaces, network bandwidth and scalability, disk bandwidths, capacities and access times along with demanding new applications requiring high bandwidth and high availability challenge the basic client-server architecture upon which distributed systems have been constructed. Distributed systems in the future will increasingly rely on clustering for high availability and will require richly interconnected storage networks to support data-intensive computing.

Given these fast, low-cost, switched networks, a serious review of the division of responsibilities between clients, servers, and storage devices has lead us to an alternative storage architecture, based upon our Global File System (GFS) [17] design that is *serverless* and consists only of *clients* and *networked storage devices.* This proposal motivates the design of the GFS, outlines its basic structure, describes the current software and related performance results and how we intend to extend and test the GFS within the context of an innovative hardware infrastructure here at the University of Minnesota.

A key GFS goal is to remove the master-slave structure found in current distributed computing client-server environments. Instead, given the low latency and high bandwidth of new network technologies, we can design a *symmetric multi-client* system where multiple clients access storage devices across a fast switched network such as Fibre Channel. This allows a cluster to behave much like a symmetric multiprocessor: processors (clients) are equal in the eyes of the kernel (there is no master) and each has equal access to the main memory (disk drives or other storage devices) via a fast bus, multistage network, or crossbar switch (e.g., a fast switched network such as Fibre Channel). This structure has many advantages:

- The GFS provides a storage architecture that allows the storage system designer and administrator *to pool disk drives into a shared disk memory* equally accessible to all clients in the system.

- There is *no single point-of-failure* for a storage device since it is not attached to a single client, thus allowing for fail-over redundancy [20]. Low-level RAID striping provides redundancy at in the disk drives, much like error-correcting DRAM main memories.

- The GFS architecture can *exploit the bandwidth capabilities* both within and across next-generation PCs, desktop workstations, high-end servers and supercomputers.

- *Local client bandwidth need not be wasted* in making transfers from a local storage device to another client as in client-server architectures.

- The *GFS architecture is inherently more reliable than other distributed file systems* since it is easier to build redundancy into a disk array than it is to insure that a complicated server (including hardware, software, and network connections) does not fail.

- The size of the file system and consequently the *size of a single file is not limited* by the size of the storage subsystem on any given client.

- The file system may *span multiple storage devices.*

- Each client connected to the peripheral network *views the devices as locally attached.*

These GFS advantages are relevant both in tightly-integrated cluster architectures but also in more loosely-coupled storage area networks, enabling more efficient data transfer and sharing.

2. Global File System—Architecture and Design

The Global File System is a distributed file system based on shared network-attached storage. Clients service only local file system requests and act as file managers for their own requests; storage devices serve data directly to clients. No direct communication is necessary between clients to enable basic GFS operation so that client failures or bottlenecks do not in general affect other clients.

How GFS Views Network Storage

As shown in Figure 1, in a GFS storage system the network-attached storage devices on the peripheral network form a global pool that we call the *Network Storage Pool* (NSP) that can be carved up into many *subpools.* This partitioning into subpools allows the system manager to configure separate subpools, each with different characteristics, including:

- number of disks (or disk arrays) in a subpool
- stripe unit size
- access attributes (such as client affinity for a particular subpool, contiguous blocks, etc.)
- performance attributes (such as meeting a strict bandwidth or latency limit).

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 will allow some user control of file placement on physical storage devices based on the appropriate attributes required such as bandwidth, capacity, or redundancy.

A Brief Description of File Systems

File systems maintain persistent user and system data on storage devices such as disk drives. They maintain files by keeping pointers to file data blocks which are fixed size (typically 1-4Kbytes) and an integer multiple of the storage device block size. Some file systems such as UFS [20] have semantics that allow applications to access data in units smaller than the file system block size which generally requires buffering in main memory.

A file is an operating system abstraction that hides from the user the details of how the data is mapped to physical storage devices. Typically, an application reads and writes data to and from a file as if the data were a linear sequence of randomly-accessible bytes (or blocks or possibly records) — but the data may, and often is, scattered throughout the blocks of the physical device.

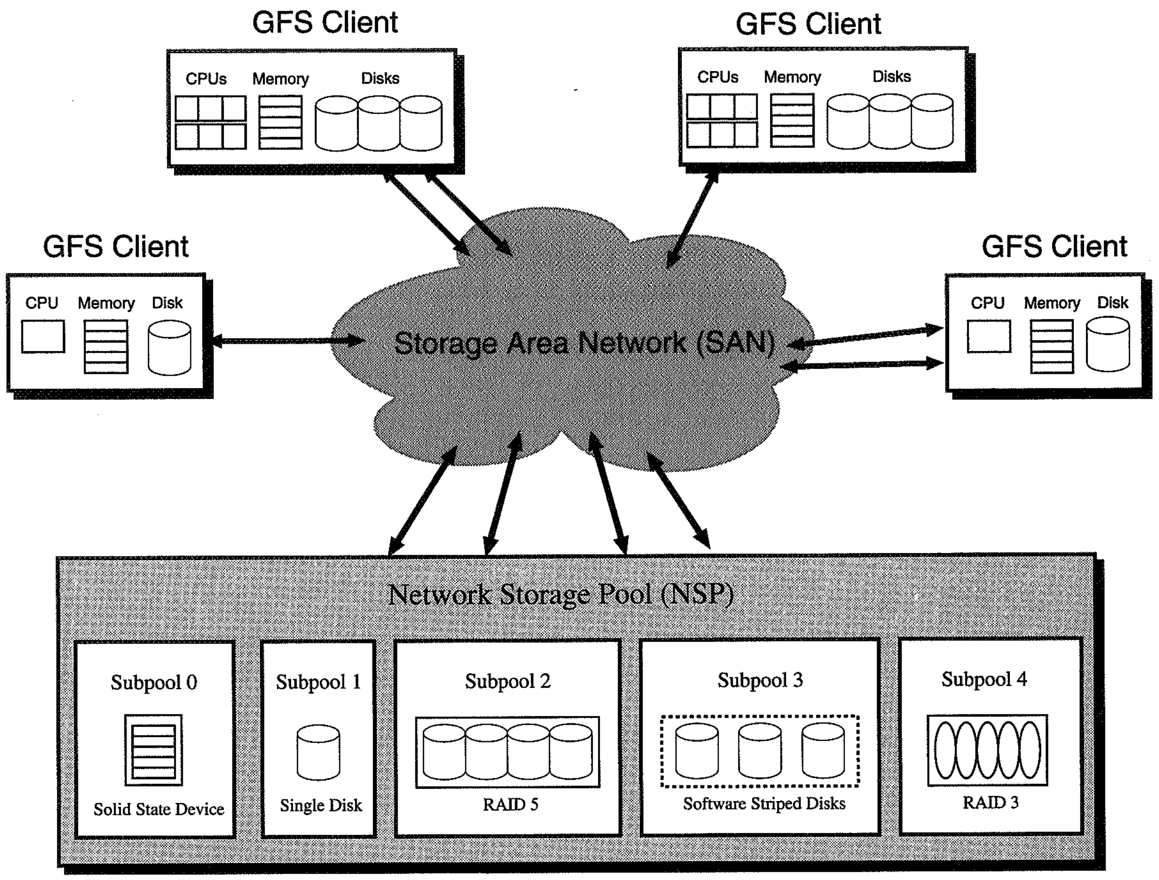


Figure 1: *Global File System Distributed Environment*

A directory is a type of file which contains groups of other files and directories. Directories are hierarchical, yielding a tree-structured name space containing all files and directories for a given file system. Associated with each file is a unique number or handle called an *inode number* in UNIX file systems. Each inode has a corresponding *dinode* located on the physical storage device which maintains information about the file owner, permissions, number of links, access times, size and *pointers* to the location of the file's data blocks on the physical storage devices. An *inode* is the in-memory data structure corresponding to the dinode.

The file system stores the dinodes, known as *metadata,* along with the actual file data. In addition to dinodes, the file system maintains *free lists* of data blocks not allocated to files. In modern UNIX file systems, free lists are implemented as *bitmap tables* where each bit represents a file system block; a bit that is set signifies that the corresponding block is already allocated. A file system maintains a single *superblock* which contains the layout of the file system, maintains counts of free dinodes and free data blocks, and stores mount information such as mount device and access privileges.

GFS Implementation: Metadata and Data

The GFS structure and internal algorithms differ from traditional file systems, emphasizing sharing and connectivity rather than caching. Unlike local file systems, GFS distributes file system resources across the entire storage subsystem, allowing simultaneous access from multiple machines. GFS also attempts to place specific data types, either metadata or data, on subpools with suitable performance characteristics.

The network storage pool (NSP) shown in Figure 1 supports the abstraction of a single unified storage address space for GFS clients. The NSP is implemented in a device driver layer on top of the basic SCSI device and Fibre Channel drivers. This driver translates from the logical address space of the file system to the address space of each device. Subpools divide NSPs into groups of similar device types which inherent the physical attributes of the underlying devices and network connections.

GFS, unlike typical file systems, distributes its metadata throughout the network storage pool rather than concentrating it all into a single superblock. As shown in Figure 2, multiple *resource groups* are used to partition metadata, including data and 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 can be thought of as partitioning the file system into distinct sets of files and available data and metadata blocks.

Resource groups are similar to the *allocation groups* (AGs) found in SGI's XFS file system [19]. Like resource groups, allocation 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 as shown in Figure 3. This information includes 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 contains a static index of the resource groups which describes the location of each resource group and other configuration information.

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.

Consider a file system block size of 16 KB and assume the dinode header information requires 128 bytes. Without stuffing, a 1-byte file requires a total of 32 KB and at least 2 disk transfers to read the dinode and data block. With stuffing, a 1-byte file only requires 16 KB and one read request. The file can grow to 16 KB minus 128 bytes, or 16,266 bytes, before GFS unstuffs the dinode.

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.

Using a flat pointer tree structure as shown in Figure 4, the maximum file size for GFS assuming 8K file system blocks and 8 byte pointer addresses is about a factor of 1000 greater than those attainable with UFS. (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 [19]. The current structure of the GFS metadata is an implementation choice and these alternatives are worth exploration in future research.

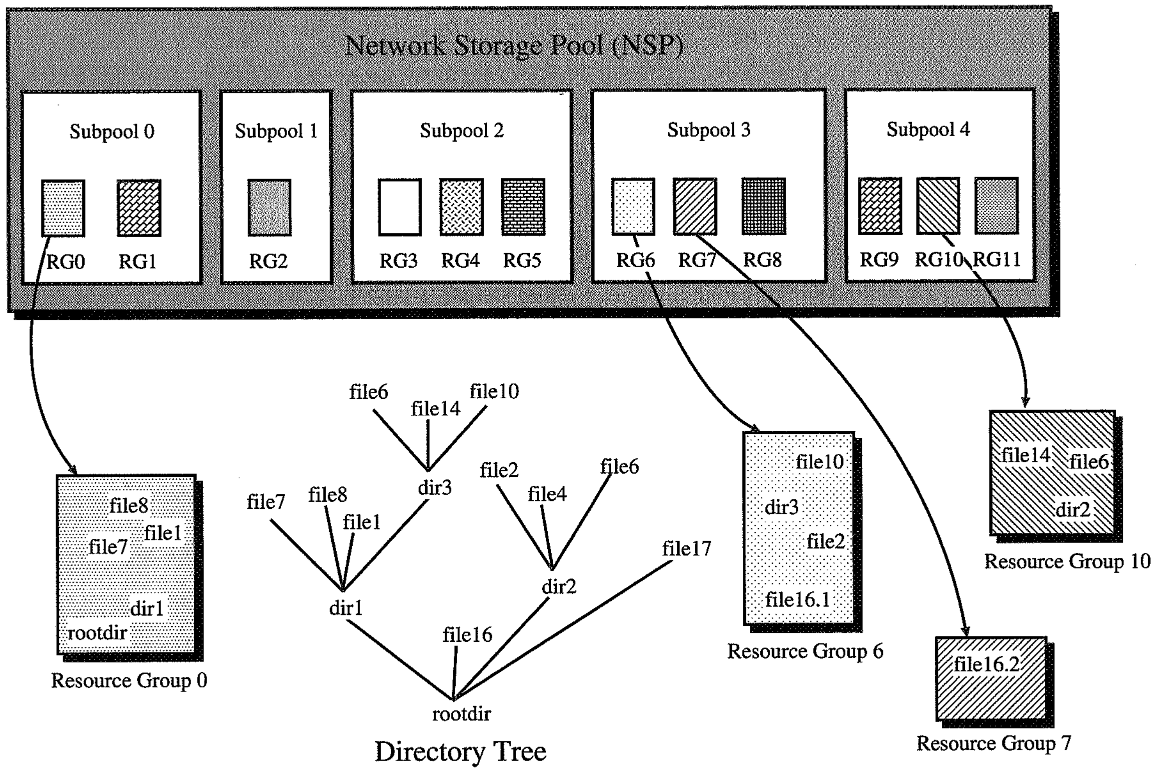


Figure 2: *Files Mapped onto Resource Groups and Subpools.*

GFS Implementation: Consistency and Caching

As shown previously, GFS exploits network-attached storage in a *Network Storage Pool* to create a shared disk memory that behaves much like the shared main memory found in multiprocessors today: each processor sees the same file system name space and has equal access to the shared disk memory. And like a multiprocessor system which must permit controlled, synchronized access to shared memory [14], a key design issue is the ability of each client to access the file system that may span many shared storage devices without destructively interfering with other clients accessing the same file system.

Multiple client accesses to shared devices must be synchronized: the three primary alternatives are disk-based [18], device-based [9] or client-based [7], [8], [3] synchronization. In the disk-based approach used by GFS, locks resident on the drive are used by multiple clients to safely manipulate file system metadata. A device-based approach is similar except that the locks are found on a shared device independent of the disk drives. Finally, client-based synchronization distributes the locking function between the clients: messages are exchanged to lock a particular file or resource. Some form of lock table (either centralized or distributed) is used to maintain the current state of shared, potentially locked resources.

GFS uses atomic read-modify-write operations on disk-resident metadata to maintain file system consistency. These operations guarantee that at a fixed point in time data exists in at most three places: the disk media, the disk on-board cache, and in client memory. The disk maintains consistency between its media and cache while the GFS file system uses disk-based locks to implement atomic operations on metadata to maintain consistency between multiple clients and disk devices. Atomic operations on shared data are performed by acquiring exclusive access to the data via a lock, reading the data from memory or storage, modifying the data, writing the data back, and releasing the exclusive access by giving up the lock.

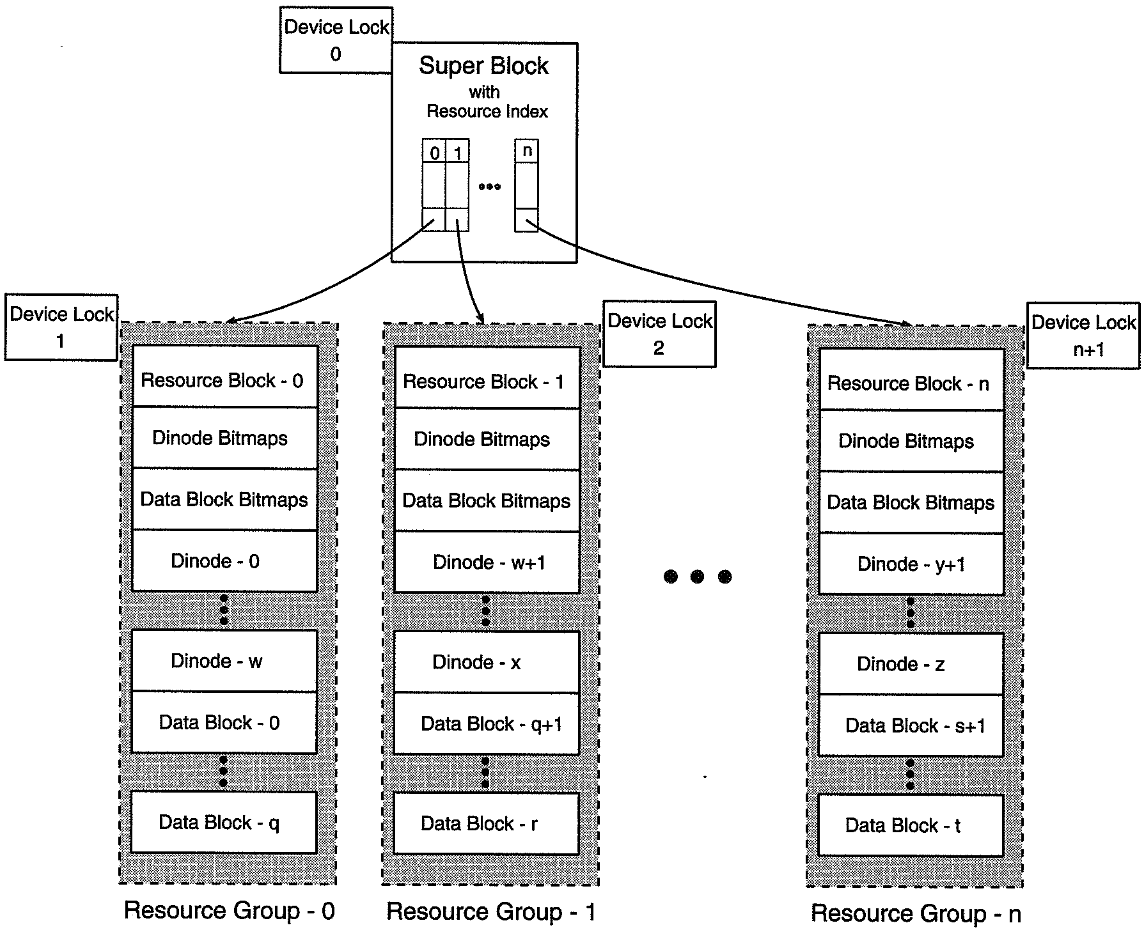


Figure 3: *GFS Metadata Structure*

Our initial GFS implementation [15] used the basic SCSI command for locking devices known as RESERVE/RELEASE [2]. However, this command works at the granularity of an entire disk which makes it impossible to have more than one lock active for a single device, severely restricting parallelism and hence scalability. The SCSI standard does allow reservations on "extents" (contiguous logical blocks) but since this command is not mandatory very few drives actually support it. In addition, in discussions with our industrial partners Seagate and Ciprico we found that this particular command would add significant overhead to other commands, making it less appealing to implement. In response, we worked jointly with Seagate and Ciprico to develop a new SCSI command called DLOCK [15] which provides a fast, efficient locking primitive — a *device lock* — ideal for our GFS implementation.

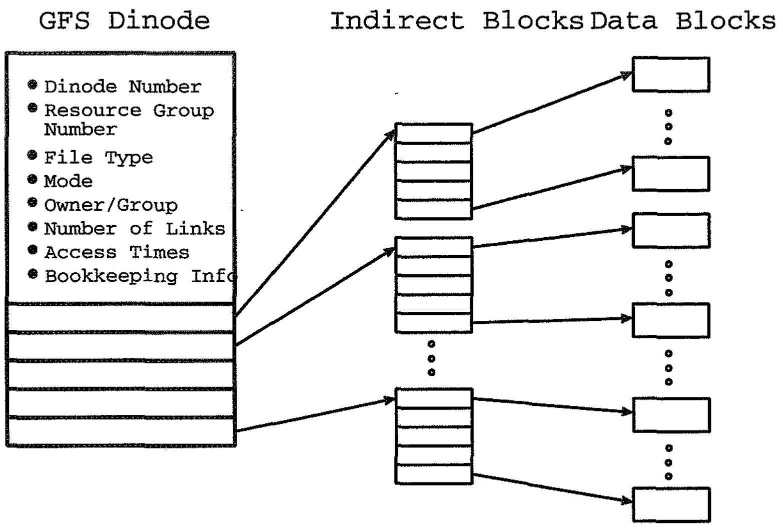


Figure 4: *GFS Dinode Internal Structure*

Device locks (*DLOCKs*) are implemented as an array of state bytes in volatile storage on each device. Each lock is referenced by number in the SCSI command: the state of each lock is described by one bit. If the bit is set to 1, the lock has been acquired and is owned by an initiator (client). If the bit is 0, the lock is available to be acquired by any initiator. The DLOCK command action *test and set* first determines if the lock value is one. If the value is 1, then the command returns with status indicating the lock has already been acquired. If the value is 0, DLOCK sets the lock to 1 and returns GOOD status to the initiator. The DLOCK command *clear* simply sets the lock bit to 0. A *test* operation is provided to read (but not set) the state of the lock.

It is important to realize that the device does not itself understand how the lock relates to the data on its own media or on other devices: that is the task of the file system or database. Hence, the granularity at which the lock is applied is up to the system software. In Figure 3, we can see that the current GFS implementation associated device locks with resource groups, dinodes, and the super block. DLOCKs are general enough to support mutual exclusion on just about any resource on the network. These locks are only held temporarily during metadata updates so it is important that the DLOCK commands be executed quickly.

The GFS metadata strategy is to cache metadata both on the disk drive caches and client memory. If exploited properly, the solid-state buffer memories on the disk drives provide a convenient cache structure that can be shared by multiple GFS clients. GFS clients can also directly cache some file system metadata that is read-only. This approach uses only a small amount of client memory for file caching and removes the burden other distributed file systems like AFS and DFS [20] have in supporting large, sophisticated file caches directly on clients. This plays directly to the new capabilities of disk drives: faster interfaces and fast access to local disk caches[3].

Reliability and availability are important to the Global File System since it is designed to support large numbers of disks and clients operating in parallel. Given high disk drive failure rates some form of redundancy is necessary at the drive level. This can be done with the appropriate RAID level applied to groups of drives to provide high availability. Client failures can be quite common: in traditional client-server systems a client failure implies loss of access to data on drives attached to the client. This problem is avoided in the GFS because drives are not attached to a single client but are instead accessed across a peripheral network. This network should in general be more reliable than either clients or devices and provides multiple, redundant paths between them[4].

But important issues such as lock contention, redundancy, fairness, and error recovery must be considered in the design. Our proposed DLOCK SCSI command provides for additional lock state known as the *logical clock* that lets clients determine when another client with a lock has failed, leaving the lock in a zombie-like state [15]. We have proposed a distributed recovery mechanism using these clocks in the event of client failure. This information can also be used to help prevent a slow client from being starved for access to a particular lock and in measuring device workloads to provide information to a file system load-balancing utility. Finally, this state information allows GFS clients to determine whether metadata cached on the client has been modified back on the device and is therefore stale. Hence, DLOCK provides a simple yet elegant command that integrates consistency, caching, and recovery in the GFS architecture[5].

3. The Current Global File System Implementation and its Performance

We have a GFS implementation developed on Silicon Graphic's IRIX operating system using the VFS/VNODE interface [20]. Our implementation is based upon the architecture described in section 2 and includes the GFS VNODE and VFS operations, a network storage pool driver, test scripts, performance measurement GUIs, and file system utilities (like *mkfs*) that together comprise nearly 40,000 lines of code. It is basically complete and further testing and device integration are proceeding as GFS becomes available to users in our laboratory and elsewhere so that we may study its performance under real application workloads. We are also adding more industrial partners to our efforts to more widely disseminate this new technology.

Though much of our initial testing and development have exploited parallel SCSI disk drives we are most interested in GFS performance and scalability on a true network-attached storage interface like Fibre Channel. A detailed report on our initial performance results can be found in [16]. Here we summarize these results; later we discuss the implications of this work relative to other research in this area and our proposed research objectives.

The test configuration we used is show in Figure 5. We were interested in how GFS performance would scale as both clients and disk arrays were added to this four-client, four-array configuration. We tested for several parameters including file transfer time and bandwidth using a range of file sizes. Some tests measured the aggregate transfer rate across all clients working together while others measure the throughput of transferring files of varying sizes across the whole system. The tests included measurements from eight Seagate Barracuda 9 Fibre Channel disk drives and 4 Ciprico 7000 series RAID-3 Fibre Channel disk arrays connected via a 16-port Brocade Silkworm Fibre Channel switch to 4 SGI Challenge computers.

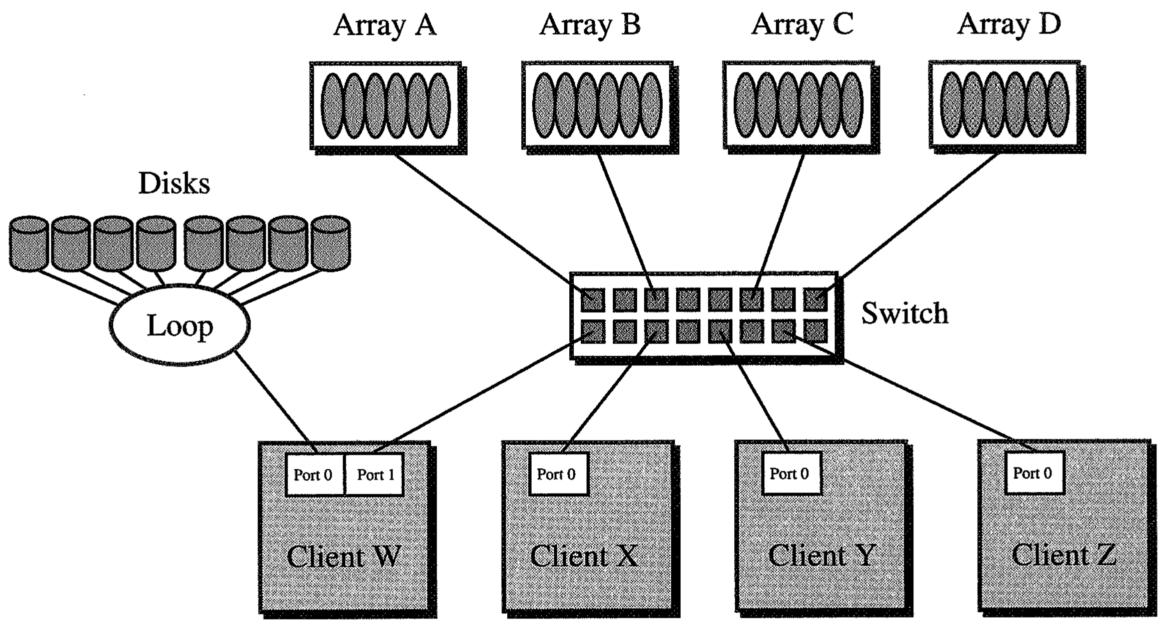


Figure 5: *GFS Hardware Configuration for Tests Performed May 1997.*

Separate measurements were performed on device lock performance which showed that on both the Seagate and Ciprico products locking required about 1.3 milliseconds (ms) while unlocking required 1.0 ms. GFS read performance relative to raw device performance is given in Table 1. It can be seen that GFS performance relative to the underlying hardware is quite good for larger files but lags for smaller files. Small file performance lags in GFS because the high metadata access overheads are not amortized by the long transfer times required by larger files. In addition, current disk device caches are not tuned well to cache metadata; these caches are focused mainly on read-ahead and write-behind of sequential data and are used to essentially cache several tracks worth of data. Hence, further joint work with our industrial partners will be necessary to develop and implement appropriate caching strategies on devices and integrate into GFS the necessary cache control features to exploit these device caches.

Sizes		Single Disk			8-Wide disks			Disk Array		
File Size	Request (MB)	Raw (MB/s)	GFS (MB/s)	Ratio (%)	Raw (MB/s)	GFS (MB/s)	Ratio (%)	Raw (MB/s)	GFS (MB/s)	Ratio (%)
4	2	10.5[1]	8.22	78.6	60.9[1]	24.8	40.7	74.8	33.5	44.8
4	4	10.4[1]	8.38	80.6	58.5[1]	25.5	43.6	76.9[1]	33.8	44.0
16	8	10.5	9.81	93.1	59.1	43.2	73.1	78.2	61.1	78.1
16	16	10.4	9.9	95.2	58.5	45.4	77.6	79.0	61.0	77.2
128	8				64.8[2]	49.9	77.0	81.4[2]	68.4	84.1
128	16				64.9[2]	54.0	83.2	82.0[2]	73.1	89.1
256	8							81.4	70.1	86.1
256	16							82.0	74.8	91.2

[1]Estimated from 16 MB sequential performance tests.
[2]Estimated from 256 MB sequential performance tests.

Table 1. *Raw Disk Performance versus GFS Read Performance*

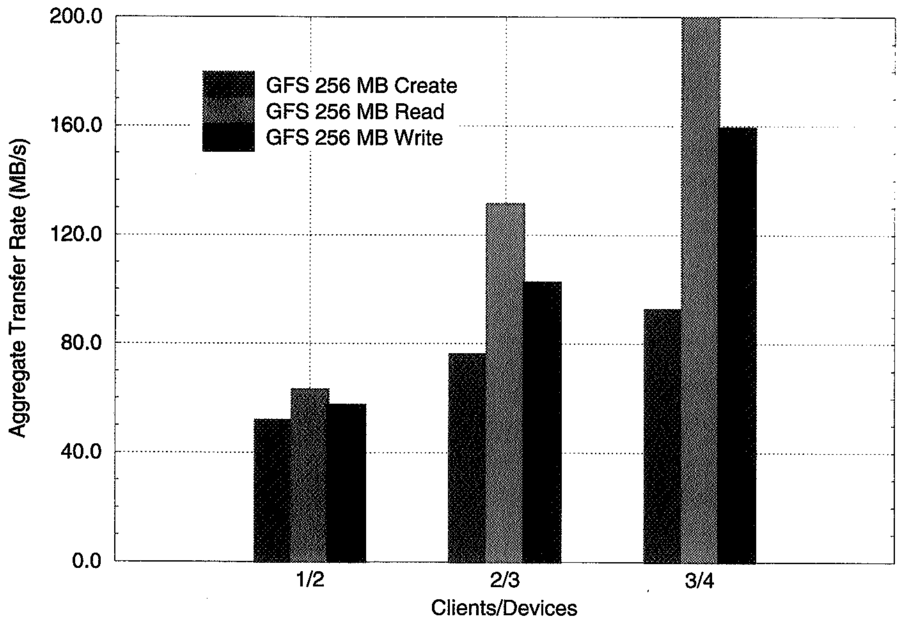


Figure 6: *Aggregate Transfer Rate of 256 Mbyte Files with a Root Directory Device.*

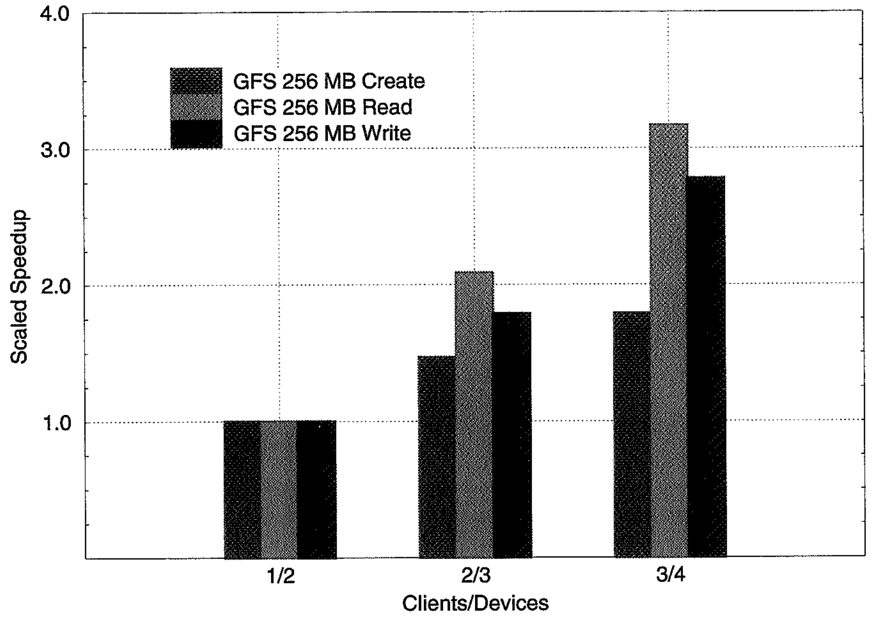


Figure 7: *Scaled Speedup of 256 Mbyte Files with a Root Directory Device.*

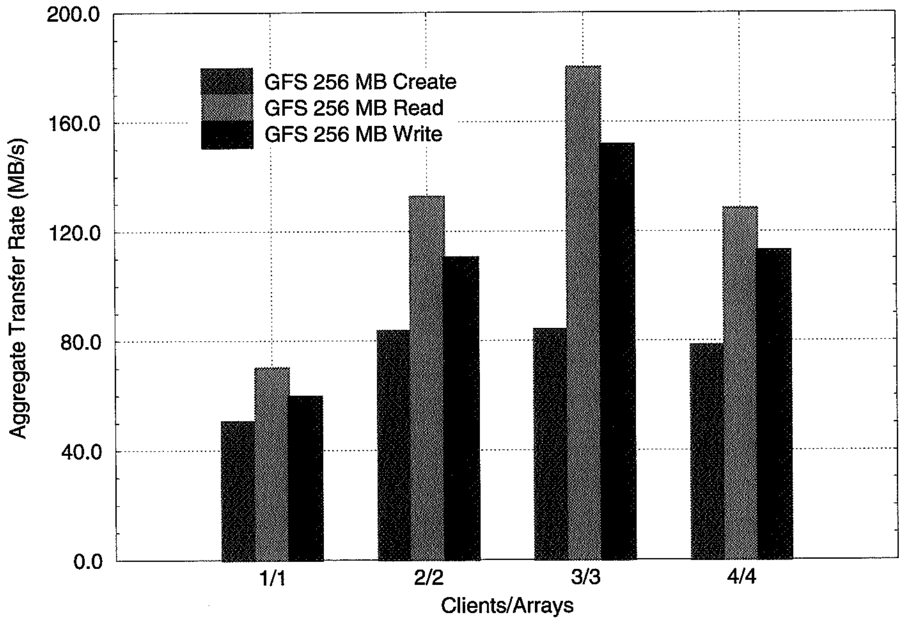


Figure 8: *Aggregate Transfer Rate of 256 Mbyte Files.*

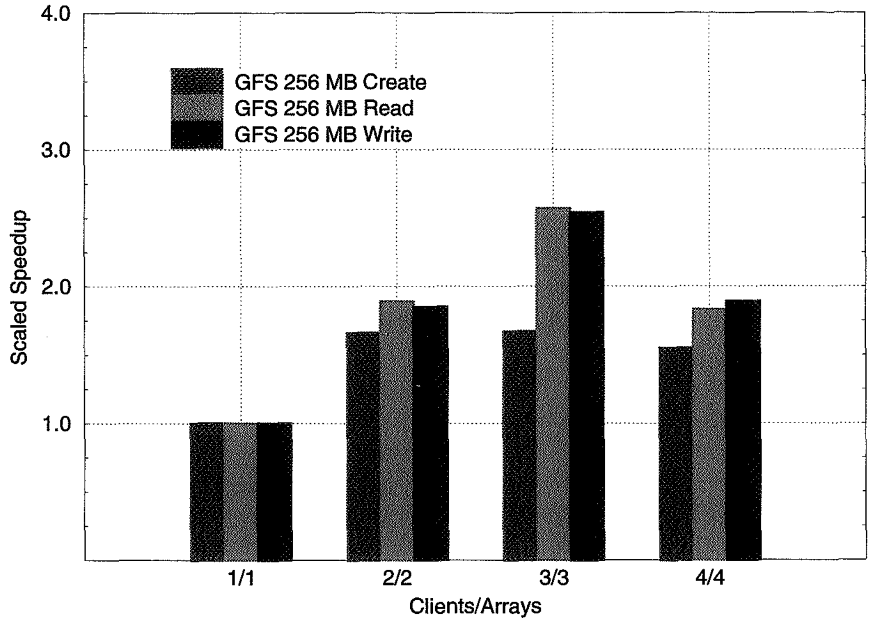


Figure 9: *Scaled Speedup for 256 Mbyte File.*

The scaling observed for aggregate transfer rates across the four machines and arrays are given in two sets of figures. In Figures 6 and 7 aggregate bandwidth and scaled speedup across from 1 to 3 disk arrays and machines is given for the case where a dedicated device is used for holding device locks and the root directory metadata. Good scaling is seen for this case with large files sizes and minimal contention between device locks, directory accesses, and access to actual file data. This contrasts with Figures 8 and 9 where these same metrics were considered for the case where a dedicated root directory and lock device was not used so that contention between accesses for file data, directories and device locks occurred on the same array. It can be seen that performance in fact decreases with the addition of the fourth array and machine due to this contention.

Fortunately this poor scaling is more the result of implementation decisions rather than the fundamental GFS architecture. In fact in this test case the resource group layout was not optimized for this configuration resulting in more contention than otherwise would have been necessary.

4. Conclusions

We believe that our results show that a new approach to cluster file system design is appropriate in the context of network attached storage. Our performance results are unique compared to earlier cluster file system studies in that our results are derived from a file system designed from scratch to exploit the data and device sharing potential of network attached storage. Future work will include reducing contention for shared resources such as the root directory, performance evaluation of GFS on larger configurations with up to 16 clients and 8 to 12 disk arrays, and more aggressive use of device caching for shared file system metadata. In addition, we are implementing a scheme for distributed recovery for device and client failures which integrates metadata consistency checking with file lock removals for failed nodes.

References

[1] D. Seachrist, R. Kay, and A. Gallant, "Wolfpack Howls Its Arrival," BYTE Magazine, pp. 126-130, vol. 22, no. 8, August 1997.

[2] D. Deming. **The SCSI Tutor**. Saratoga, CA: ENDL Publishing, 1994.

[3] M. Devarakonda, A. Mohindra, J. Simoneaux, W. Tetzlaff, "Evaluation of Design Alternatives for a Cluster File System," *1995 USENIX Technical Conference*, January 1995.

[4] G. Gibson *et al.*, "File Serving Scaling with Network-Attached Secure Disks," *Proceedings of the ACM Int. Conf. on Measurements and Modeling of Computer Systems (SIGMETRICs '97),* Seattle, WA, June 15-18, 1997.

[5] S. Koegler, "SPANStor Adds on Network Storage with Ease and Convenience," *Network Computing,* November 1, 1995.

[6] R. Katz, G. Gibson, and D. Patterson, "Disk System Architectures for High Performance Computing," *Proceedings of the IEEE,* vol. 77, pp.1842-1858, 1989.

[7] N. Kronenberg, H. Levy, W. Strecker, "VAXClusters: A Closely-coupled Distributed System," *ACM Transactions on Computer Systems,* vol. 4, no. 3, pp. 130-146, May 1986.

[8] I. Lloyd, "The Oracle Parallel Server Architecture," *Proceedings of Supercomputing-Europe 92,* pp. 5-7, 1992.

[9] K. Matthews, "Implementing a Shared File System on a HiPPI Disk Array," *Fourtheenth IEEE Symposium on Mass Storage Systems,* pp. 77-88, September 1995.

[10] R. Meter, "A Brief Survey on Current Work on Network Attached Peripherals," *ACM Operating Systems Review,* pp. 63-70, January 1996.

[11] G. Pfister, **In Search of Clusters**. Upper Saddle River,NJ: Prentice-Hall, 1995.

[12] T. Ruwart and M. O'Keefe, "A 500 Megabyte/Second Disk Array," *Fourth Nasa/Goddard Conference on Mass Storage Systems and Technologies,* College Park, Maryland, March 1995.

[13] M. Sachs, A. Leff, and D. Sevigny, "LAN and I/O Convergence: A Survey of the Issues," *IEEE Computer,* vol. 27, no. 12, pp. 24-33, December 1994.

[14] C. Schimmel, **UNIX Systems for Modern Architectures.** Addison-Wesley: Reading, MA, 1995.

[15] S. Soltis, *The Design and Implementation of a Distributed File System Based on Shared Network Storage.* Ph.D. Dissertation, Department of Electrical and Computer Engineering, University of Minnesota, Minneapolis, MN, August 1997.

[16] S. Soltis, G. Erickson, K. Preslan, T. Ruwart, M. O'Keefe, *The Global File System: A File System for Shared Disk Storage,* submitted to the *IEEE Transactions on Parallel and Distributed Systems,* October 1997.

[17] S. Soltis, T. Ruwart, and M. O'Keefe, "The Global File System," *Fifth NASA Goddard Conference on Mass Storage Systems and Technologies,* College Park, MD, September 1996.

[18] S. Soltis, M. O'Keefe, T. Ruwart, and B. Gribstad, *SCSI Device Locks,* technical report, Department of Electrical Engineering, University of Minnesota, April 1996.

[19] A. Sweeney, D. Doucette, W. Hu, C. Anderson, M. Nishimoto, G. Peck, "Scalability in the XFS File System," *1996 USENIX Technical Conference*, January 1996.

[20] U. Vahalia, **UNIX Internals: The New Frontiers.** Prentice-Hall, Upper Saddle River, NJ, 1996.

---

[1] This work was supported by the Office of Naval Research under grant no. N00014-94-1-0846, by the National Science Foundation under grant no. CDA-9414015 and no. ASC-9523480, by NASA through grant no. NAG2-1151 and by equipment grants from Seagate Technology, Brocade Communications, Silicon Graphics Inc. and Ciprico. Contact the authors at okeefe@lcse.umn.edu.

[2] Leslie Lamport, a well-known researcher, is quoted as saying a "distributed system is one where the failure of some computer I've never heard of can keep me from getting my work done."

[3] Solid-state disk caches were not widely used until about 1990. Most operating systems and file systems were developed prior to this time, when no solid-state disk caches were available.

[4] Much like earlier IBM mainframe channel architectures, future Fibre Channel drives have multiple ports which could be used to improve availability.

[5] DLOCK has been implemented by Seagate Technology and Ciprico in their disk products and we report our initial performance results in [SoE97]. We are in the process of filing a patent application for the DLOCK invention.

HPSS/DFS: Integration of a Distributed File System with a Mass Storage System

Rajesh Agarwalla, Madhu Chetuparambil, Craig Everhart, T. N. Niranjan
Transarc Corporation
The Gulf Tower
707 Grant Street
Pittsburgh, Pennsylvania 15219
{rajesh, madhuc, cfe, niranjan}@transarc.com
Tel: +1-412-338-4467
Fax: +1-412-338-4404

Rena Haynes, Hilary Jones
Sandia National Laboratories
P.O. Box 5800, MS0807
Albuquerque, New Mexico 87185
Sandia National Laboratories
P.O. Box 969, MS9011
Livermore, California 94551
rahayne@sandia.gov, hilary@ca.sandia.gov
Tel: +1-505-844-9149,+1-510-294-2892
Fax: +1-505-844-2067,+1-505-294-1225

Donna Mecozzi
Lawrence Livermore National Laboratory
7000 East Avenue
P.O. Box 808
Livermore, California 94551
dmecozzi@llnl.gov
Tel: +1-510-442-6020
Fax: +1-510-423-8715

Bart Parliman
Los Alamos National Laboratory
MS-B269, CIC-11
Los Alamos, New Mexico 87545
bartp@lanl.gov
Tel: +1-505-667-8410
Fax: +1-505-665-6333

Jean E. Pehkonen
IBM Software Group, Software Solutions Division
11400 Burnet Road
Internal Zip 9151
Austin, Texas 78758
jean@austin.ibm.com
Tel: +1-512-838-3457
Fax: +1-512-838-0156

Richard Ruef, Benny Wilbanks
IBM Global Government Industries
1810 Space Park Drive
Houston, Texas 77058
ruef1@llnl.gov, benny@clearlake.ibm.com
Tel: +1-510-422-0256, +1-281-335-4040
Fax: +1-281-335-4231

Vicky White
Oak Ridge National Laboratory
P.O. Box 2008
Oak Ridge, Tennessee 37831-6203
vyw@ornl.gov
Tel: +1-423-574-7999
Fax: +1-423-546-9150

Abstract: To provide cost-effective access to data in storage-intensive computing environments, mass storage systems must be integrated with underlying file systems. Previous projects have presented specialized client interfaces or have integrated mass storage with local file systems. Our approach is to integrate the distributed file system DFS, with support for DMAPI, and HPSS, a high performance mass storage system. The result is a mass storage system that includes a fully distributed file system, data transfer rates that scale to the gigabyte/sec range, and archival storage, scalable to multiple petabytes.

Introduction

Enterprises with large or networked computing environments often employ distributed file systems, providing several advantages: data consistency guarantees, location transparency, uniform name space, ease of administration, performance, and cost.

In recent years, the need to store high-resolution images, scientific data, etc., has created a serious imbalance between I/O and storage system performance and functionality[1]. To solve this imbalance, the performance and capacity of current mass storage systems must improve by orders of magnitude. Coupling mass storage systems with underlying distributed file systems, providing the best features of each and providing a seamless view of the file system, is desirable. Three emerging technologies have made this possible: The Distributed File System (DFS™)[2], the High Performance Storage System (HPSS)[3], and the Data Management Application Interface (DMAPI)[4].

In this paper, we discuss the work of the Transarc Corporation and the HPSS Collaboration to integrate DFS with HPSS using the DMAPI. We discuss past approaches to mass storage integration, the features and requirements of our system, and the overall architecture. We also describe the extensions needed in DFS, HPSS and DMAPI to support this integration, HPSS/DFS configuration options, with their corresponding performance data, and future work.

Background

Integrating mass storage with file systems to provide hierarchical storage management (HSM) has typically followed one of two philosophies:

- The file system back-end approach uses mass storage to extend the storage on the local platform.
- The mass storage file system approach implements a file system within the mass storage application.

File system back-ended mass storage allows seamless integration of mass storage with the file system platform. Leveraging the functionality of the platform simplifies HSM functionality but limits file and storage features to those supported by the local file system. These implementations require specialized software in the platform operating system, and network file services supported by the platform must be used for network access to files. Implementations based on this scheme include Data Migration Facility (DMF)[5], AMASS™ [6], and E-MASS™ [7].

Mass storage file system approaches typically present their file systems through specialized client interfaces. These approaches do not require operating system kernel modifications, and additional functionality is implemented directly in the HSM. This approach offers flexibility in file and storage features, such as support for network-attached peripherals, third-party transfers, and parallel I/O. However, specialized client code may be required to take advantage of any extended features. Implementations based on this scheme include Common File System (CFS)[8], UniTree™ [9], and HPSS.

Features and Requirements

The High Performance Storage System (HPSS) was developed by a collaboration of IBM and four Department of Energy laboratories: The Lawrence Livermore National Laboratory, the Los Alamos National Laboratory, Oak Ridge National Laboratory, and Sandia National Laboratories. A goal of the project was to provide mass storage capacity scalable to multiple petabytes, using standard languages and communications without operating system kernel modifications.

HPSS supports a high-speed, parallel interface[10] to achieve data transfer rates greater than 1 GB/sec for a single file. It supports third party I/O transfers[11] which allows data to move directly from the storage media to the client without intermediate server processing. HPSS supports secure access to files and directories through DCE security features, such as Access Control Lists (ACLs) and authenticated Remote Procedure Calls (RPCs). File sizes up to 2^{64} bytes are supported.

A secure, platform-independent global name space is desired for several HPSS deployment sites. DFS emerged as the primary candidate to provide this service. It has since become a requirement for HPSS to support DFS. However, the vast majority of these HPSS sites still want the high-speed I/O performance for very large files, which is the hallmark of HPSS.

DFS, developed by Transarc Corporation through the Open Software Foundation (now called The Open Group), is a highly scalable distributed file system. DFS provides a uniform view of file data to all users through *a global name space*. In addition to directory hierarchies, DFS supports logical collections of files called filesets. A fileset is a directory subtree, administered as a unit, that can be mounted in the global name space. Multiple filesets may reside on an *aggregate,* which is analogous to a disk partition. Filesets may be moved between aggregates, either on the same or different servers to achieve load balancing. DFS also supports ACLs on both directories and files to allow granting different permissions to many users and groups accessing the object. DFS uses the DCE concept of cells, and allows data access and authorization between clients and servers in different cells.

DFS uses the Episode™ (DCE LFS)[12] physical file system. This log-based file provides features like filesets, cloning (on-line backup through a fast snapshot capture mechanism), ACLs, etc., that DFS utilizes. DFS can also use other native file systems, such as UFS.

Coupling DFS with HPSS reduces the cost of storage by allowing data to be exported to media other than the disks directly attached to the DFS file server and supports files greater than the size of the disk space managed by DFS. However, the integration had to meet requirements from both Transarc and the HPSS Collaboration:

- A standard interface for coupling DFS with HPSS must be employed.
- Transparent, automatic archiving and caching of data must be provided.
- Both DFS and HPSS must support partially-resident file data.
- Changes made to a file or directory through the DFS interface must be visible through HPSS interfaces and vice versa.
- HPSS high speed file transfer mechanisms must continue to provide single file transfer rates over 1 GB/sec.

- The implementation must not effect the data transfer speeds of any data resident on disks managed by DFS, and file accesses made through DFS must not impact HPSS performance.

Fortunately, standards for interfacing file systems and data management applications (DMAP) are emerging. In the mid-nineties, the Data Management Interfaces Group (DMIG), with representatives from several file system and mass storage vendors, defined a data management interface (DMAPI) for UNIX based file systems. DMAPI, now called XDSM [13], has been adopted as a standard by the Open Group. It defines an API that can be implemented by file system vendors and used by a DMAP. The primary advantage of this standard is that mass storage vendors implementing a DMAP need not modify the OS kernel, thus enhancing portability and the ease of developing such applications.

DMAPI is a low-level interface to the physical file system. It provides the DMAP with the ability to store important attribute information with a file and allows for the generation of notifications to the DMAP on occurrence of various file system operations. DMAPI enables the DMAP to control disk storage by allowing the DMAP to move disk-resident file data to tertiary storage systems and vice-versa.

Configuration Options

A straightforward design approach satisfying both DFS and HPSS requirements was to use DMAPI to integrate DFS with HPSS. As noted, a primary objective of this effort was to provide the distributed services of DFS and the high-speed I/O performance of HPSS. Because DFS and HPSS have separate customer bases, with a relatively small overlap, the cost of providing the high-speed I/O through DFS was deemed too high. Therefore, at the fileset level, two data management configuration options, corresponding to the HSM integration philosophies previously mentioned, are provided:

- HPSS is used strictly as an archive facility for DFS, making the integrated DFS/HPSS system a traditional HSM. Access to the name and data space is provided only through the DFS interfaces. Filesets managed with this option are called *archived* filesets.
- Consistency between HPSS and DFS name and data spaces is maintained. DFS data is archived to HPSS, but access to the name and data space is available through both DFS and HPSS interfaces. Updates made through the DFS interface are visible through the HPSS interface and vice versa. Thus, a user may access data through DFS, at standard DFS rates, and when high performance I/O rates are important, use the HPSS interface. Filesets managed with this option are called *mirrored* filesets.

Architectural Overview

The design for integrating HPSS with the Episode file system is shown in Figure 1. The details for each component are discussed in the following sections.

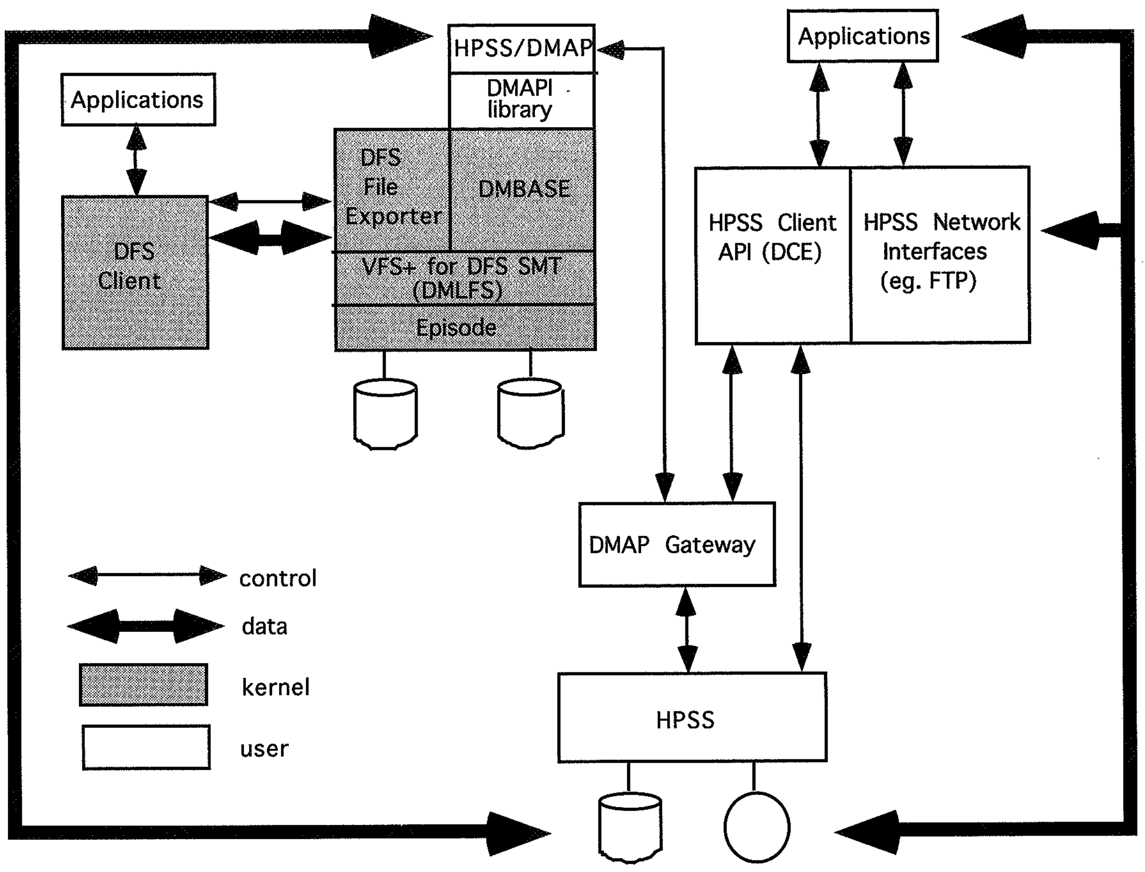


Figure 1. DFS/HPSS DMAPI Integration Architecture

DMAPI Implementation for DFS

The DMAPI implementation for DFS, called the DFS Storage Management Toolkit (DFS SMT), is fully compliant with the corresponding standard XDSM specification. In addition, it provides the following optional features; persistent opaque DM attributes, persistent event masks, persistent managed regions, non-blocking lock upgrades and the ability to scan for objects with a particular attribute.

The bulk of DFS SMT is implemented in the file server. DFS SMT has both user and kernel space components. A user space shared library implements all APIs in the DMAPI specification. A device driver provides communication between the user space component of DFS SMT and the kernel component.

The DFS SMT kernel component consists of two sub-components: a file system independent layer (DMBASE) and a file system dependent layer (DMLFS). DMBASE receives requests via the device driver, processes them and responds to the DMAP via the driver. It maintains DMAPI sessions, DMAPI tokens, event queues, and the registered disposition of events for various file systems. This layer is also responsible for receiving events from the DMLFS and dispatching them to the DMAP. For each DMAPI call, DMBASE is responsible for making appropriate calls to the managed file system via DMLFS.

DFS uses an enhanced virtual file system[14], VFS+, that is implemented for the DFS client and any physical file systems exported by DFS servers. VFS+ was augmented to fetch and store DM attributes, provide persistent managed regions and events, perform invisible I/O, purge data from files, and verify file residency.

DMLFS was designed as a layer in the stackable virtual file system model[15] and is an implementation of the VFS+ layer for DFS SMT. DMLFS operations are responsible for generating events. It consults with DMBASE to determine if any DMAP has registered to receive notification for events related to that particular operation and then generates the events. If the event is synchronous, it causes the file system operation to wait for a response to the event before proceeding. It also provides for interlocking between DFS SMT requests and file system calls.

To support persistent DM-related metadata, an extended attribute facility was provided in Episode. DM attributes, event masks, managed regions, and attribute change times (dtime values) are stored as extended attributes. This eliminates the need to store the attributes in a separate auxiliary file visible in the file name space. These extended attributes are treated as file metadata and changes to attributes are logged.

Episode was modified to support files that become sparse by punching holes that release disk resources. With a conventional sparse file, reading from a hole returns zeroes. To assume these same semantics for a hole that exists because the DMAP migrated the data to tertiary storage is incorrect. In this case, the DMAP must retrieve the data from tertiary storage. Hence, a facility is provided in Episode to mark file blocks as being off-line (in tertiary store) instead of as a hole. This allows the file server to handle partially resident files.

To prevent blocking file server (kernel) threads while waiting for a response to an event, a mechanism to notify the client to retry after a specified interval of time was added. The retry interval is exponentially backed off on each retry.

The DFS fileset dump and restore capability was augmented to include extended attributes and migrated regions. Migrated data is not recalled when a dump is taken, producing an "abbreviated" dump. Checks were added to DFS and retrofitted into prior releases to prevent restoring an abbreviated dump with a file system that may not be able to interpret its contents.

HPSS Implementation for DFS Support

The HPSS collaboration developed two new components, an HPSS Data Management Application (DMAP) and a DMAP Gateway. The HPSS Name Server, Bitfile Server, and Client API also required modifications.

HPSS/DMAP

HPSS Data Management Application (HPSS/DMAP) is responsible for initiating and coordinating file and data interactions between Episode and HPSS. It catches and processes desired name and data space events generated by Episode sent through the DFS SMT; migrates file data from Episode to HPSS; purges unneeded Episode file data after that data migrates to HPSS; and processes requests originating in HPSS interfaces that require either Episode name or data resources. HPSS/DMAP resides on the same machine as the disk(s) managed by the Episode file system.

For portability, HPSS/DMAP communicates with HPSS components via XDR over TCP sockets. The HPSS/DMAP also exists to isolate dependencies on DFS, Episode and DCE.

HPSS/DMAP registers to receive name and data space events. After catching a name space event involving a mirrored fileset, the appropriate requests are made to HPSS to keep the name spaces synchronized. For archived filesets, only create and destroy name space events are processed, but no HPSS resources are utilized. HPSS/DMAP receives data space events when a file is read, written, or truncated and the data region involved is registered with the DFS SMT. HPSS/DMAP is responsible for caching data to Episode that is not present; invalidating HPSS data, if necessary; and manipulating data regions to minimize the occurrence of events involving the same region. HPSS/DMAP can cache partial files.

HPSS/DMAP provides an interface for HPSS to request that an action occur in DFS to keep the HPSS and DFS name and data spaces synchronized. This mechanism is only used with mirrored filesets and occurs when an HPSS client requests to create, delete or modify a name space object. This interface is also used by HPSS to migrate or purge data from Episode disks. Before file data can be altered through HPSS interfaces, the data must first be purged from Episode disks. The capability to forward DCE credentials is provided, enabling HPSS/DMAP to make DFS requests on behalf of the user.

HPSS/DMAP migrates file data from Episode to HPSS. To free Episode disk resources, it also purges file data from Episode. Only the data that has been modified on Episode is migrated to HPSS, thus, a minor modification to a very large file will not result in re-migrating the entire file. Because policies for migrating and purging data are separately configurable, file data migrated from Episode is not automatically purged. In many cases, data for a given file is present in both Episode and HPSS and that data can be read from either interface without any data movement between Episode and HPSS.

DMAP Gateway

The DMAP Gateway is a conduit between HPSS/DMAP and HPSS. HPSS servers use DCE/RPCs, the Encina transaction manager, and Transarc's SFS for metadata managing HPSS objects. The DMAP Gateway encodes requests using XDR and sends them via sockets to HPSS/DMAP and translates XDR from the HPSS/DMAP to DCE/TRPC/Encina calls to the appropriate HPSS server. XDR and sockets were used for portability. When a connection between the HPSS/DMAP and Gateway is made, mutual authentication occurs.

The DMAP Gateway keeps track of the location in DFS and in HPSS of all the filesets it manages. For scalability, multiple DMAP Gateways are supported. However, a given DMAP Gateway will only operate on the filesets it manages. At this time, only filesets managed by DFS and HPSS are supported.

In addition to translating requests between HPSS/DMAP and HPSS servers, the DMAP Gateway keeps fileset request statistics and internal DMAP Gateway resource utilization. Heavily used filesets can be identified and management of these filesets could be distributed across multiple DMAP Gateways to improve performance.

Name Server

Support for filesets (disjoint directory trees) was added to the Name Server. HPSS supports three types of filesets:

- HPSS-only filesets contain objects accessible only through standard mechanisms available to HPSS Name Server clients.
- Archived filesets are directory structures with private files containing copies of file data from any DFS files that have migrated to HPSS. Path names to directory objects are generated by HPSS/DMAP based on configuration policies. At any given time, the data in these files may be out of date with the DFS data. Access to objects in these filesets is restricted to the DMAP Gateway and HPSS root.
- Mirrored filesets contain objects that are kept synchronized between HPSS and DFS. Objects in mirrored filesets have corresponding objects in DFS and HPSS with identical names and attributes. HPSS clients access these objects as before, but whenever a client's request alters the name space, the alteration occurs as a side effect of the change made in Episode. Such requests are forwarded by the Client API to the DMAP Gateway and then to HPSS/DMAP. HPSS/DMAP makes appropriate Episode system calls, causing events to be generated. Processing the event ultimately results in a request to the HPSS Name Server to perform the appropriate action. Requests processed this way include create, remove, symlink, hardlink, and permission changes. This mechanism is transparent to the HPSS client.

Bitfile Server

The Bitfile Server was modified to recognize when file data on HPSS was inconsistent with its DFS counterpart. Inconsistent data occurs whenever file data altered through the DFS interface has not yet migrated to HPSS. The Bitfile Server was modified to return a special error value whenever an HPSS I/O request involves file data on a mirrored fileset that is in this state. This error signals the Client API to request that the file data be migrated to the HPSS file before retrying the I/O request. Before the Client API reissues any write requests, it first requests that the Episode data be invalidated.

HPSS Client API

The HPSS client API was modified to handle fileset domains, recognize an object's fileset type, determine which server to contact to process a request for a given object, and cross junctions (fileset domains). Junctions provide the ability to link filesets managed by the same Name Server or by another HPSS Name Server. With this mechanism multiple HPSS name spaces can be joined to form a single name space.

For HPSS-only filesets, the Client API issues the same requests as in prior HPSS releases. Behavior of objects in this type of fileset is identical to current releases of HPSS, unless the client crosses a junction. Access to objects in this type of fileset is only allowed through non-DFS HPSS interfaces. If a system administrator decides to also permit access through DFS, then tools must be run to import the HPSS metadata into the Episode file system.

For archived filesets, client access through HPSS interfaces are prohibited. In general, file data in this type of fileset must be accessed through DFS, but in special cases, a system administrator may access the HPSS objects.

If an HPSS client request will alter the name space of a mirrored fileset, the Client API forwards the request to the DMAP Gateway. The Gateway sends the appropriate request to HPSS/DMAP which then requests Episode to make the modification. When Episode processes the request, a DMAPI event is generated and the HPSS/DMAP event handler makes the necessary HPSS calls to keep the name spaces synchronized. All name space modifications in mirrored filesets are the result of DMAPI events generated either by DFS calls, or by Client API calls forwarded to the HPSS/DMAP.

The Client API was modified to provide a consistent view of file data between DFS and HPSS interfaces for mirrored filesets. The Client API recognizes the new error returned by the Bitfile Server when data is inaccessible through HPSS because of DFS activity. When this error occurs, the Client API initiates steps to migrate and purge Episode data so the HPSS and Episode data remain synchronized.

DMAPI Extensions

To support mirrored filesets our design requires a superset of the DMAPI standard. DFS SMT was extended to support synchronous post events generated after Episode completes a name space modification but before the original user request completes. These events are necessary to keep HPSS and DFS name spaces synchronized. Delivery of synchronous post events is guaranteed to be fast. Also, to support mirrored filesets, the ability to register and generate events for permission changes to a file or directory were added. These events occur whenever the owner, group, mode, or ACL is changed.

Unique issues are raised because DFS supports filesets and aggregates, while the DMAPI was designed for traditional file systems. These issues are addressed by the DFS SMT. Specifically, to handle fileset destruction additional name space events were defined. Mounting and unmounting aggregates is treated by DMAPI as a single event. The actual implementation for DFS SMT is a multi-step process, with complex recovery logic, because events must be generated and processed for each fileset on the aggregate.

Additional fields were added to DMAPI structures. A new field was added to the events structure to associate paired events (pre and post events). A new field was added to the region structure to store user defined opaque data. Two fields were added to the statistics structure that identify the presence of ACLs associated with a file and to store a fileset identifier.

Additional DMAPI management interfaces were added to handle ACLs, DCE security authentication information, to enumerate information about filesets, and to determine the handle for a named file in a known directory.

Examples

To illustrate the interaction between the system components, an overview of creating and reading a file follows.

DFS Create Example

When a client requests to create a file through DFS, DFS SMT generates a sequence of DMAPI create events. HPSS/DMAP catches these events and if the fileset type is mirrored it issues a request to the DMAP Gateway to create the HPSS file. The DMAP Gateway issues a request to the Name Sever and Bitfile Server through the Client API to create a name space object and a bitfile. After these HPSS resources have been created, the DMAP Gateway responds to HPSS/DMAP, which then responds to the DMAPI create events. Finally, the DFS file resources are allocated and after that completes, the DFS client's create request completes.

When a client requests to create a file in an archived fileset, DFS SMT still generates create events. However, HPSS/DMAP does not immediately create the HPSS file. It simply marks the file as a candidate for migration and responds to the create event. HPSS resources will be allocated when the file is migrated to HPSS. Since HPSS resource allocation is delayed, the DFS client's request completes only slightly slower than a create request in a DFS fileset not integrated with HPSS.

HPSS Read Example

This example applies only to mirrored filesets. When an HPSS client requests to read data from a file, the request is issued through the Client API to the Bitfile Server. If the Bitfile Server has the desired data and that data is consistent with the data mirrored on DFS, the Bitfile Server proceeds to read the data. However, if the data is inconsistent with DFS, the Bitfile Server returns an error to the Client API.

Upon receiving an inconsistent data error, the Client API requests the DMAP Gateway managing the fileset to migrate the data from Episode to HPSS. The DMAP Gateway contacts the HPSS/DMAP, which uses DFS SMT to migrate the data from Episode to HPSS, and then marks the data as synchronized. HPSS/DMAP responds to the DMAP Gateway, which responds to the Client API. At that point, the desired data has migrated to HPSS and the Client API retries the read request.

Performance Comparisons

Performance data was gathered from tests run on two platforms; an AIX 4.1 system and a Solaris 2.5.1 system. The performance of these systems varied, due to different processor speeds, but the overall trends were identical. We measured performance for a DFS only fileset and compared it to archived and mirrored filesets. We also measured the performance for an HPSS only fileset and compared it to a mirrored fileset. To determine the impact of the DFS SMT on file system performance, we measured performance on Episode aggregates with and without DFS SMT. In both cases, the DM application (HPSS/DMAP) was not running. The numbers reported are averages of several runs, where each run was performed with an empty cache.

DFS SMT Performance

Sequential read/write performance was measured using the UNIX "cp" command to copy a file between the Episode aggregate and a UFS aggregate. Our tests found that an aggregate configured with the DFS SMT layer enabled to support DM applications, had a 1.5-2.5% decrease in read/write performance. The small overhead due to the extra layer in the file system is overshadowed by the time taken for synchronous disk I/O.

In addition to read/write performance measurements, we also ran the NFS Connectathon benchmark suite that attempts to thoroughly exercise file system functionality such as file and directory creation and deletion, lookup, setattr, getattr, chmod, stat, read, write, link, rename etc. We found that it takes 5.7% longer for file system calls to execute on a DFS SMT managed aggregate. In contrast to the read/write tests, the Connectathon test may lead to a significantly smaller proportion of synchronous I/O operations because of cached metadata. Without the masking effect of I/O, the overhead incurred by the DFS SMT layer is more perceptible.

We are working towards making the DFS SMT code more efficient, and expect the overhead to decrease in future releases.

DFS Performance

Archived filesets generate and process events whenever files are created and deleted, incurring some overhead. Events are not posted for any other name space events, such as link, rename, etc., so these operations, perform at the same rate as a DFS only fileset. Unless file data must be staged from HPSS, I/O performance rates for an archived fileset is identical to those of a DFS only fileset.

Creating a file in an archived fileset incurs an overhead between 30-40% above creates in a DFS only fileset. This overhead is attributed to event processing in DFS SMT and to event handling by HPSS/DMAP which must set DM attributes, which involves I/O. The cost incurred for processing file deletes has some variance. At a minimum, the DM attributes must be checked to determine if the file has migrated to HPSS. In this case, the overhead for deletes is 20% greater than deletes in a DFS only fileset. If the file has migrated to HPSS, HPSS/DMAP must log object handle information for the file and the overhead increases by the amount of time it takes to write to the log.

The overhead associated with client I/O to files is insignificant for both mirrored and archived filesets, unless the data must be cached to DFS. Moving data between DFS and HPSS is highly dependent on disk speed, network speed, and system load, but during our tests, data transfers between DFS and HPSS were as fast as permitted by the hardware.

HPSS Performance

Mirrored filesets incur an additional overhead for any name space activity that alters the name space. Such activity includes creates, unlinks, renames, and owner or permissions changes. Name space activity that does not alter the name space, such as "ls" and "cd", perform at the same rate as HPSS only filesets. In general, the overhead for keeping the DFS and HPSS name spaces synchronized is about 10% greater than the same activity in an HPSS only fileset. The overhead seen from HPSS is significantly lower than the overhead seen from DFS because HPSS has a higher cost for name space updates than DFS. This cost is largely due to the infrastructure and efforts are underway to reduce it.

I/O throughput for mirrored filesets is the same as for HPSS-only filesets, if all the required data is present on HPSS media. If data must be migrated or purged from Episode, HPSS I/O may be delayed. The length of the delay depends on the amount of data to be migrated, network and media speed, and the load currently on the DFS and HPSS systems.

Data and name space activity on HPSS-only filesets perform at the same rates as previous HPSS releases. The cost incurred by the use of filesets is insignificant.

Conclusions

The integrated DFS/HPSS system is flexible enough to support a variety of environments. Data I/O rates for both DFS and HPSS are consistent with the rates before DMAPI was implemented, except when data must be migrated or staged. Supporting partial data migration and caching allows pieces of files to be moved, improving the speed of migrating and caching file data between DFS and HPSS.

Providing consistent name spaces between DFS and HPSS is possible, but at an additional cost. However, the additional cost of creating files becomes less significant as the files increase in size.

Drawbacks of the integration include greater administrative complexity and performance overhead arising from DMAPI, especially when there is a need to keep the HPSS and DFS name spaces synchronized.

Future Work

Future DFS work includes support for cloning filesets on DFS SMT managed aggregates, to enable fileset movement and replication. When dumping a fileset, DFS will allow for purged data to be recalled from tertiary storage, supporting full and abbreviated fileset dumps. Extensions will be added to the DFS client-to-file-server protocol, allowing the DM attributes to be examined.

HPSS/DMAP can be ported to other platforms that support DMAPI, allowing these file systems to be archived in HPSS, however, mirroring name spaces will be impossible without DMAPI extensions. Functionality must be added to HPSS/DMAP to support cloning and replication.

Though we have made DFS/Episode conform to DMAPI, it is not a perfect match since DMAPI mainly addresses local file systems like UFS. To provide a better match between DFS and DMAPI, we would like the following extensions to DMAPI:

- To support backup applications, it must be possible to capture all attributes for a file (ACL, Episode file property lists, etc.).
- DFS allows filesets to move between file servers at different sites. We would like to move filesets without recalling all the data. The DMAP at the new site should be able to interpret DM attributes correctly and recall migrated data. This requires DM attributes to contain location information.

HPSS would like the following extensions to DMAPI:

- Support for permission change events and additional events for name space synchronization.
- Modification to the locking and I/O interfaces for better parallel file system support.

References

[1] S. S. Coleman, R. W. Watson, R. A. Coyne, and H. Hulen, "The Emerging Storage Management Paradigm", *Proceedings Twelfth IEEE Symposium on Mass Storage Systems*, Monterey, CA, April, 1993.

[2] Kazar et al., "DEcorum File System Architectural Overview,. *of USENIX Summer Conference*, Anaheim, California, 1990.

[3] D. Teaff, R. W. Watson and R. A. Coyne, "The Architecture of the High Performance Storage System (HPSS)", *Proceedings Goddard Conference on Mass Storage & Technologies*, College Park, MD, Mar. 1995.

[4] CAE Specification: Data Storage Management - A Systems Approach, The Open Group, 1997.

[5] Cray Research, Inc., Data Migration, UNICOS™ System Administration Guide Volume 1 (SG-2113), 1991.

[6] M. J. Teller and P. G. Rutherford , Petabyte File Systems Based on Tertiary Storage, htpp:/www.emass.com/World_HQ/Collaterals/Whitepapers/Petabyte.pdf.

[7] R. E. Bredehoft, “Advances in E-Systems Modular Automated Storage System (EMASS™) For the Cray Environment”, *Cray User Group Spring Proceedings*, Berlin, April 1992.

[8] M. W. Collins. and C. W. Mexal, “The Los Alamos Common File System,” *Tutorial Notes*, Ninth IEEE Symposium on Mass Storage Systems, October 31, 1988.

[9] F. McClain, “Distributed Computing Solutions, DataTree™ and UniTree™: Software for File and Storage Management,” *Digest of Papers Tenth IEEE Symposium on Mass Storage Systems*, 1990.

[10] R. A. Coyne and R. W. Watson, "The Parallel I/O Architecture of the High Performance Storage System (HPSS)", *Proceedings Fourteenth IEEE Symposium on Mass Storage Systems*, Monterey, CA, Sept. 1995.

[11] R. Hyer, R. Ruef, and R. W. Watson, "High Performance Direct Network Data Transfers at the National Storage Laboratory," *Proceedings Twelfth IEEE Symposium on Mass Storage*, Monterey, CA, Apr. 1993.

[12] Chutani et al., “The Episode File System”, *Proceedings of the Winter USENIX Conference*, San Francisco, California, 1992.

[13] Open Group, "Systems Management: Data Storage Management (XDSM) API", *Open Group CAE Specification*, Prentice-Hall, March 1997.

[14] Kleiman, "Vnodes: An Architecture for Multiple File System Types in SUN UNIX", *Proceedings of the USENIX Summer Conference*, 1986.

[15] J. Heidemann and G. Popek, "File System Development with Stackable Layers", *ACM Transactions on Computer Systems*, 12/1, February 1994.

Page intentionally left blank

A Method To Share Data Between Compute Clients With Network Attached Storage Implementation, Operation, and Performance Results

Ken Fallon
Bill Bullers

Impactdata
605 E. Huntington Dr.
Monrovia, CA, 91017
billb@impactdata.com
kenf@impactdata.com
+1-626-359-4491, +1-626-930-9405
Fax: +1-626-930-9478

Abstract: A Distributed Storage Node Architecture (DSNA) where multiple compute clients can concurrently share the same data through a high-bandwidth file system will be installed at the National Energy Research Scientific Computing Center (NERSC) and at the NASA Ames Research Center. A storage protocol defined by the DSNA, includes methods that produce applied, distributed, concurrent data management capabilities designed to allow multiple SGI and CRAY clients to reliably share data through network attached storage. This paper discusses the implementation, operation, and measured performance of the DSNA and the applied protocol. Implementation issues and initial test results are discussed. DSNA is an available, open standard.

A Network Peripheral Adapter (NPA) is embodied within the Distributed Storage Node Architecture of a network-centric computing and data storage enterprise. The NPA is an intelligent controller and optimized file server that empowers network-attached processors with DSNA protocol the means to unlock the potential of distributed data access while enjoying the benefits of centralized management and control. Optimized for high-performance and peer-level storage, the NPA is fabric-independent and capable of linking virtually any workstation, high-performance computer and/or network with any type of storage device.

By connecting high-performance disk arrays and tape drives to high-speed networks, the NPA creates network storage and allows computers to access and share data. First integrations target data sharing between multiple SGI, CRAY, and NT systems but will be extended to include SUN, HP, and AIX clients.

Introduction

The cutting edge of business, science, and industry, driving for computing, visualization, communications and information content are constantly creating challenges in data and information storage and retrieval processes. Context properties and intelligent formats for management agents are being applied to data. The ever increasing demand for managing larger quantities of data, and the growing requirement for rapid, distributed, global access is forcing the exploration of new approaches to handle and process data to be stored, searched, mined, and made available. This demand is sponsoring the evolution of data storage products that insulate users from storage location, media, and protective services, and is bringing about the concept of enterprise wide ***network computing***. Retrieval, archive, backup, hierarchical storage management applications, and distributed objects are evolving but, intelligence within the data storage devices and media has been slow to develop. As a result, there is an overwhelming demand for more sophisticated storage servers and intelligent storage media controllers.

This paper describes measured performance results using an architecture that provides intelligence at the storage device level and enables network attached processors to access and share data.

Distributed Storage Node Architecture

Distributed Storage Node Architecture (DSNA) was created to facilitate data management, data storage, data sharing, data archive, distributed objects, backup and retrieval. DSNA was developed with the flexibility to expand with the needs of high performance computing and high speed network users. Network file systems, like NFS, operate by copying data through multiple memory levels from the client to the file system, to cache, to user space, using small datagram block sizes. As a result they are inherently unable to provide high-performance network data-flows. These systems typically apply stateless protocols, with unmanaged concurrency, that requires the full intelligence including file serialization to be provided by the client. NFS clients must be 'smart' because NFS servers are 'dumb'. DSNA is designed with an entirely different structure that integrates file system intelligence into the server and operates with peer-level data flows to deliver large blocks and avoid multiple memory transfers. DSNA is a solution to the limitations of sending large files at high speeds. Applying DSNA protocol, a set of drivers is implemented with a Common Peripheral Interface (CPI), to utilize high speed networks effectively. There are no application level client drivers required.

DSNA is structured to be a complete storage system solution that provides several integrated benefits:

- A Coherent and Cohesive Network Storage Environment
- Intelligent Storage and Data Mining
- The Management of Data Stores
- Handling Increased Network Bandwidth and Response Times
- Storage Servers and Controllers that Provide for Rapid Growth and Change
- Support for Evolving Distributed Object Storage

The implementation of Distributed Storage Node architecture is based on standard components that support distinct network and storage device interfaces. These components are used to achieve fabric-independent integrations for high-performance and traditional networks. DSNA can accommodate interface components for connection to HIPPI, Fibre Channel, ATM OC-3/12, SCSI, Gigabit Ethernet, and even 10-BaseT and 100-BaseT. These components are integrated in a Network Peripheral Adapter (NPA) and are connected together through a PCI data bus. High-rate data transfer is achieved by bus-mastering the interface components. Taking advantage of memory buffers built into the interfaces, the NPA processor reacts to data transfer commands by initiating peer-level Direct Memory Access (DMA) data-flow between the components and eliminates multiple processor/cache read interactions with the data. By circumventing the processor, data-flows up to 100 MB/s can be achieved across the 32 bit, 132 MB/s PCI bus.

Figure 1 shows the internal structure of the NPA.

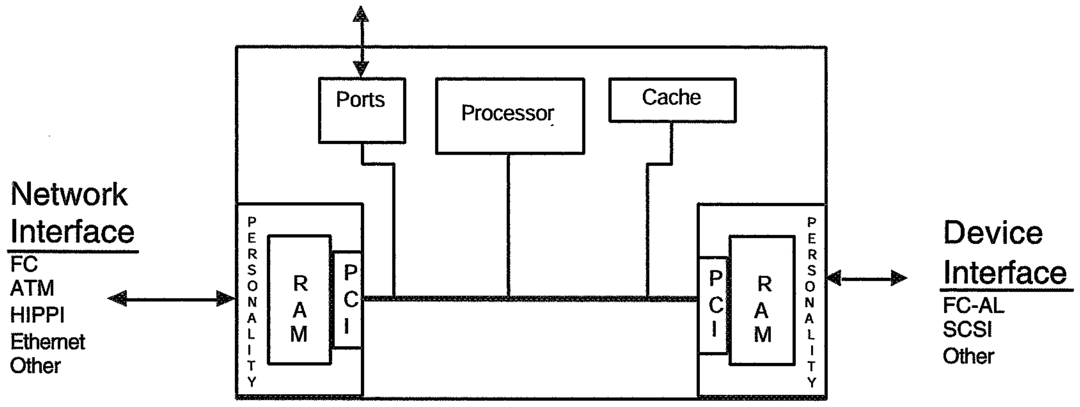


Figure 1

The PCI bus-mastering technique achieves high-rate data-flow while significantly freeing the burden on the processor and operating system, and allows a low cost NPA implementation with 'Wintel' technology. The use of standard component modules achieves significant flexibility for the DSNA to provide data storage in an open system context. The physical level operation and structure embodied in the NPA is a significant building block, but is only one of several major elements that enable DSNA to provide high-performance distributed data storage.

Elements of DSNA

DSNA creates an environment for multiple network attached storage nodes each with a Network Peripheral Adapters (NPA) and attached storage devices as shown in the next figure.

DSNA supports the notion of distributed file management across NPAs and provides the capability for true managed file-extent concurrency across clients. Files can be created on one client with extents of the file opened, concurrently modified, or locked for pristine manipulation by other clients, even using different operating systems. Each storage node can be setup with DSNA applications that provide Hierarchical Storage Management, data archive, backup, and disaster recovery processes. DSNA provides integrated storage management as part of a 'Distributed Management' methodology in which any processor on the network can assert management functions. Each processor maintains its own management data, such as operational status, configuration records, file and media level storage statistics, alarms, accounting and billing, and software update facilities. Each processor can query every other processor and display this management information for any storage node. Any NPA can also be setup to act as a security server to manage the security resources across a DSNA network. Operator access requires ID and password authentication with data access restrictions by ID. C2 security and POSIX compliance are provided in all processors.

A DSNA storage node implementation can be as simple as a single NPA and storage device that provides intelligent network connected shared data storage. It can be a diverse hierarchical arrangement of disks, removable media drives, and libraries.

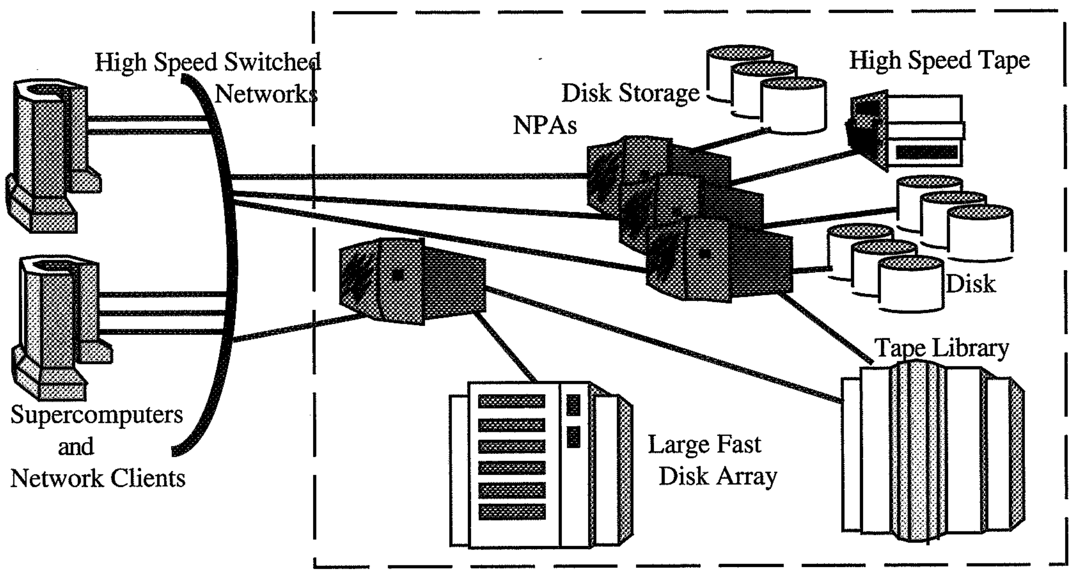


Figure 2: Typical DSNA Storage Nodes

Personality Modules defined to support standard application familiar protocols such as NFS and FTP are embodied within DSNA. Personality Modules include distributed object services with support for Object Request Brokers as defined through CORBA and ActiveX to promote open access to intelligent storage.

The Common Peripheral Interface is middleware that enables UNIX (and Windows NT) clients to use and share network storage as if it were locally attached. Used with any application that reads and/or writes to locally attached devices, the CPI converts Input/Output (I/O) operations to DSNA network commands with DSNA protocol. Users can take advantage of either a 'transparent' access to DSNA storage nodes or an Application Programmers Interface (API), through which they can apply a rich set of DSNA features including metadata and local file management. The DSNA protocol has as its foundation the proven IPI-3 storage protocol over which several extensions are specifically designed to accomplish high-performance network shared data storage. Device drivers are provided for targeted systems that transform standard file I/O requests into DSNA commands and messages. These CPI drivers provide a local file system connection through defined mount points to DSNA storage nodes with disk and tape media. Host application programs only require modifications if the rich DSNA feature set is to be fully exploited. The DSNA protocol and the CPI integrate a metadata facility that collects useful information about the files, the storage devices, and the media. The DSNA Metadata is a vital component of the Storage Node File System (SNFS). The information is collected in real-time and is available to DSNA utilities that use the information to locate and manage directories and files, support file access at the block level, maintain file security and locking, and provide operational statistics. Other utility agents link this Metadata to the HSM and Archive Manager to place and locate files.

Common Peripheral Interface Operation

CPI drivers support two modes of operation, Transparent Mode, and API mode. Transparent Mode allows existing applications to apply DSNA without modifications and achieve high-performance results. Transparent Mode does not give user applications the full range of DSNA features. API mode makes the full feature set available to the user. Applications that use Transparent Mode issue OPEN, CLOSE, IOCTL, READ and WRITE calls to a mount point. The mount point configuration/access table insures that the call is routed to the SNFS interface where it is converted to CPI commands. Two classes of storage device are supported, tape and disk. The API mode is a programmatic interface to the CPI Driver that uses calls to OPEN, CLOSE, READ, WRITE, IOCTL and CPICMD. CPICMD makes Metadata and other DSNA capabilities available to the application. These capabilities include access to file metadata to exploit a broad set of execution, and administrative content:

- Formatting and Configuration
- File Attribute Reports and Control
- Metadata Reports and Controls
- Operation and Execution Status
- Abort Execution Methods
- File Mark and Position Control
- Diagnostics and Error Logs
- Session Control
- Data Delivery Times

Figure 3 shows the difference between how CPI Transparent Mode and API Mode operate within User Space (Application level) and Kernel space (CPI level).

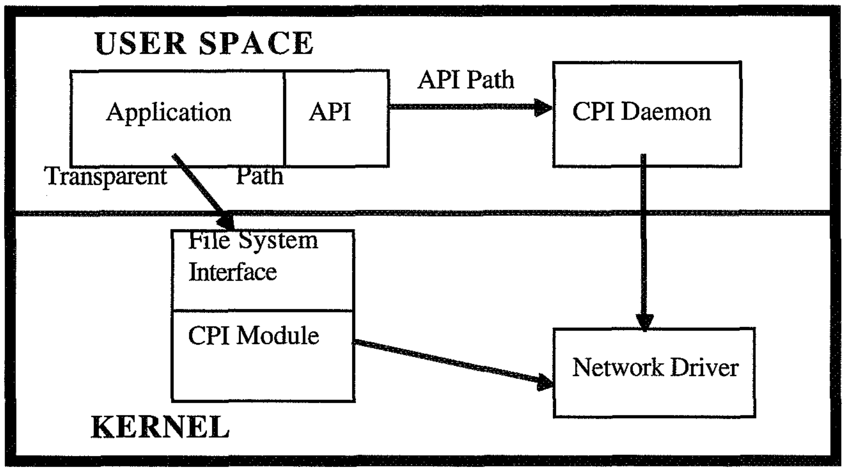


Figure 3: Common Peripheral Interface Operation

API Mode

Details of the API Mode operation are presented in the next diagram. The CPI daemon process and the user application that calls the API are independent UNIX processes. The API code runs as part of the application and communicates with the CPI daemon through UNIX inter-process communication facilities. The CPI daemon can service multiple applications allowing several simultaneous accesses to the same NPA storage device. The API is a socket connection interface through which the CPI daemon and application communicate. The CPI daemon spawns a separate network 'child' process for each socket connection and returns the results to the application. The socket connection is used to communicate the API commands and responses. The transfer of data, defined in the API call, takes place through shared memory. The socket connection alerts the CPI daemon to associate an area of shared memory with the application that is used for the data transfer.

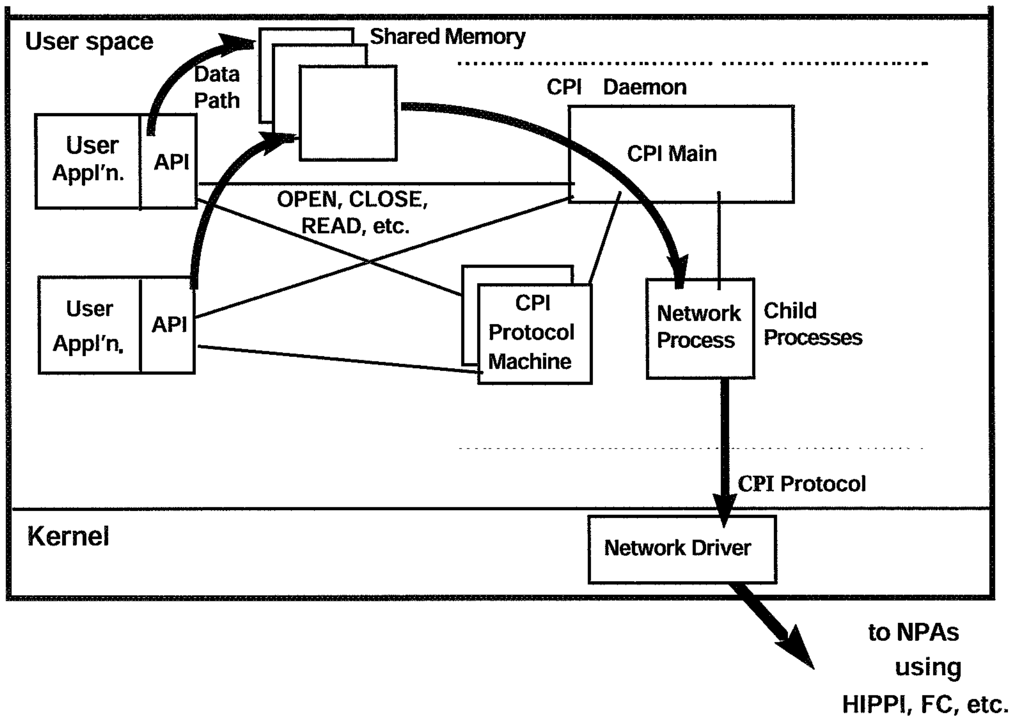


Figure 4

Transparent Mode

The file system appears to be local to the UNIX client in the CPI Transparent Mode. UNIX commands operate normally even though files are maintained on the NPA and available through the network. The CPI Driver uses the mount point identifier to route the file request to the Storage Node File System (SNFS). Through the SNFS, requests are converted to CPI/DSNA protocol and issued to the network driver which then delivers them to the NPA. The NPA processes the request and delivers the data to the destination device through the bus-mastered interface. The SNFS operates as a Virtual File System within UNIX and supports all the attributes of a local file system. The Local UNIX file system processes client application requests and routes the requests through the UNIX Vnode Interface to SNFS as shown in the diagram that follows. The SNFS is architected into UNIX in the same way other file systems like NFS integrate.

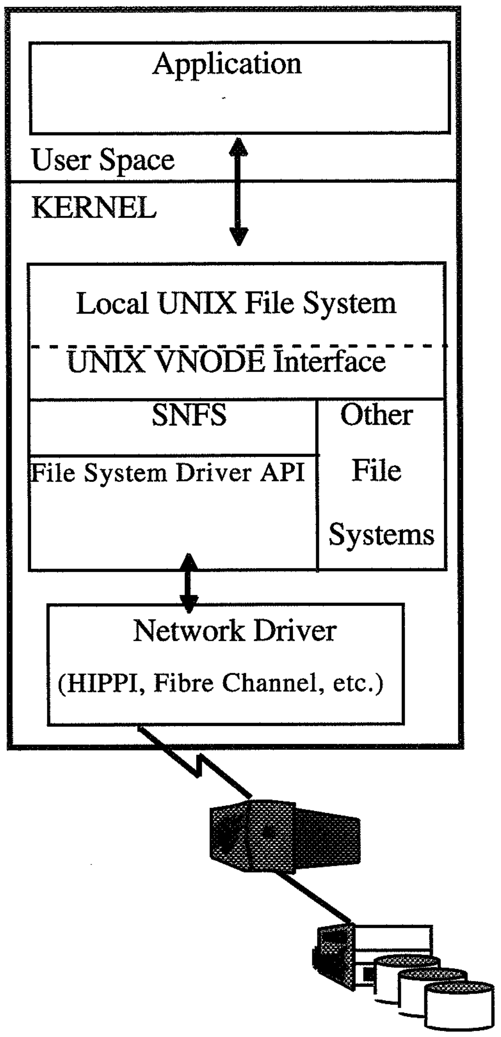


Figure 5: File System Interface

The SNFS supports features that are common to the Local File System. A configuration/access file is used to enforce system security through encrypted NPA address IDs, license keys, and passwords. Metadata maintained at the NPA, controls file access modes including locking, and maintains user ownership, group ownership, authorization, and privilege information that manages user files. Files that are shared by multiple clients can be locked at different levels including NO locking, READ/WRITE locking, and WRITE locking. The default is lock on WRITE.

CPI supports multiple concurrent accesses from different network attached clients but it is envisioned to support multiple accesses from a single client. The initial implementation of the architecture is limited to a single shared file handle per client but future releases will enable separate file handles to be created for each of multiple file accesses made by a client. This capability will permit concurrent parallel processing applications to be performed on distinct extents of a file by a single client.

Other File Systems

DSNA is structured to operate within several different file systems based on the mount point that is selected. The simplest format for operating with multiple file systems like NFS, and SGI's BDS, is to define and allocate specific disk partitions and tape volumes to each system. Other architectures have been devised that reformat files and transfer their data from one file system to another. DSNA implements a completely open solution that eliminates the need for separate partitions and allows data to be concurrently shared across processes and across file systems. The Windows NT operating system provides an NT Redirector that allows Personality Module drivers to be written and installed that 'redirect' file accesses for selected mount points from the 'other file system' to CPI. A client application that chooses to operate through NFS, for example, is able to do that by defining specific mount points for NFS files to be managed. The NT Redirector diverts the file access control from NFS through CPI to the NPA and allows the file to be opened at the NPA as an SNFS file and at the client application as an NFS file. Clients attached to networks that support TCP/IP such as Gigabit Ethernet and future releases of the Essential Communications HIPPI Card can apply the network performance benefits of CPI to standard current applications.

DSNA Integration and Benchmarks

DSNA was first integrated as a high-performance network storage node in a High Performance Storage System (HPSS) at The Caltech Center for Advanced Computing Research by connecting a 72 GB FibreRAID through an NPA to the HIPPI network. The HPSS server provides native support for third-party IPI-3 disk which allows direct data transfers to be setup and executed from the HPSS server to create a data flow path directly between the disk and the client application. This avoids the performance lag caused by staging the data through multiple memory read operations. Several performance measurements and benchmarks have been made using files of 64MB and smaller. Performance using large files (to 2GB) will be evaluated in early 1998 to compare transfer rates between local disk and network attached disk. A similar configuration will be installed at the Lawrence Livermore National Laboratory where additional benchmark testing is planned. These first integrations do not explore the full capabilities of the DSNA because they are limited to direct IPI-3 connection and do not require the data sharing capability of the Storage Node File System.

Performance

The use of standard, commercially available network and device interface components to structure a cost efficient design has required close cooperation between Impactdata and partner suppliers to realize the performance objectives of the DSNA. A Pentium Pro processor card in a passive backplane system with a standard 32 bit, 33 MHz PCI bus is used in all the testing. HIPPI network connectivity is accomplished with the Essential Communications PCI adapter using an Impactdata developed peer-level driver that includes a DMA stream engine. For disk array attachment with SCSI protocol on Fibre Channel, Emulex and Systran Adapters have been tested. Both use Impactdata developed drivers optimized for bus mastered peer-level transfers. Current test results show 35 Mbytes per second data transfers with small transfer sizes of 4MB. This rate was achieved by adding a 16MB transfer memory buffer between the network and device cards. Impactdata expects to achieve 50 Mbytes per second by expanding the memory buffer size to 128MB and increasing the transfer size to 32MB. Additional driver optimization will be possible with firmware changes in work at Essential and the Fibre Channel card vendors. These changes coupled with data transfers of 128 Mbytes or greater are expected to achieve the objective 65 Mbyte per second transfer rate.

Conclusions and Future Work

The cost efficient implementation achieved through the network peripheral adapter coupled with high data transfer and managed file sharing make DSNA a very practical architecture to embrace. The first applications to use true file sharing properties achieved through CPI are with SGI Origin 2000 and CRAY client systems in an integration at the National Energy Research Scientific Computing Center (NERSC). The NERSC integration is the first opportunity to move DSNA file sharing out of the Impactdata laboratory into a computer center test configuration. An installation is also scheduled at NASA Ames to operate with CPI in the Transparent Mode and enable multiple clients to share files. Additional validation tests to collect file transfer statistics and performance benchmarks are planned at Ames, along with stress tests of concurrency, file locking, and hierarchical storage management. Development is in process for similar CPI integrations with HP Convex, and SUN Systems.

Page intentionally left blank

Towards Improved Tape Storage and Retrieval Response Time

John J. Gniewek
IBM Corporation
9000 S. Rita Road
Tucson, Arizona, 85744
gniewekj@us.ibm.com
Tel: +1-520-799-2390
Fax: +1-520-799-4138

Introduction

Magnetic tape has maintained a prominent place in the computer data storage hierarchy for in excess of forty years. As a result of continuing technology advances, the improvements made in both areal and volumetric recording density have assured that magnetic tape remains the lowest cost storage medium for most computer data storage applications. Relative to the impressive advances in reduction of disk storage costs, tape storage has maintained a 10X to 100X advantage in lower cost per Gigabyte. For applications such as back-up, disaster recovery, and passive archive of very infrequently accessed files, this lower cost, as well as the removability and transportability of the tape storage media, has ensured a continuing role for magnetic tape. The frequent projections of the demise of magnetic tape as a result of disk storage price-performance improvements have ignored the enablement of new applications that require significantly more storage capacity and hence, provide the driving force to maintain the 40+ year storage hierarchy paradigm. Several of these new applications place increased demands on the ability to retrieve data objects quickly and require that the tape storage serve as an active member of the hierarchy, not merely a passive member, as in applications such as disaster recovery. Until recently however, not much effort had been made to improve the retrieval response time of tape. In this metric, tape storage has continued to be disadvantaged by three to four orders of magnitude relative to disk storage.

Concomitant with the very significant tape technology advancements in the last several years, there has been, and continues to be a proliferation of new types of recording devices and robot systems. In order to provide a means for judging suitability of particular types of devices for these new emerging tape applications, it would be desirable to be able to develop a figure of merit for comparing different recording systems, each with widely different component characteristics. The analyses presented here has been of value to design engineers and can be expected to be of value to application systems engineers and system integrators.

Figure of Merit--What is Appropriate?

The diversity of applications precludes the possibility of any one figure of merit representing all requirements equally well. However, we can start by identifying individual favorable attributes and proceed to integrate the individual parameters into a composite figure of merit. This composite figure of merit is developed as a result of applying the analyses to meet the requirements of several specific applications. The approach to constructing appropriate figures of merit is developed in three stages. In Figure 1, a parameter called the 'effective data rate', (EDR), is first presented without system capacity or cost considerations. In Figure 2, drive cost, but not total storage capacity, supplements the EDR parameter. Finally, in Figures 3 and 4, total system cost is factored in and is presented in units of (MB^2/second/$) as a proposed comparative figure of merit. At this last stage, in addition to the device hardware variables, two new application variables, aggregate data rate, and library capacity are introduced.

Efforts to make improvements in retrieval response time for tape storage systems must be considered at the system level involving hardware, software, and application data organization approaches. Hardware factors include items such as robot cycle time, drive load and unload times, tape search and rewind times, drive data rate, recording density (linear and track), cartridge capacity, etc. In addition to these more obvious factors, items such as track format and cartridge design can also provide a means for improving retrieval response time. An example of two different types of drive/cartridge designs illustrated the advantage a midpoint load cartridge design could provide in improving response time for a serpentine longitudinal recording format (1). For certain types of retrieval patterns, this design has the effect of doubling the search speed. The purely mechanical aspects of improving response time by simply going faster runs counter to wanting to reduce costs since without concomitant increases in recording density or mass reductions, faster implies larger, more expensive motors. Thus, in addition to the mechanical improvements, there is room for significant improvements via I/O scheduling algorithms (2), enhancing locality of reference by data organization during the writing operation, and utilizing special attributes of the recording technology to enable partitioning of the cartridge recording format such that a high capacity cartridge can have multiple partitions, all located at the logical beginning of tape.

A) Effective Data Rate--Enhancing Locality of Reference

Smart software and intelligent data organization with some prior knowledge of the expected data retrieval pattern can be expected to have a profound effect on the retrieval response times. This data organization is generally referred to as improving the "locality of reference". The benefits result from a higher 'hit rate' on a mounted cartridge with a data rate more closely representing the device streaming rate. The 'read' performance with various intelligent I/O scheduling algorithms and with knowledge of the drive characteristics has been analyzed and reported at this conference by Hillyer (3) for the 3570 Magstar MP device. Sometimes, however, the read retrieval benefits are not without cost to the 'write' operation. In general, in order to improve the locality of reference, an increased number of tape mounts is required during the data entry operations such that common data types may be collocated on separate cartridge classes. An example from a commercial application of where this would apply is described and used to develop one relevant composite figure of merit parameter. Other examples from scientific/technical applications can be envisioned and could include the appropriate collocation of spatial/temporal data.

For a number of commercial financial applications involving monthly billing cycles, such as check and credit card statement processing, very large numbers of small objects must be processed expeditiously. As this industry evolves to include images, the data capacity requirements increase dramatically. As a result of the storage cost advantage of tape, these high total capacity requirements make it highly desirable to be able to employ tape storage for a portion of the processing requirements. Thus, this application may be defined as: 1) a streaming data ingestion (e.g. from an image capture check sorter, or an instrumentation satellite), 2) intermediate data labeling and processing, and 3) storage to tape in several versions; a) back-up for the original information until first-order processing is complete, b) chronological entry for ad-hoc retrievals for some period of time, and c) collocated data organization for anticipated future processing. In the case of financial records, this would correspond to end-of-the-month statement processing. It is the high capacity requirement that provides the driving force for wanting to use tape. For low capacity applications, all the intermediate processing would employ disk storage.

The relevant figure of merit for this type of application is thus defined as "effective data rate" (EDR) and corresponds to the total amount of data transferred versus clock time. Clock time includes not only the data transfer time, but also all the other nonproductive times including load, search, robot moves, etc. Thus, the EDR figure of merit encompasses more than just the native device data rate. EDR thereby serves as a means of integrating several independent parameters such that a comparison can be made between types of devices with widely different characteristics. Since the effective data rate for any given device is higher for larger object sizes (a greater percentage of the time is spent in the data transfer mode), by evaluating EDR as a function of object size, the interests of different types of applications are considered.

1) Defining the EDR Parameter

Using the commercial application as a template, in order to minimize processing time during the end-of-the-month processing cycle, it is necessary to provide collocation of the ingested data into G groups (a group is a set of cartridges that contains data only for that group). The ingest rate is defined as IR (MB/sec). The analysis requires that the disk buffer size and the number of tape storage devices be balanced to provide the most economical solution. Obviously, if G devices were provided, there would be no need to swap cartridges during the loading operation other than for full cartridges. However, this is not the most economical solution. Rather, we seek to configure the system such that the effective data rate for the write operation balances the ingest rate. An adjustable parameter is the size of the object that will be transferred each time the cartridge is mounted. Thus, EDR as a function of object size written becomes a relevant parameter and serves to estimate the number of drives required given the constraint of a certain disk buffer capacity. The EDR thus serves as an integrated figure of merit and allows direct comparison between devices with diverse characteristics such as, for example,: a) a device with high data rate and high capacity (but long search and rewind times due to long tape length) with b) a device with moderate data rate and capacity, but short search and rewind times. For illustrative purposes, several hypothetical devices are constructed and compared for EDR values. This application corresponds to a scheduled write operation and no queuing is involved. Higher numbers for EDR is in the direction of goodness.

2) Quantifying EDR

The sequence of operations used to develop an expression for EDR is shown in Table 1.

Table 1
Sequence of Operations for Defining EDR

Operation	Parameter
Robot Get Cartridge	
Cartridge Exchange	AS
Load to Drive	LD
Search	C/2KV*
Read	O/D
Rewind	C/2KV*
Unload	ULD

* The search and rewind times are expressed in terms of cartridge capacity, C, search velocity, V, and recording density, K (MB/M). O is object size in MB. All other terms are in self-consistent units to express EDR in MB/sec. (1).

The expression for the effective data rate, EDR is then given as:

$$EDR = \frac{O}{\left[AS + LD + \left(\frac{C}{K \bullet V}\right) + \frac{O}{D} + ULD\right]}$$

In Figure 1, EDR is expressed in (MB/sec) and represents a situation corresponding to a series of APPEND operations that include multiple cartridge mounts. It would approximate the measured elapsed time data rate under conditions where the appended file size is small relative to the cartridge capacity and the cartridge is filled by the multiple write operations. It would not apply for low capacity cartridges where, for example, an 800 MB object is written to an 800 MB cartridge. In this case, a full cartridge write would result from a single cartridge mount and no search and rewind operations would be invoked.

The EDR figure of merit defined in this manner may not represent a specific tape application but it does represent 'goodness' of the device when considered as a measure of throughput. It becomes more relevant as advances in tape recording technology provide higher capacity cartridges. The parameters chosen for several hypothetical devices are shown in Table 2. Results for EDR as a function of object size for these devices are shown in Figure 1. The results presented in Figure 1 represent the base performance of the different devices without regard to cost. In Figure 2, EDR for each device is divided by the device cost so that the figure of merit is represented in units of Bytes/second/$. In both Figures 1 and 2, higher numbers are better. These results provide a figure of merit which values effective data rate without valuing cartridge capacity. For a given technology, higher capacity cartridges result in lower EDR as a result of longer search and rewind times. A means to incorporate capacity and data rate into an integrated system level figure of merit is developed in a subsequent section.

Table 2a
Hypothetical Device Parameters

Device ID

	A	B	C	D	E	F
Parameter						
AS (sec)	8	8	8	8	8	8
LD (sec)	20	40	5	10	15	5
ULD (sec)	10	15	5	5	15	5
D (MB/sec)	10	5	2	3	12	7
V (M/sec)	5	4	10*	1	4	10*
K (MB/M)	35	60	35	80	120	35
C (MB)	10,000	40,000	5,000	20,000	50,000	5,000
DC ($)	30,000	10,000	8,000	8,000	80,000	10,000
CC ($)	50	100	50	50	100	50

Table 2b
Hypothetical Device Parameters

Device ID

	G	H	J	K	M	N
Parameter						
AS (sec)	8	8	8	8	8	8
LD (sec)	20	20	5	5	5	5
ULD (sec)	10	10	5	5	5	5
D (MB/sec)	10	10	7	7	15	15
V (M/sec)	5	5	10*	10*	15	15
K (MB/M)	35	70	35	70	150	150
C (MB)	20,000	20,000	10,000	10,000	30,000	100,000
DC ($)	30,000	30,000	10,000	10,000	30,000	30,000
CC ($)	100	50	100	50	100	150

*Effective Search velocity for midpoint load two-reel cartridge design.
DC and CC are hypothetical costs of drive and cartridge respectively.

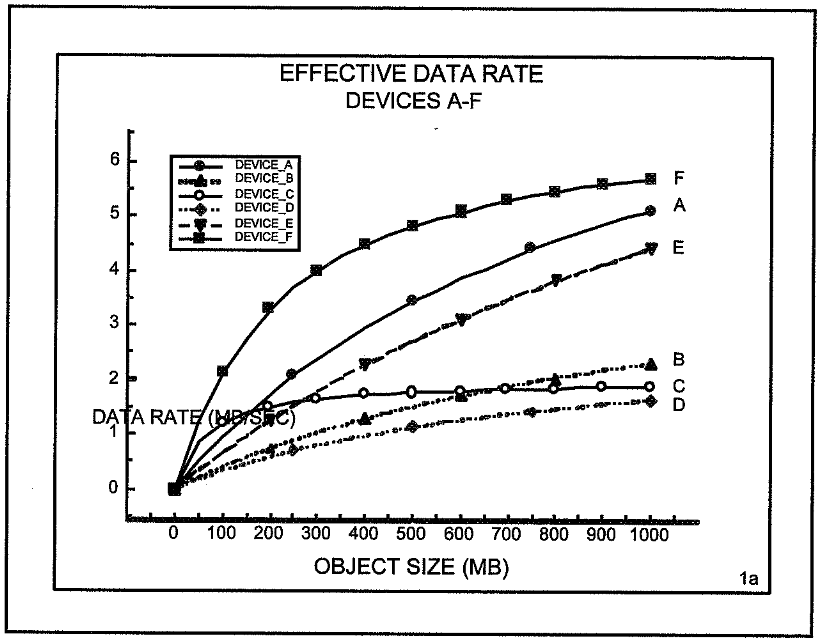


Figure 1a. Effective Data Rate (EDR), (MB/second), as a function of object size transferred in a cartridge mount cycle. Devices A-F. This metric corresponds to random retrievals from full cartridges or to separate sequential write 'Append' operations. See text.

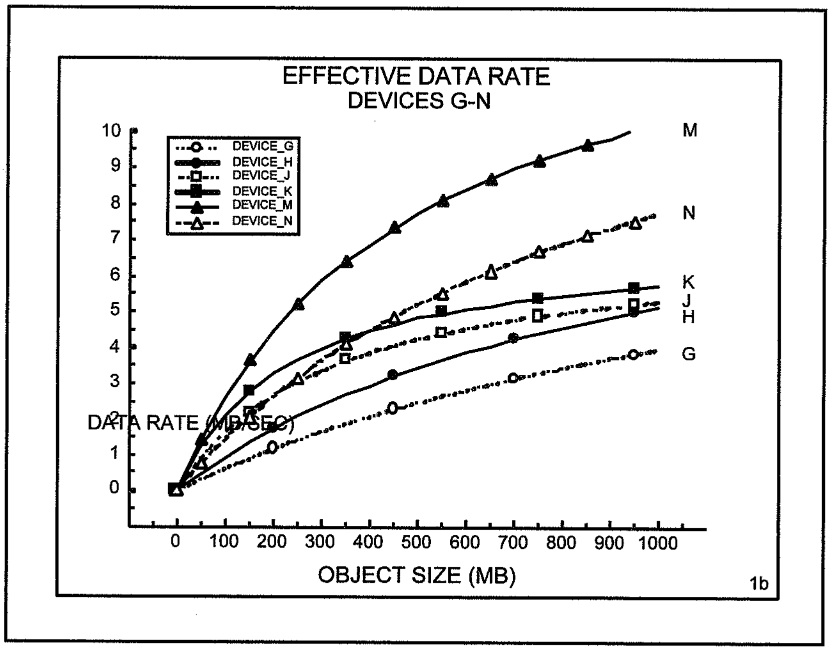


Figure 1b. Effective Data Rate (EDR), (MB/second), as a function of object size transferred in a cartridge mount cycle. Devices G-N. This metric corresponds to random retrievals from full cartridges or to separate sequential write 'Append' operations. See text.

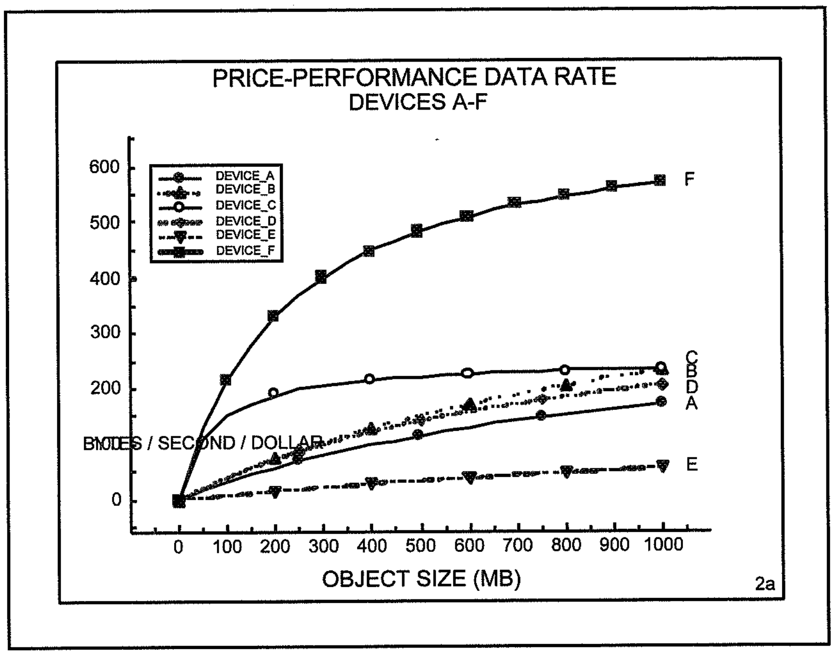


Figure 2a. Price-Performance effective data rate, (MB/second/$). Devices A-F. Calculated as EDR/drive cost per characteristics listed in Table 2. See text.

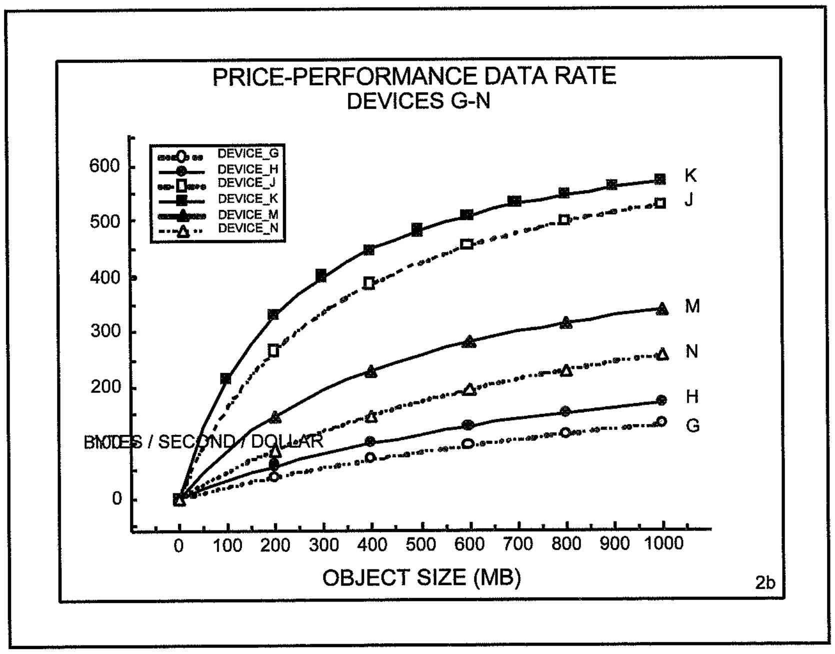


Figure 2b. Price-Performance effective data rate, (MB/second/$). Devices G-N. Calculated as EDR/drive cost per characteristics listed in Table 2. See text.

3) <u>EDR-Discussion of Results</u>

The six devices hypothesized in Table 2a are composites of characteristics similar to those existing in several different current technology devices. From the table of numbers alone, because of the widely disparate individual parameters, it would be difficult to rank the devices in order of effective data rate over the range of object sizes considered. Some non-intuitive results are, however, apparent from Figure 1a which corresponds to these six devices. Device C, which has the lowest native data rate (2 MB/sec), provides a higher effective data rate than all other devices other than Device F for the following object size ranges: Device A (10 MB/sec.), up to approximately 150 MB object size; Device B (5 MB/sec.), up to approximately 600 MB; Device D (3 MB/sec), up to > 1000 MB; Device E (12 MB/sec.), up to approximately 250 MB. When the hypothetical device costs are considered, the results are presented in Figure 2a in units of Bytes/second/$. Devices F and C are then ranked one and two for all object sizes up to 1 GB.

In Table 2b, Devices G and H were constructed such that the cartridge capacity for both is double the capacity listed for Device A. However, this was achieved by different means. Device H doubled capacity by doubling the recording density (and keeping tape length the same), while Device G doubled capacity by increasing tape length at the same recording density as Device A. The improved performance of H compared to G is evident in Figure 1b. Similarly, Devices J and K double the capacity of Device C in an analogous manner. Devices M and N assume a further improvement in recording density representative of what is expected to appear in devices in the near future. These data are presented in Figures 1b and 2b. Note the scale change in Figure 1b relative to 1a. The areal density improvements assumed in Devices M and N are expected reasonable extensions of current technology, but are still far short of the 10 year goals projected by a recent NSIC tape storage study group (4).

B) <u>Cartridge Capacity Considered</u>

The manner by which cartridge capacity may be incorporated into an appropriate figure of merit is not unambiguous. Value can be judged by several different criteria. For a single user desktop application, probably the most important criteria is the ability to provide a single cartridge file backup. Drive cost would also be high on the priority list, but because of the limited number of cartridges likely to be in use, media cost would not be heavily weighted.

For larger users, particularly on a multi-user network sharing common tape storage, tape automation is considered to be essential. The analysis developed here addresses the needs of medium and large size storage libraries where the storage system is composed of drives, media, and automation. A system is defined by specifying: A) the required library capacity (LCAP), and B) a required aggregate data rate (ADR). Device and media costs are assumed per the values in Table 2. Automation costs are introduced in a simplified manner by assuming a fixed cost per slot (in the examples given here, a value of $100/slot is used). The value of higher capacity cartridges is then reflected in the figure of merit as a result of requiring fewer slots and hence lower automation costs. For a given technology recording density, higher capacity cartridges will however, reduce the EDR as a result of longer search and rewind times. Thus, if 'goodness' is considered to be high effective data rate, high library capacity, and low system cost, the figure of merit chosen to represent this composite is: ((EDR) x (LCAP))/System cost. System cost is defined as the sum of media cost (number of cartridges for a capacity of LCAP times the cartridge cost), plus the drive cost (number of drives required to meet the ADR using EDR as the individual device data rate times the individual drive cost), plus the automation cost (simplified here as $100 per slot times the number of slots required). In this analysis, an adjustment factor is provided to allow for different cartridge sizes and the number of cartridges that may fit in a given wall space of the automated library.

The results are presented in Figures 3 and 4 for library capacities of 1 TB and 10 TB respectively. The units are MB^2/sec./$. This results from ((MB/sec.) x (MB))/$. The units do not have any direct functional application and should thus be considered strictly as a figure of merit providing appropriate weighting for data rate, capacity, cost, and application conditions, i.e. library capacity, aggregate data rate, and object size transferred. The step function in the graphs results when the number of drives required to achieve the desired aggregate data rate decrements by one as a result of the higher EDR at larger object sizes. For this example, a value of 20 MB/second was used for the desired aggregate data rate. The difference in the ranking of device types at 1 TB and 10 TB libraries reflects the weight given to higher capacity cartridges at larger library sizes and the amortization of system cost over a greater number of cartridges. Note the scale change between Figures 3 and 4.

The concepts developed in this analysis have been simplified compared to what would be required for a specific application system configuration analysis. Refinements, such as allowance for variable cost of storage slot, alternative definitions for the effective data rate, and treatment of costs as reflective of total cost of ownership, could be introduced. Also, there may be other specific constraints, such as maximum object retrieval response time, storage space, reliability factors, etc., which could provide additional weighting factors in a comparative evaluation. The analysis given here is sufficient to guide product development engineers as to the relative importance of various individual attributes towards a competitive design point and may be useful to systems and application engineers charged with evaluating many diverse potential storage solutions.

Tape Track Format Design

The development of high track density serpentine linear recording formats has provided additional opportunities to improve the retrieval response time from magnetic tape storage devices. In formats of this type, the time required to 'jump' across tracks is small compared to the time required to search down the length of a track. These attributes, when coupled to intelligent request reordering algorithms, can improve the response time required for retrieving multiple objects randomly located within a given cartridge (2,3).

The manner by which the serpentine track format is written varies among the different types of devices. In some cases, the tracks are written by a 'shingling' process where the width of the track left on tape is less than the write head width as a result of partially overwriting the previous track when the write head is indexed to a new position. In other designs employing precise track-following head positioning servo systems, each track is independently written without overlapping the neighboring tracks. This format design is thus amenable to logically partitioning the cartridge capacity into multiple partitions, each partition located at the logical beginning of tape and each partition capable of being individually rewritten while maintaining the data integrity of the neighboring partitions. The number of partitions and the capacity of each is a function of the total cartridge capacity and the number of tracks that are written concurrently.

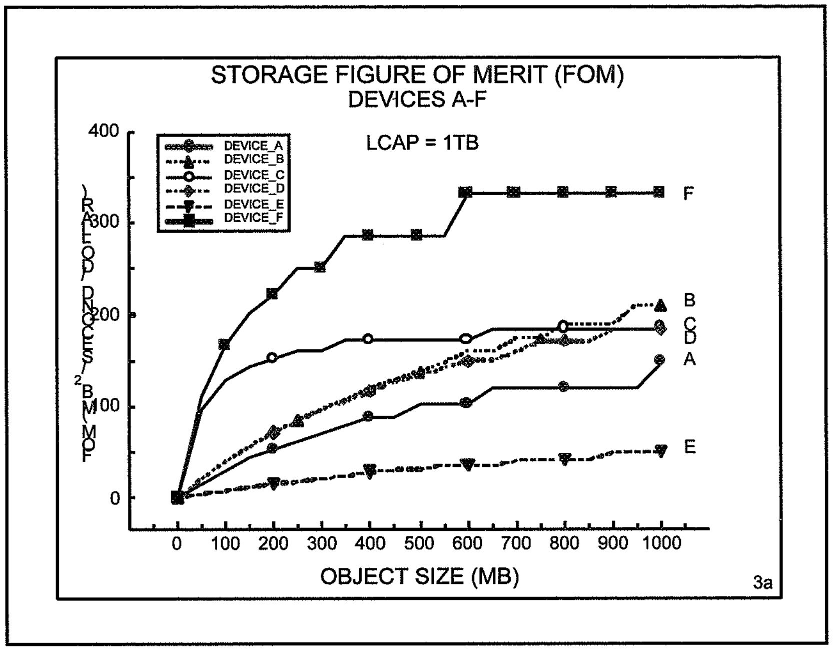


Figure 3a. Storage Figure of Merit in units of (MB^2/second/\$). Devices A-F. Library capacity is 1.0 TB. Aggregate effective data rate is 20 MB/second. See text.

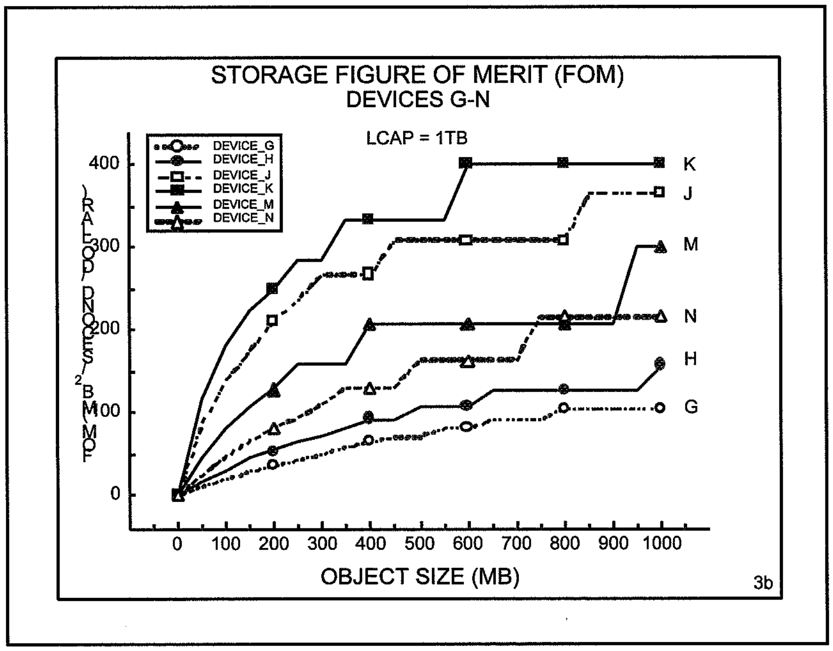


Figure 3b. Storage Figure of Merit in units of (MB^2/second/\$). Devices G-N. Library capacity is 1.0 TB. Aggregate effective data rate is 20 MB/second. See text.

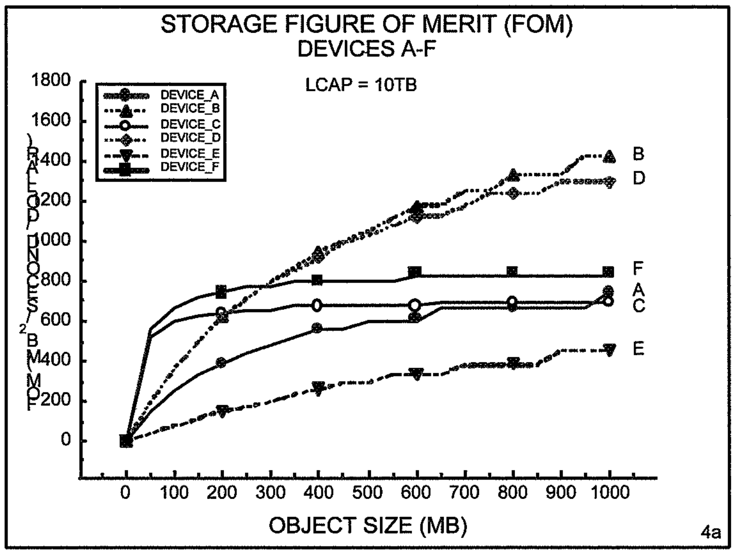


Figure 4a. Storage Figure of Merit in units of (MB^2/second/\$). Devices A-F. Library capacity is 10.0 TB. Aggregate effective data rate is 20 MB/second. See text.

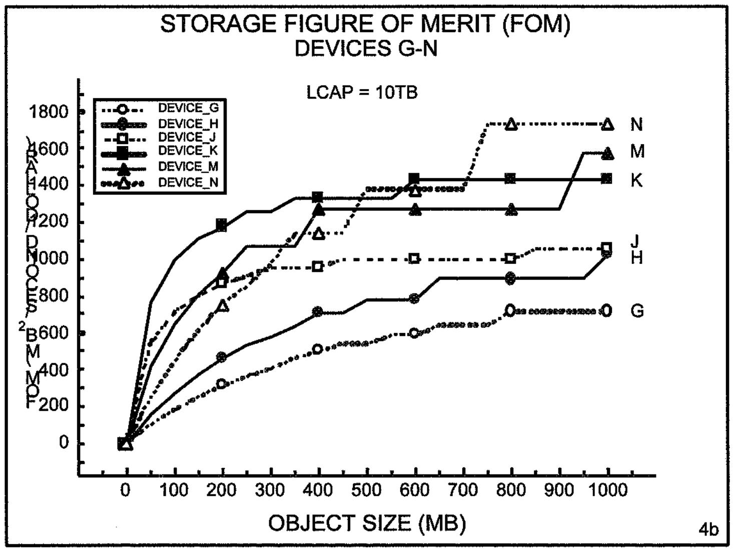


Figure 4b. Storage Figure of Merit in units of (MB^2/second/\$). Devices G-N. Library capacity is 10.0 TB. Aggregate effective data rate is 20 MB/second. See text.

Consider a cartridge with 128 tape tracks and a capacity of 5000 MB, written 4 tracks at a time, and with a midpoint load. In this case, a total of 32 partitions, each of approximately 155 MB capacity and each beginning at the logical beginning of tape (LBOT) would result. If the application required data set sizes in this range, the response time to first byte of data would be improved as a result of eliminating the search time. Technology factors that enable this capability include track-following head positioning servo systems and precise dimension thin film recording head fabrication processes. A schematic of the track layout illustrating this tape format is shown in Figure 5. Data management software enhancements are required to exploit these features.

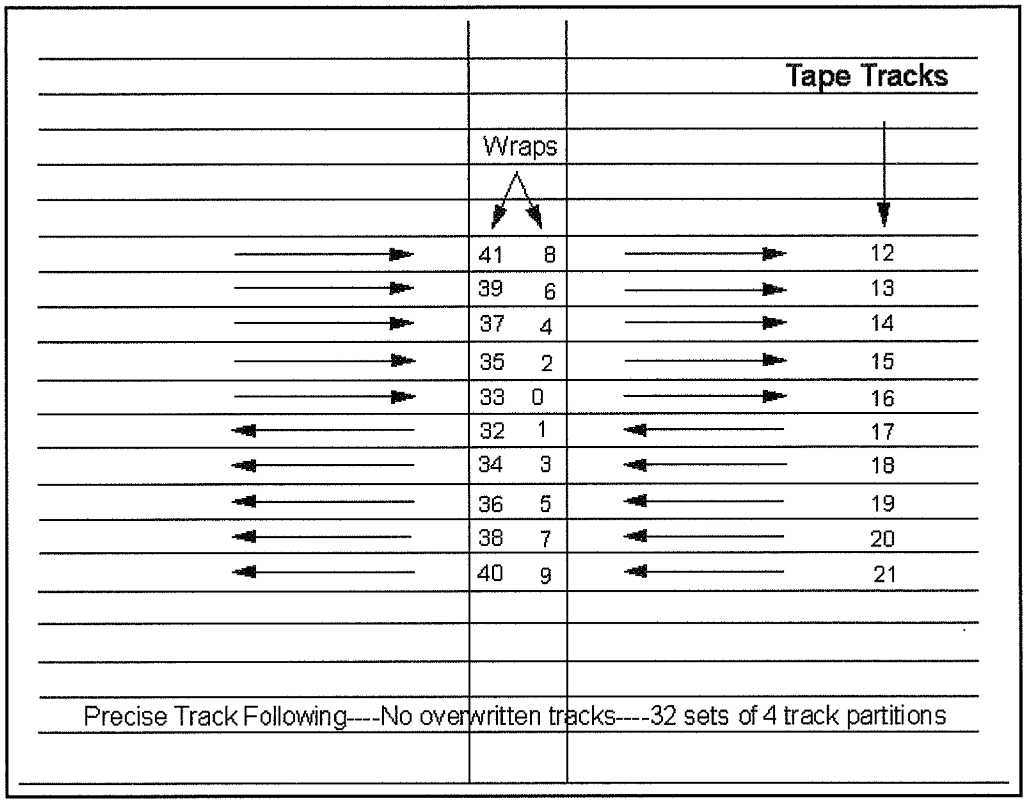

Tape Tracks
Wraps
41 8
39 6
37 4
35 2
33 0
32 1
34 3
36 5
38 7
40 9
12
13
14
15
16
17
18
19
20
21
Precise Track Following----No overwritten tracks----32 sets of 4 track partitions

Figure 5. Schematic of track layout for a track-following servo, midpoint load tape device illustrating capability of multiple logical partitions, all beginning at the logical beginning of tape. Sequentially written tracks do not overlap or overwrite previously written tracks. Illustrated is a portion of the track sequence for one head element of a track format with four concurrently written tracks. 'Tape Tracks' shows track location on tape. There are a total of 128 data tracks on tape. The sequence of written tracks proceeds from 0 to 1 to 2....etc. Middle region is the midpoint load region. The right half is written first whereupon the left half begins writing at wrap # 32. Any full wrap, for example (4,5), may then be overwritten while maintaining integrity of the neighboring tracks.

Integrated System Solutions

Current open system storage systems incorporate, via software, direct access to tape storage in a manner that appears as expanded disk storage, albeit with longer response time. Recently several system solutions that feature 'virtual' tape drive capability for the mainframe market segment have been introduced. Common to both approaches is the use of a disk buffer that results in greatly improved performance in the write to tape mode and in the ability to get full utilization of the cartridge capacity. Retrieval of data is improved to the extent that the 'hit rate' to the disk cache is significant. Thus, the aggregate performance will be very much application dependent. In situations where the library capacity is large and there is a random retrieval pattern, real tape devices with fast response time are required. Under high load conditions, queuing delays become significant. The response time as a function of various application conditions and types of devices has previously been presented (1).

Tape Storage Trends

The natural operating domain for tape storage is in systems requiring large storage capacity. What is considered large is, however, a moving target. The lower cost per MB of tape storage must translate into significant actual dollars to accommodate the total costs of a storage hierarchy. Continued decreases in $/MB for disk storage and the introduction of new technologies such as high density removable floppy disk storage, and increased density recordable optical disk products are putting pressure on tape storage devices at the individual desktop user market segment. For tape storage to maintain a presence in any individual market segment, tape storage costs should maintain at least one order of magnitude advantage relative to competing technologies that have more favorable response time attributes. This is expected to be maintained for midsize and large storage capacities, but is doubtful for low-end individual user segments (4). Based on the NSIC study, improvements in tape storage volumetric density can be expected to advance in an evolutionary manner by 10-20X over the next 10 years. Some of these technology advances will be used to improve tape storage and retrieval response time.

Conclusions

The analyses presented here have made an approach to developing a comparative figure of merit that allows quantitative comparison of devices and systems with widely different characteristics. This was prompted by the arrival of new applications for tape storage which must be analyzed in a manner that requires greater sophistication than the historical metrics of only cartridge capacity and device data rate. Use of such comparisons can be expected to lead to further product enhancements that will provide improved response time from tape systems.

1. Effective data rate is proposed as a measurement parameter, in general, more useful than native device data rate. It gives appropriate weighting to other drive attributes and is reflective of the throughput that can be expected. Examples were given that illustrated higher effective data rates for lower native data rate devices.

2. The key tape device technology factor that would allow improved response time (at constant cartridge capacity) is the recording density, K, given as MB per meter length of tape. From basic recording physics constraints, this can be expected to be achieved predominantly by higher track densities. Most recent developments have concentrated on increasing capacity via thinner, longer length tape. Response time is degraded relative to what could be achieved from density improvements.

3. A system level figure of merit that integrates cartridge capacity, cost, and application conditions, as well as EDR, is proposed as a means of quantitatively comparing widely different system component characteristics. The analysis is of value in guiding development engineering priorities. Enhancements and refinements to the methodology can be expected.

4. Intelligent software is required to take advantage of device characteristics that could improve response time from current technology devices. This could include I/O scheduling algorithms and use of multiple logical volume partitioning within a high capacity cartridge. Continuing future increases to cartridge capacity can be expected to increase the demand for such features.

References

1) J. Gniewek, "Evolving Requirements for Magnetic Tape Data Storage Systems", NASA Conference Publication 3340. Fifth NASA Goddard Conference on Mass Storage Systems and Technologies, pp. 477-491, September, 1996.

2) B.K. Hillyer and A. Silberschatz, "Random I/O Scheduling in Online Tertiary Storage Systems, Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data, Montreal, Canada, June 3-6, pp. 195-204.

3) B.K. Hillyer, "Scheduling Noncontiguous Tape Retrievals", Joint NASA and IEEE Mass Storage Conference, March 23-26, 1998.

4) National Storage Industry Consortium (NSIC) Tape Storage Workshop, September, 1997. Boulder, Colorado. Final report due 1Q98.

Benchmarking Tape System Performance

Theodore Johnson
AT\&T Labs - Research
180 Park Ave., Bldg. 103
Florham Park, NJ 07932
+1-973-360-8779, fax +1-973-360-8050
johnsont@research.att.com

Ethan L. Miller
Computer Science & Electrical Engineering Department
University of Maryland Baltimore County
1000 Hilltop Drive
Baltimore, MD 21250
+1-410-455-3972, fax +1-410-455-3969
elm@acm.org

Abstract: In spite of the rapid decrease in magnetic disk prices, tertiary storage (i.e., removable media in a robotic storage library) is becoming increasingly popular. The fact that so much data can be stored encourages applications that use ever more massive data sets. Application drivers include multimedia databases, data warehouses, scientific databases, data-intensive scientific research, and digital libraries and archives. The research community, has responded with investigations into systems integration, performance modeling, and performance optimization.

Tertiary storage systems present special challenges because of their unusual performance characteristics. Access latencies can range into minutes even on unloaded systems, but transfer rates can be very high. Tertiary storage is implemented with a wide array of technologies, each with its own performance quirks. However, little detailed performance information about tertiary storage devices has been published. As a result, mass storage system (MSS) implementers must rely on vendor-reported numbers or their own tests to select appropriate tertiary storage devices. Additionally, MSS designers must have detailed knowledge of the performance characteristics of their devices to optimally place files on media and perform other optimizations.

In this paper we present detailed measurements of several tape drives and describe the tests used to gather this data. The tape drives we measured include the DLT 4000, Ampex 310, IBM 3590, 4mm DAT, and the Sony DTF drive. This mixture of equipment includes high and low performance drives, serpentine and helical scan drives, and cartridge and cassette tapes. This data is suitable for system performance modeling or system performance optimization studies. By measuring and modeling a variety of devices in a single study, we are able to characterize a wide range of tertiary storage devices. In addition, we hope that our simple benchmarks will become more widely used to gauge tape performance and identify potential performance bottlenecks.

1. Introduction

A tertiary storage system typically refers to a data storage system that uses drives that accept removable media, a storage rack for the removable media, and a robot arm to transfer media between the storage rack and the drives. The media can be disks (usually optical disks) or tapes, though in this paper we concentrate on tape-based tertiary storage. Tertiary storage is used for massive data storage because the amortized per-byte storage cost is usually two orders of magnitude less than on-line storage (e.g., see [1]). Tertiary storage has other benefits, including the removability of the media and fewer moving parts. However, access time to a file stored on tertiary storage can range into the minutes.

In this paper, we measure a variety of devices used in tertiary storage systems and present performance characterizations of these devices. The contribution of this work is the scope and detail of our measurements and the benchmarks used to generate them. We measure many aspects of tape access, including mount time, seek time, transfer rates, rewind time, and unload time. The devices we measure include high, medium, and low performance tape drives. We include measurements that have not previously been published (to our knowledge) but are vital to efficient tertiary storage system implementations, such as short seek times. This information can be used to guide the design of better tertiary storage systems by showing strengths and weaknesses of tape technologies as well as those of specific tape systems.

2. Background

Tertiary storage is often viewed as a necessary evil by file system designers and users — the massive storage capacity of a robotic storage library incorporating high-volume tape drives is needed to store massive data sets, but management of and access to tape-resident data can be painful. Tertiary storage is often used in "write-once, read-never" applications such as storage of large data sets (in which data is rarely reused), backup, and archiving. Today, however, it is becoming more common to use tertiary storage to store active data that is still useful, but is not used sufficiently frequently to warrant the cost of purchasing additional secondary storage. Fortunately, much work has been done on hierarchical storage management systems (HSMs) to simplify access to tertiary storage data.

2.1. Hierarchical Storage Systems

Hierarchical storage systems extend secondary storage (disk-based) file systems by adding tertiary storage as an "overflow area." In a typical implementation, HSMs use secondary storage as a cache for the set of files that reside on tertiary storage. If an application (including a shell tool) opens a file that is not in the cache, the file is brought in from tertiary storage. The user only notices a delay in opening the file. HSMs that have been studied previously include Unitree, HPSS [2], AMASS, and ADSM [3]; there are several other systems that are in production use. An alternative approach is to build a log-structured file system on top of tertiary storage, with secondary storage being treated as a cache for log segments [4,5].

Computer data analysis is transforming scientific research, and it is also forcing the creation of systems that can store terabytes or even petabytes of data. The very large data sets that can be collected have created enormous data storage and retrieval problems. For example NASA's EOSDIS [6], which supports research into climate change, will collect and archive on the order of ten petabytes of data. Many other scientific projects, such as high-energy physics [7,8] also have very large data storage requirements. More recently, though, newer applications such as data warehouses [9], scientific databases [6,10], multimedia [11], and digital libraries [12] are driving the creation of very large databases that integrate tertiary storage into a database system.

2.2. Tertiary Storage Modeling and Optimization

The building of very large scale scientific archives and the efforts at integrating tertiary storage into database systems have motivated considerable recent research activity. In this section we summarize tertiary storage modeling and optimization work. The planning and integration problems of building large tertiary storage installations have motivated recent work in the performance modeling of tertiary storage systems. Pentakalos et. al. [15] present an analytical model of a scientific computing system that incorporates tertiary storage, and Johnson [14] presents a detailed queuing model of a robotic storage library. Menasce, Pentakalos, and Yesha [15] give an analytical model of tertiary storage as a network attached storage device, and Nemoto, Kitsuregawa, and Takagi [16] make a simulation study of data migration.

The above cited research all shares the characteristic of depending on a model of the behavior of tertiary storage devices (robot arms, tape drives, etc.) to either model or to optimize performance. However, the performance of tertiary storage devices is not well understood. As a result, the otherwise high-quality work discussed above use limited, inaccurate, or incorrect models of tertiary storage devices. Fortunately these deficiencies can be remedied by the use of accurate device models that rely on performance measurements gathered from a variety of tertiary storage devices.

While some work has been done to measure, model, and classify the performance of tertiary storage devices, broad-based, comprehensive studies have not appeared, though comprehensive models of secondary storage (i.e, disk drives) have been published [17]. Many works on systems incorporating tertiary storage include benchmarking studies [18,19,20], while other studies have focused on an aspect of a particular device. Ford and Christodoulakis [21] model optical continuous linear velocity disks to determine optimal data placement. Hillyer and Silberschatz [22] give a detailed model of seek times in a DLT 4000 tape drive, to support a tape seek algorithm [23], and van Meter [24] is researching appropriate delay estimation models for tertiary storage. The Mass Storage Testing Laboratory (MSTL) is developing benchmarks for HSM systems [25]; however, these benchmarks test the performance of a software and hardware system, while our benchmarks are only concerned with hardware performance. Additional performance measurement studies include [26] and [27].

3. Taxonomy

The technology used to implement a tape drive influences the performance that the user will obtain from the drive. In this section, we discuss the technologies used to build common tape drives. Table 1 shows the list of tape drive features relevant to the material in this paper. For a deeper discussion of these matters, we refer the reader to the many papers in

Tape drive attribute	Possible values
Data track layout	Helical scan, linear (serpentine)
Tape package	Cartridge, cassette, cassette with landing zones, "scramble bin" (tape loop)
Directory	None, at beginning or end of tape, calibration tracks, embedded microchip
Data compression	Yes, no
Block size	Fixed, variable
Partitioning	Yes, no, not important

Table 1. Characteristics of tape drives.

other proceedings of the IEEE Mass Storage System Symposium and the NASA Goddard Conference on Mass Storage Systems and Technologies as well as other storage system conferences and journals.

A fundamental characteristic of a tape drive is the layout of data on the tape. To achieve a high density, the tape drive must use as much of the available surface area as possible, and a tape is typically much wider than the data tracks. A helical scan tape writes data tracks diagonally across the tape surface, and packs the diagonal tracks tightly together (e.g., as in a VHS video cassette). A linear tape lays multiple sets of data tracks across the tape. Typically, the data tracks alternate in direction, hence the name "serpentine" (e.g., an audio cassette with autoreverse).

The tape package can be a cartridge (containing 1 reel) or a cassette (containing 2 reels). The tape in a cartridge must be extracted from the cartridge before the tape mount can complete. In addition, the tape cartridge must be rewound before it is unmounted. A cassette can be removed from the tape drive without being rewound. However, the tape in a cartridge must be positioned at a special zone (a "landing zone") to ensure that data is not exposed to contaminants. If the tape drive does not support landing zones, the cartridge must be rewound.

The geometry of a tape makes defining the position of a particular block more difficult than for disk drives. Modern data storage tapes typically embed some kind of directory to expedite data seeks; this directory is implemented in hardware and is separate from any user-created directory. These directories can be written at the beginning of the tape (or at other special tape positions), in special directory tracks, or in silicon storage devices mounted on the tape package. A precise directory can permit high-speed seeks. In addition, the requirement to read a directory area can increase the mount time, and the requirement to write a directory area can increase the unmount time.

Many tape drives use hardware data compression to increase their capacity and to improve their data rates. However, compressed data is variable sized. Since the location of a block can vary widely, fast seeks can be more difficult to implement. Similarly, a variable size record length increases the flexibility of a tape drive, but can lead to increased seek times.

Some tape drives allow the user to partition the tape into distinct regions. The Ampex tape drive that we tested allows partitioning, while the others do not. Partitioning simplifies some data management functions, and does not have a significant effect on performance. Some serpentine tape drives that support partitioning can improve seek times within a partition; however, we were unable to gain access to such a device for this paper.

Other factors that can affect performance are the tape transport implementation and the use of caching. Helical scan tape drives need to wrap the tape around the read/write head. Performing a high-speed seek requires that the tape be moved away from the head to prevent excessive wear, resulting in a large delay in starting the seek. Linear tapes use a simpler transport and do not suffer from this problem. Because the data rate from the host may not be constant, many tapes use data caches to allow the drive to remain in streaming mode even if the host machine suffers occasional delays in submitting read or write requests. Additionally, this buffer can be used to store read-ahead blocks; many tape drives read a few blocks after the current location even if they are not explicitly requested (yet). Some drives will return this pre-fetched data after short block seeks.

There are many other considerations involved in tape drive technology, especially those of reliability and longevity, that we do not address in this paper. Another important consideration is cost. Some of the drives we measure in this paper can have an order of magnitude better performance than another drive, but they typically cost an order of magnitude more money as well.

4. Benchmark Methodology

Our interest is to measure and develop performance models for the following access characteristics listed below. Taken together, they summarize the end-to-end performance of a tertiary storage device.

- **Mount time**: This is the time from when the robot arm has placed the tape into the drive to the time when the tape is "ready" (i.e., the special file for the drive can be opened and operations performed without incurring I/O errors).
- **Seek time**: This is the time from when a seek command is issued to the time when the seeked-to data block can be read into memory (the seek system call might return before the read operation can be initiated). We measure three particular types of seeks:
  a. **Long seek from beginning of tape**: We measure the time to seek to an arbitrary location in the tape.
  b. **Long seek from the middle of the tape**: We measure the time to seek from one arbitrary location on the tape to another arbitrary location. Since this requires O(B^2) measurements (where B is the number of tape blocks), we pick representative locations in the middle of the tape.
  c. **Short seek from the middle of the tape**: A seek is expensive to initiate on most tapes. The behavior of a seek for a short distance can be very different from that for a long seek.
- **Transfer rate**: This is the rate (Mbytes / second) at which the tape drive will service read or write requests. This rate can be influenced by the compressibility of the data, the record size, and by the time between successive requests to for tape reads (writes).

- **Unmount time**: This is the time from the request to when the tape can be extracted from the drive by the robot arm.

While we tried to make our measurements as consistent as possible from platform to platform, we needed to take special measures for some of the devices. We tested the devices on a wide variety of platforms, each with its own local environment. In all cases the tape drive is attached to a SCSI bus. Also, some devices have special characteristics such as compression, seek location hints, partitioning, etc. Lastly, we had access to some devices for only a limited time.

5. Tape Systems Tested

We tested a wide variety of tape systems ranging from low to high performance (and cost) and with a wide range of characteristics listed in Table 1. In this section, we discuss the basic characteristics of each drive.

5.1. 4mm DAT

The 4mm DAT drive is a low-cost, low-performance drive, in common use for backup and data transfer. It uses helical scan recording, and comes in a cartridge. However, the cartridge does not have landing zones, so the tape must be rewound before unmounting. We measured a data transfer rate of 0.325 Mbytes per second, independent of block size. We were able to measure an average mount time of 50 seconds with a standard deviation of 0.0, and an unmount time of 21 seconds with a standard deviation of 1.3.

5.2. DLT 4000

The DLT 4000 is a moderate-cost, medium performance tape in common use for backup and archiving. The DLT 4000 is a serpentine tape that is packaged in a cartridge. The drive supports automatic data compression; however, all of our experiments were carried out with tape compression turned off. Although the DLT 4000 supports variable block sizes, the SCSI interface limited the maximum writable block size to 64 Kbytes. We found that the transfer rate did not depend strongly on the block size. For block sizes between 16 and 60 Kbytes, the transfer rate is 1.27 Mbytes per second, and the transfer rate declined for larger and smaller blocks.

The software environment that we used to measure the DLT 4000 did not allow us to measure the mount and dismount times directly because the volume management software would load a requested tape, and return when the tape is mounted. If all drives were full, the volume management software would first unload a drive and return the tape to the shelf. To get around this problem, we submitted requests to load tapes when the drives were either all full or all empty. By subtracting the estimate of 9 seconds (from the Storagetek 9710 specifications) to perform a tape fetch using the robot arm, we found that mounting a DLT 4000 tape requires 40 seconds and unmounting a DLT 4000 tape requires 21 seconds. These values are in line with the DLT 4000 performance specifications.

5.3. Ampex DST 310

The Ampex DST310 is a high performance helical scan tape drive that uses a tape cassette. The tape can be formatted with landing zones, eliminating the need to rewind before an unload. The tape also can be formatted with multiple partitions, but they are intended for data management — keeping files together, allowing a partial tape erase, etc. An unusual feature of the Ampex is the availability of "high-speed positioning hints". These hints are returned from the `get_pos` query and can be used in subsequent seek commands. The logical block size is 8 Kbytes, and all data transfers must be made in multiples of 8 Kbytes.

While the recommended transfer size for the Ampex is 4 Mbytes, we tested the read and write transfer rates with a variety of transfer sizes, and found that for transfer sizes of 1 Mbyte and larger, we achieved a throughput of about 14.2 Mbytes/sec. However, the transfer rate declined rapidly for transfer sizes smaller than 1 Mbyte. The average mount time is 10.1 seconds, with a standard deviation of 0.63. The unmount time is more complex. If an unmount is requested without rewinding the tape, the tape is moved to a system zone and then unmounted (this is done to protect the tape), so variance in rewind time becomes variance in unmount time.

5.4. Sony DTF

The Sony DTF is a high-performance tape drive. It uses a helical scan data layout and is packaged in a cassette. The cartridge can be unmounted without rewinding, but only when the current position is near the end of tape. In our system, however, the default command to eject the tape always rewound the tape completely. The Sony DTF supports variable size blocks and compression. We measured an average of 51 seconds to mount a tape, with a standard deviation of 0.0, and an average of 17.8 seconds to unmount a rewound tape, with a standard deviation of 1.2. We achieved a transfer rate of 12 Mbytes/sec with a block size of 512 Kbytes.

5.5. IBM 3590

The IBM 3590 is a high performance tape drive. It uses a serpentine data layout and is packaged in a cartridge. The drive supports variable size blocks and compression. We measured a transfer rate of 8.9 Mbytes/second using a block size of 512 Kbytes and a SGI host; this rate was the same for reads and writes.

6. Benchmark Results

In this section, we discuss the more interesting benchmark results gathered on the tape drives listed in Section 5. While transfer rates were relatively uninteresting (except for their relationships to the values quoted by the manufacturers), both seek times and mount/unmount times provided interesting comparisons. In particular, we focus on both long and short seeks and show that, often, reading can be faster than seeking to a nearby position.

6.1. Mount and Unmount

As the discussion in Section 5 indicates, mount and unmount times are nearly deterministic when the tape is rewound before unmount. However, the Ampex drive is different from the other drives we measured because we were able to unload the tape without rewinding it. If an unmount is requested without rewinding the tape, the tape is moved to a system zone and then unmounted (this is done to protect the tape). Variance in rewind time then becomes variance in unmount time. We tested the unmount time by seeking to a random location on the tape and then unmounting, as shown in Figure 1. The effect of the system zones can be seen in the sets of two parallel lines, offset by about 6 seconds that appear in the data. The average unmount time is 12.24 seconds with a standard deviation of 3.1 seconds.

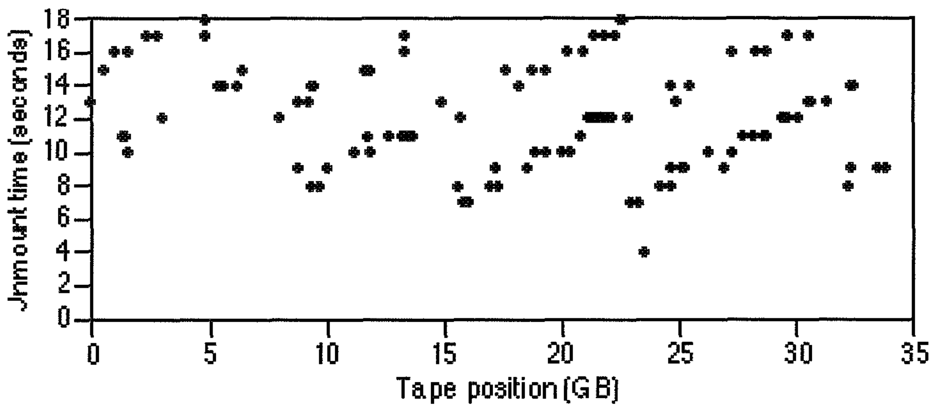


Figure 1. Unmount times for the Ampex 310.

If an Ampex tape is unmounted without being rewound, the first seek time increases (as is shown in Section 6.2). Because the seek and rewind times on the Ampex are so fast, rewinding a tape before unmounting reduces access times on average. We ran an experiment of repeatedly mounting a tape, seeking to a random location, reading 1 block of data, then unmounting returning the tape. We collected 60 data points for the case of rewinding before unmounting, and 60 data points for the case of unmounting without a rewind. If we rewound the tape before unmounting, then a fetch/return cycle takes 71 seconds with a standard deviation of 13. If we unmounted the tape without a rewind, the fetch/return cycle takes 85 seconds with a standard deviation of 30. A difference of means test indicates a significant difference between the two quantities.

6.2. Seek and Rewind Times

Because tapes are sequential media, they have large seek times. Overcoming seek time delays is a major focus of system optimization research. However, seek times on tapes can exhibit unexpected behavior. In this section we measure and model three types of seeks: the first seek after a mount (usually, but not always, seek from beginning of tape), a seek from the middle of the tape, and a short seek.

6.2.1. First Seek From Mount

For most tapes, “first seek from mount” is equivalent to “seek from beginning of tape” because the tape can only be unmounted (and thus mounted) when it has been rewound. This is true for all of the tapes in our study except for the Ampex; however, we will also report seek from beginning of tape for it.

The seek time from beginning of tape (BOT) for all of the tape devices we measured is shown in Figure 2. These charts also show the rewind time. As expected, seek times track rewind times well. For the helical scan tapes, both seek and rewind are linear in the distance the tape must travel.

The serpentine tapes, however, exhibit more complex behavior. Because the IBM 3590 and DLT 4000 have many pairs of tracks running in opposite directions, the seek & rewind times for a single pair of tracks characterize the seek & rewind behavior for an entire tape. The IBM 3590 has 4 pairs of tracks, and the DLT 4000 has 32 pairs. Figure 3 shows a detail of the DLT 4000’s seek & rewind behavior, equivalent to one peak & valley (corresponding to a single forward and reverse track from the graph in Figure 2. This graph is similar to that of the IBM 3590, shown next to it for comparison; both show a piecewise linear seek time function along with the linear rewind function. Both serpentine tape drives use the two-dimensional topology of the tape as the primary mechanism for implementing a high-speed search. The fast seek speed is only 1.5 times and 2 times faster than the read speed for the DLT 4000 and the 3590, respectively. To attain high-speed search, the drives store the location of particular blocks in the tape’s directory. To implement a distant seek, the tape moves to the last known block position that occurs before the desired block using the high-speed movement and then reads the tape until the desired block is located.

The Ampex 310 can be unmounted without a rewind. When the tape is mounted again, it is positioned at the middle of the tape and therefore is closer to the desired first seek position. In Section 5.3, we also measured the time to perform the first seek after a mount. The block positions for the first seek were randomly selected. We recorded the block position at unload and the block position for the first seek, as well as the seek time. However, we did not find any correlation between the distance between the block positions and the seek time. Instead, the seek time seems to be correlated with the seek block position. Figure 4 shows the result of the experiment. The time to seek to a block if no rewind is done before a mount is considerably larger (i.e., 25 seconds larger) than the time to perform a seek on a tape that has been rewound. Since most rewinds take less than 25 seconds (64% of the tape), we found that it is faster on average to rewind tapes after use.

6.2.2. Seek Times from Mid Tape

For the helical scan tapes, seek times between distant blocks in the middle of the tape fit well to a linear function. Figure 5, which shows seek times to and from a fixed block position on a 4mm DAT drive, is representative of helical scan drives.

However, seek times on a serpentine tape follow a more complex topology. The starting position divides the tape into four regions: before the starting point on a same-direction track, after the starting point on a same-direction track, before the starting point on a

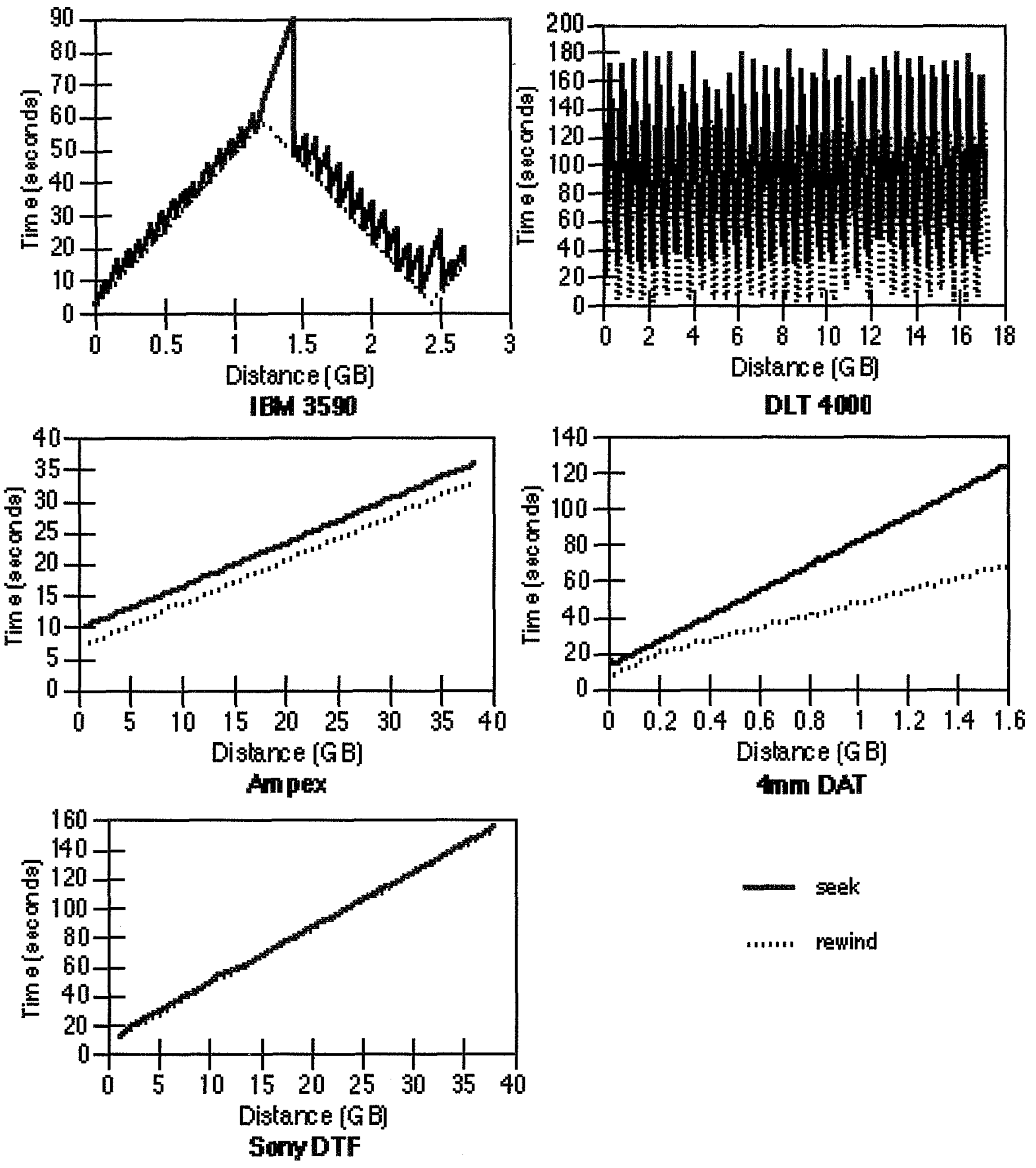


Figure 2. Seek and rewind times for various tape drives.

reverse-direction track, and after the starting point on a reverse-direction track. Figure 6 shows seek times from a tape block 73% into a reverse track of a DLT 4000 tape to other nearby track positions. The peaks of the seek time curve are offset from the track ends. After examining a number of these seek time curves, we found that the sizes of the offsets are reasonably stable. However, it is important to note that the seek times are non-linear, and that they indeed fall into the different categories mentioned above. This difference must be taken into account when laying out files that will potentially be accessed together.

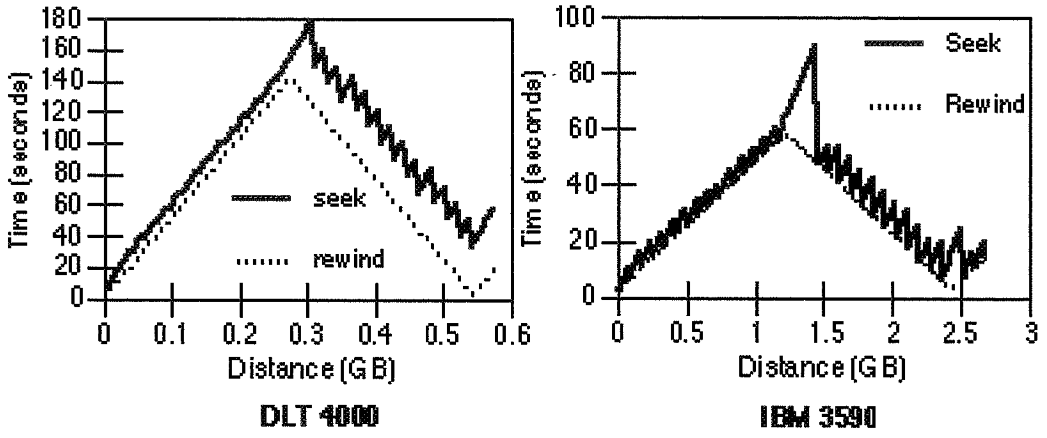


Figure 3. Detail of seek times for linear tape drives DLT 4000 and IBM 3590.

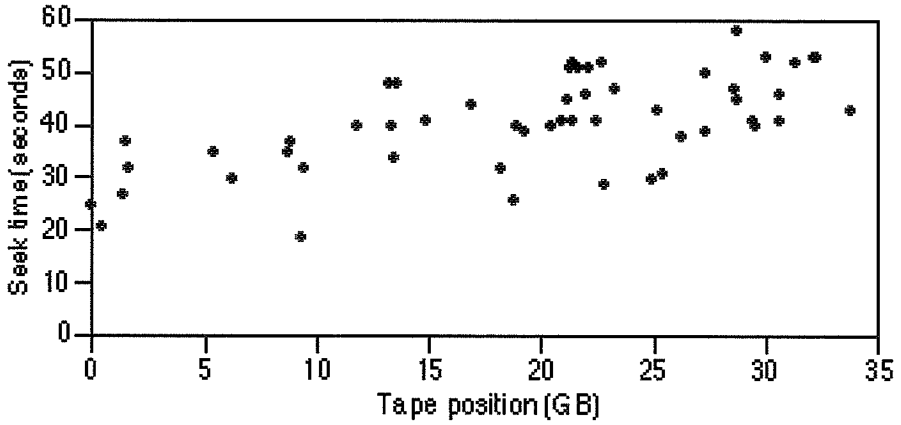


Figure 4. Seek times for Ampex 310 for tapes ejected without rewinding

6.2.3. Short Seeks

The increasing interest in building using tertiary storage in a more active role in both traditional mass storage systems and newer applications such as databases points to the need to investigate the performance of short seeks on tape. Data placement, indexing, and sizing algorithms have been developed with the assumption that seek time is directly proportional to seek distance. Often this is not the case. Furthermore, the seek time function over a short distance can be significantly different from the seek time function over a long distance. Because a tape seek often incurs a substantial delay, an important characterization is the distance at which it is faster to seek to the desired block position than it is to read in the unnecessary blocks.

We ran a set of experiments where we would repeatedly read in K blocks, and another set of experiments where we should seek past K-1 blocks and read in one block. The results of our short seek tests on all of the tape drives are shown in Figure 7. As the graphs show, serpentine drives differ markedly from helical scan drives in short seeks. This difference comes from the way that helical scan tapes must wrap the tape around the tape heads after

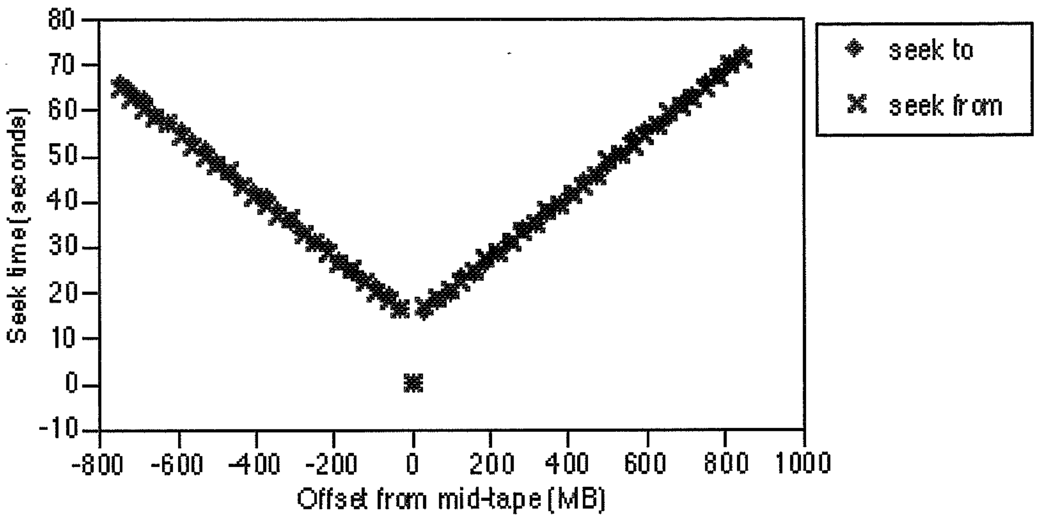


Figure 5. Mid-tape seeks on an 4mm DAT.

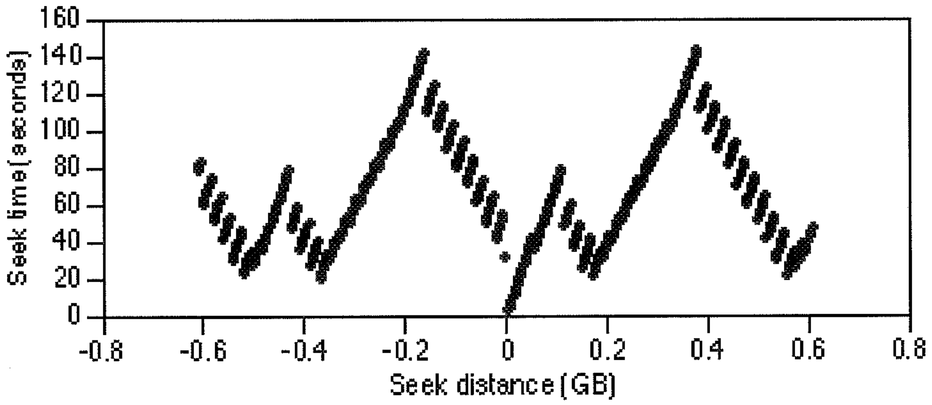


Figure 6. Mid-tape seeks on a DLT 4000, starting from 1 GB into the tape.

a high-speed seek. This wrapping process takes time, so it is often faster to simply read the intervening data than it is to perform a seek on helical scan drives.

For serpentine tapes, on the other hand, it takes almost no time to switch from high-speed tape transport to reading. As a result, seek and read times are close together. Additionally, the serpentine tape drives that we tested were sufficiently smart to pick the fastest seek method (high-speed scan or read) when a seek was requested.

7. Implications for Mass Storage System Designers

Our experiments provide several major benefits for mass storage system designers. First, they provide accurate (and unbiased) performance measurements of several modern tape drives, as reported in Section 6. Second, they highlight the performance differences between modern, high-performance tape drives using the competing technologies of helical scan and serpentine recording; in this section, we detail the implications these differences have for mass storage systems. Third, the benchmarks we used to gather the

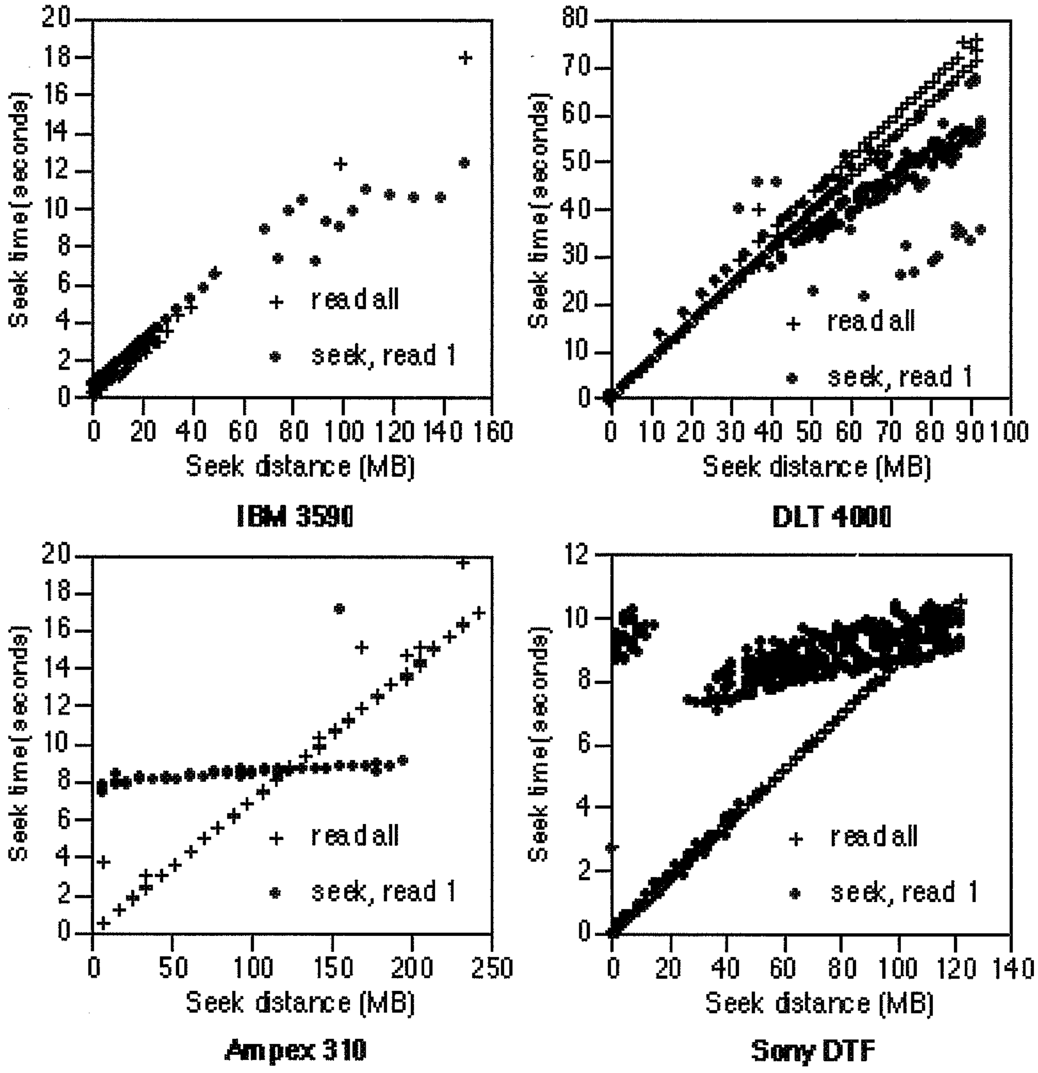


Figure 7. Short seek times for various tape drives

measurements can be run on newer tape drives, enabling others to do the benchmarking we reported in this paper.

7.1. Helical Scan vs. Serpentine Tape Drives

Our benchmarks uncovered several important performance differences between helical scan and serpentine tape drives that have implications for those building mass storage systems. While these differences are intuitively obvious, few (if any) mass storage systems take them into account.

7.1.1. Data Layout on Tape

The first major difference between the two types of tape recording is in the arrangement of data tracks. While maximum seek times for helical scan and serpentine drives are comparable, the seek profile from beginning of tape is not. Helical scan systems always require longer to reach block N + X than they do to seek to block X from the start of the tape. However, serpentine tapes can often reach block N+X more quickly because of the arrangement of data on the many forward and reverse tracks.

As a result, designers of mass storage systems must take these differences into account in two ways. First, seek planning should consider track arrangement. It is not always faster on a serpentine tape to read sparsely stored data from a tape in "standard" order because this may involve many back-and-forth traversals of the tape. Instead, an algorithm such as the SCAN or CSCAN disk seek algorithms [28] should be used. Hillyer and Silberschatz have done some introductory work on applying such scheduling to tape systems [23], but there is still much to be done.

We also believe that data placement algorithms should take track arrangement into account. Large files should be placed at the physical start of track because the seek time to them will be considerably shorter. Even if this is done at the cost of wasting small amounts of space, the savings in seek time and thus response time to requests will balance the relatively small cost of purchasing additional tape media.

It is important to note that these layout optimizations apply only to serpentine tape; however, they may serve to make serpentine tape more attractive to mass storage system users by reducing the average seek time to the start of the data.

7.1.2. Seeking vs. Reading

Most current mass storage systems issue seek requests whenever they need to advance to new location on a tape. As the experiments in Section 6.2.3 show, however, this method results in significantly longer seek times than simply reading the intervening data.

This problem can be addressed in two ways. Systems using current tape drives should use the profile data presented in this paper (or other data gathered in similar ways) to compute whether seeking or reading to the desired location is faster. This phenomenon occurs not just for short seeks of a megabyte or two, but over seek distances of 100 MB or more; thus, storage systems that do not take this seek profile into account will have worse response time than those that do.

However, the optimal solution to this problem is for tape manufacturers to incorporate this knowledge into their tape systems. The seek profiles are relatively simple, allowing the tape drive itself to compute whether "fast seek" or reading is a faster way to reach the destination. Even if this is done, though, mass storage system designers must know what the seek profile is to optimally lay out their data.

7.2. Tape Benchmarks

The benchmarks we used to gather the data in this paper will also be very useful for those designing mass storage systems and those who build tape systems. Our benchmarks provide an extensive profile of tape drive performance for a wide variety of tape drives, better enabling storage system designers to optimize access.

7.2.1. Performance Quirks

Our benchmarks can uncover performance "hiccups" in a tape system, as they did for one of the tape drives covered in this paper. The poor seek performance near the start of the reverse track on the IBM 3590 is the result of a bug in the tape's microcode, which has since been fixed by IBM (though our tape drive was not updated before the tests were run). Nevertheless, the IBM 3590 still performs relatively poorly on seeks to the start of a reverse track even with the microcode fix applied. To work around this problem, we would suggest not starting files in this area and instead placing "dummy" data there. This optimization would reduce seek time to all files on the tape at relatively little overhead in space.

Other performance quirks can similarly be worked around if mass storage system designers know of their existence. Both the IBM 3590 and the DLT 4000 have a list of locations to which they can "fast seek." Mass storage systems that know the locations in the list could optimize file placement to minimize the seek time to most files on tape, not just the few that are placed near the physical end of a tape.

While helical scan tape drives do not share either of the above performance issues, they have different issues, as was discussed in Section 7.1. Knowledge of the crossover point where seeking becomes faster than reading will allow MSS designers to optimize access to random locations on tape, thus improving performance.

It is important to realize that many features implemented to improve performance may not actually do so. For example, mid-tape unmounting on the Ampex 310 is (presumably) intended to provide a method of optimizing performance by reducing rewind time; however, our experiments show that mid-tape unmounting is actually slower than rewinding before each unmount when the tape is formatted with the default number of system zones. Also, fast unmounts followed by slow mounts may be desirable in some environments such as real-time recording. Overall, however, performance "enhancements" should be benchmarked to ensure that they do indeed improve performance.

7.2.2. Benchmarking Issues

Our experiments also uncovered difficulties in gathering the performance data necessary for this study. Most of these were related to the software interface between the tape system and higher-level software. It is important for those benchmarking tape systems to understand the potential traps of gathering performance data.

The most significant problem we experienced was that some software "lied" about when a tape was actually ready to execute the next command. In many cases, opens, seeks and rewinds returned before the tape was positioned properly; as a result, the ensuing read would seem longer than it should be. Designers benchmarking other tape systems should be careful of this problem.

Another problem we experienced was that it was impossible to use a single program to conduct all of the benchmarks because of the differences in the interfaces between the many tape systems. One workaround for this problem was to use Perl rather than C for some of the benchmarks; while this made coding easier, we still had to modify the code to handle different tape systems on different platforms.

Discovering the “true” tape block size was also a difficult task. Some tape drivers accepted tape “blocks” larger than a certain amount, but chopped the blocks into smaller sizes for writing to tape. If the benchmark is trying to track changes in performance as tape block size changes, it is important to make sure that the device driver is actually writing the data in the desired block size.

8. Conclusions

We took detailed measurements from five common tertiary storage tape drives, spanning the range of low to high performance. The drives included helical scan and serpentine data layouts, and cassette and cartridge tape packages. Based on the benchmarks that we ran on the tapes, we made suggestions for improving the performance of any system that uses tapes as an active data storage medium — traditional mass storage systems, scientific databases, and archival storage systems.

Our future work will include more detailed measurements of tape system parameters as well as analytic models for the time necessary to perform various tape operations such as seeking and reading. While the measurements in this paper provide a good foundation for mass storage system designers, we hope that simple formulas for seek time and other parameters will make optimization of data layout and access an easier task.

We also plan to take performance measurements of additional aspects of tertiary storage devices. Of particular interest are measurements of very large robotic storage libraries in which the variance in the time to fetch a tape can be large, and serpentine tape drives that support partitioned tapes.

Acknowledgments

We would like to thank all of the people who gave us access to the tape drives and helped get the experiments set up. They include Jim Ohler and Jim Finlayson at the Department of Defense, P. C. Hariharan, Jeanne Behnke, and Joel Williams at the Mass Storage Testing Laboratory at NASA Goddard, and Gary Sagendorf at the Infolab at AT&T Labs - Research. We are also grateful for the feedback and guidance provided by the program committee, particularly our paper shepherd.

References

[1] J. Myllymaki and M. Linvy, “Disk-Tape Joins: Synchronizing Disk and Tape Access,” *Proceedings of the 1995 ACM SIGMETRICS Conference*, 1995.

[2] D. Teaff, D. Watson and B. Coyne, “The Architecture of the High Performance Storage System (HPSS),” *NASA Goddard Conference on Mass Storage Systems and Technologies*, 1995, pages 45-74.

[3] L-F. Cabrera, R. Rees and W. Hineman, “Applying Database Technology in the ADSM Mass Storage System,” *Proceedings of the 21st Very Large Data Base Conference*, 1995, pages 597 - 605.

[4] D. A. Ford and J. Myllymaki, “A Log-Structured Organization for Tertiary Storage,” *International Conference on Data Engineering*, 1996.

[5] J. Kohl, M. Stonebraker, and C. Staelin, "HighLight: A File System for Tertiary Storage," *Proceedings of the 12th IEEE Symposium on Mass Storage Systems*, 1993, pages 157 - 161.

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

[7] L. Lueking, "Managing and Serving a Multiterabyte Data set at the Fermilab D0 Experiment," *Proceedings of the 14th IEEE Mass Storage Systems Symposium*, 1995, pages 200 - 208.

[8] J. D. Shiers, "Data Management at CERN: Current Status and Future Trends," *Proceedings of the 14th IEEE Mass Storage Systems Symposium*, 1995, pages 174 - 181.

[9] D. Schneider, "The Ins and Outs (and Everything Inbetween) of Data Warehousing," *Proceedings of the 23rd International Conference on Very Large Data Bases*, 1997, pages 1-32.

[10] M. Stonebraker, "Sequoia 2000: A Next-Generation Information System for the Study of Global Change," *Proceedings of the 13th IEEE Mass Storage Systems Symposium*, 1994, pages 47 - 53.

[11] P. Triantafillou and T. Papadakis, "On-Demand Data Elevation in a Hierarchical Multimedia Storage Server," *Proceedings of the 23rd Very Large Database Conference*, 1997, pages 226 - 235.

[12] R. A. Coyne and H. Hulen, "Toward a Digital Library Strategy for a National Information Infrastructure," *Proceedings of the 3rd NASA Goddard Conference on Mass Storage Systems and Technologies*, 1993, pages 15 - 18.

[13] D. Menasce and O. Pentakalos and Y. Yesha, "An Analytical Model of Hierarchical Mass Storage Systemswith Network Attached Storage Devices," *Proceedings of the 1996 ACM SIGMETRICS Conference*, 1996, pages 180 - 189.

[14] T. Johnson, "An Analytical Performance Model of Robotic Storage Libraries," *Performance '96*, 1996, pages 231 - 252.

[15] O. Pentakalos, D. Menasce, M. Halem and Y. Yesha, "Analytical Performance Modeling of Mass Storage Systems," *Proceedings of the 14th IEEE Mass Storage Systems Symposium*, 1995.

[16] T. Nemoto, M. Kitsuregawa and M. Takagi, "Simulation Studies of the Cassette Migration Activities in a Scalable Tape Archiver," *Proceedings of the 5th International Conference on Database Systems for Advanced Applications*, 1997.

[17] C. Ruemmler and J. Wilkes, "An Introduction to Disk Drive Modeling," *IEEE Computer*, March 1994, pages 17 - 28.

[18] A. Drapeau and R. H. Katz, "Striped Tape Arrays," *Proceedings of the 12th IEEE Mass Storage Systems Symposium*, 1993, pages 257 - 265.

[19] J. Myllymaki and M. Linvy, "Efficient Buffering for Concurrent Disk and Tape I/O," *Performance Evaluation*, volume 27, pages 453 - 471.

[20] L. Golubchik and R. R. Muntz and R. W. Watson, "Analysis of Striping Techniques in Robotic Storage Libraries," *Proceedings of the 14th IEEE Mass Storage Systems Symposium*, 1995, pages 225 - 238.

[21] D. Ford and S. Christodoulakis, "Optimal Placement of High-Probability Randomly Retrieved Blocks on CLV Optical Disks," ACM Transactions on Information Systems, **9**(1), 1991.

[22] B. Hillyer and A. Silberschatz, On the Modeling and Performance Characteristics of a Serpentine Tape Drive," *Proceedings of the 1996 ACM SIGMETRICS Conference*, 1996, pages 170 - 179.

[23] B. Hillyer and A. Silberschatz, "Random I/O Scheduling in Online Tertiary Storage Systems," *Proceedings of the 1996 ACM SIGMOD Conference*, 1996, pages 195 - 204.

[24] R. van Meter, "SLEDs: Storage Latency Estimation Descriptors," *Proceedings of the NASA Goddard Conference on Mass Storage Systems and Technologies*, March, 1998.

[25] R. Venkataraman, J. Williams, D. Michaud, H. Gu, A. Kalluri, P C Hariharan, B. Kobler, J. Behnke, and B. Peavey, "The Mass Storage Testing Laboratory at GSFC," *Proceedings of the NASA Goddard Conference on Mass Storage Systems and Technologies*, March, 1998.

[26] G. Hull and S. Ranade, "Performance Measurements and Operational Characteristics of the Storagetek ACS 4400 Tape Library with the Cray Y-MP EL," *Proceedings of the NASA Goddard Conference on Mass Storage Systems and Technologies*, 1993, pages 229 - 240.

[27] D. Therrien and Y. L. Cheung, "Using Magnetic Tape Technology for Data Migration," *proceedings of the NASA Goddard Conference on Mass Storage Systems and Technologies*, 1993, pages 241 - 256.

[28] R. Geist and S. Daniel, "A Continuum of Disk Scheduling Algorithms," ACM *Transactions on Computer Systems*, **5**(1), 1987, pages 77-92.

Scheduling Non-Contiguous Tape Retrievals

Bruce K. Hillyer, Avi Silberschatz
Bell Laboratories, Rm. 2A-310
700 Mountain Avenue
Murray Hill, NJ 07974
{hillyer,avi}@bell-labs.com
tel +1-908-582-2262
fax +1-908-582-4623

Abstract

Large data installations normally archive relatively inactive data to a near-line tape library. The tape library performs reasonably well for sequential-access retrieval workloads. However, if the retrieval access slices across multiple data sets, or makes retrievals from scattered portions of a large data set, then the performance can suffer drastically.

In this paper, we use modeling and simulation to study several scheduling algorithms for random-access retrievals from tape. This study is based on extensive measurements of the IBM 3570 Magstar MP tape library, which is designed for fast random access. We study the retrieval performance of the Magstar MP as a function of the request size and the queue length (i.e., retrievals per tape switch). The following data point illustrates the performance. Averaged over many trials, the execution time of a schedule of 4 random retrievals from one tape (each retrieval transfers 12 MB) is 98 seconds, including the overhead of rewind, unload, eject, tape switch, and load. This schedule gives an average data rate of 0.5 MB/s, which is comparable to the average data rate of a magnetic hard disk that is fetching randomly-located pages.

1 Introduction

The traditional design of a large-scale storage system is hierarchical, using magnetic hard disks for hot data, and a tape library to archive cold data. This design reflects constraints such as cost, and practical limitations on the number of magnetic hard disks in a system. A hierarchical storage system is well-suited to scientific supercomputing installations that do not have excessive demands for retrievals from the tape archive.

In recent years, we have witnessed the emergence of application classes in which archive-retrieval workloads no longer consist of a few large sequential requests. In commercial settings, on-demand retrievals of archived documents, such as claim forms, check images, and account histories, generate a random-access workload to the tape archive. Web access to public sites can induce large numbers of concurrent accesses to an archive. Even in scientific environments, we see examples of access to widely-scattered portions of large data sets, and slicing across multiple data sets to retrieve relatively small portions. In these cases, staging in all the data to disk may be inefficient. Although tape is not considered to be a random-access medium, these examples suggest that we cannot ignore the problem of numerous, relatively small accesses to a tape library.

In previous work, we modeled the positioning time of the Quantum DLT4700 tape library [1], and studied the performance of scheduling algorithms for random retrievals from the DLT [2]. For random retrievals on the DLT, sophisticated scheduling and very large I/Os are necessary for good performance, because of the large positioning times and tape switch times. (By "tape switch time", we mean the sum of the times required by the tape drive and library to unload and eject the old tape, swap the old tape for another one, load the new tape into the drive, and wait for the drive to become ready to use the new tape.) Even with good scheduling and large I/Os, the random-retrieval performance of the DLT is relatively insensitive to the streaming bandwidth of the drive, and is largely determined by the positioning time.

In this paper, we examine the performance of the IBM 3570 Magstar MP tape library, which is designed for high-speed positioning and tape switches. The Magstar MP uses a modified serpentine track layout. As with other serpentine tape units, it is difficult to predict the positioning time between two blocks on the tape. It is not a simple function of the logical block numbers, and it is not strictly proportional to the physical distance between the blocks.

In section 2, we give a model of locate time for the Magstar MP. In measurements of 55,000 random locates, this model is accurate to within 1 second in 96% of the cases. In section 3, we describe several algorithms to schedule an efficient retrieval order for several requests from one tape. In section 4 we use our locate-time model for the Magstar MP to evaluate the retrieval performance of these algorithms. For uniformly-distributed random requests, we characterize the effective data rate of the Magstar MP as a function of the I/O request size in MB and the number of requests per tape switch. In section 5 we give concluding remarks.

2 A Model of Locate Time

A scheduling algorithm chooses a retrieval order for a set of requests to minimize the overhead of positioning the tape head from one block to the next. To develop such an algorithm, we need the ability to predict the positioning time between arbitrary blocks on the tape. In this section, we develop a formula that predicts the positioning time. Because the `SCSI LOCATE` command is used to position the tape, the model is called the *locate-time model.*

To begin, we describe the data layout used by a Magstar MP tape, and show a graph of locate-time measurements. We observe that the slope of this graph is discontinuous, and we describe algorithms to determine the "key points" at which these discontinuities occur. These key points are the input parameters of a formula that predicts the locate time between any two blocks on a Magstar MP tape. We describe the formula for the locate time of the Magstar MP, and show the results of experiments that validate the accuracy of this model.

Before ejecting a tape, most drives rewind to the beginning of the tape. The IBM

Magstar MP is different. It is designed to rewind to the middle of the tape, to a position called the *load point*.

The Magstar MP uses a modified serpentine layout. This layout first covers one half of the tape with 32 serpentine "half tracks", called *wrap halves*, that run between the load point and one end of the tape. Then the head crosses over the load point to the other half of the tape, where 32 more wrap halves are written in a serpentine pattern.

The nominal capacity of a Magstar MP tape is 5 GB. (In this paper, the terms KB, MB, and GB denote 2**10, 2**20, and 2**30 bytes, respectively.) The effective capacity of a tape depends on the compression ratio of the data, and also on the number of defects on that particular tape. For our experiments, we filled tapes with blocks of size 32 KB. Each block contained pseudorandom bytes with the high bit of each byte masked off. The drive's data compression hardware should be able to reduce this kind of data to about 7/8 of its original size. Over a sample of 4 tapes, the number of 32 KB blocks per tape ranged from 170878 to 180418 (5.6E9 - 5.9E9 bytes). The time to write a tape ranged from 45m58s (45 minutes + 58 seconds) to 50m37s. Thus, the effective writing rate ranged from 1.79 to 1.95 MB/s (i.e., slightly more than 2 million bytes per second).

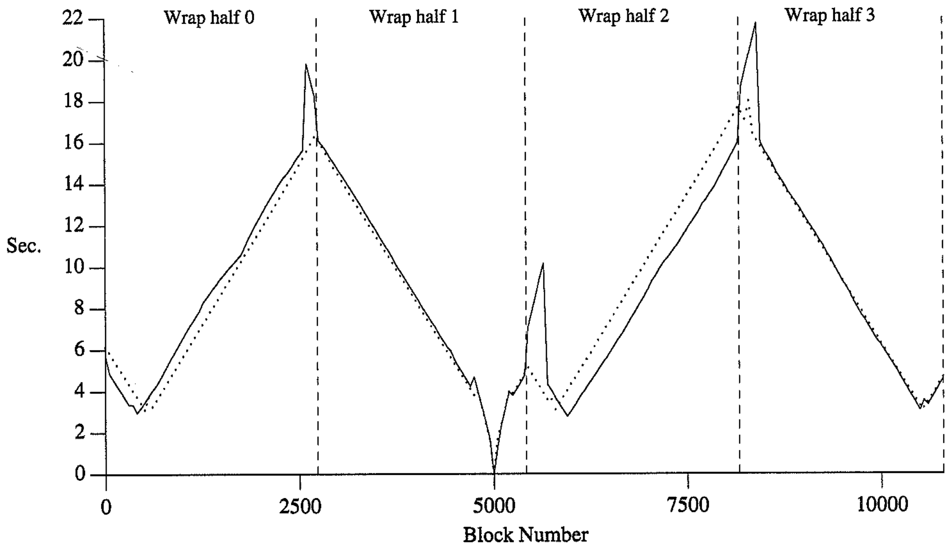


Figure 1: Locate Time from Block 5000 to Blocks 0–10,800.

Figure 1 shows a typical graph of locate-time measurements, taken on an IBM 3570 Magstar MP tape library running microcode version D1I3_27D. The host computer is a Sun Sparc-20 model 61 running Solaris 2.3. We perform the `SCSI LOCATE` command via the Solaris `USCSI ioctl()`, which enables a sufficiently privileged application to send an arbitrary SCSI command to a target. For each measurement in this graph, the starting position is logical block 5000, which is in wrap half 1, about 2/3 of the distance from the outer end of the tape back toward the load point. An experiment measured the locate time from block 5000 to blocks 0, 50, 100, 150, ..., 10800. These measurements are shown by the solid line. The reverse measurements, returning back to block 5000, are shown by the dotted line. The graph shows several interesting aspects of the Magstar MP locate time.

- If the destination of the locate is on the same wrap half as the starting point, and fairly close (i.e. within about 1/20 of the length of the tape), the locate happens at the speed of tape reading, i.e., about 2 MB/s.

- If the destination of the locate is on the same wrap half as the starting point, but further away than 1/20 of the tape, the locate happens at a higher tape transport speed.

- If the destination of the locate is physically very close to the starting point, but on a different wrap half, the time required to switch tracks and locate to the destination is nearly 3 seconds.

- If the destination is on the far side of a wrap turn, but within about 1/20 of the length of the tape, the locate requires an extra 2 seconds. It appears that the drive locates at high speed to the wrap turn, does some housekeeping for about 2 seconds, and then reads into the next wrap half. In the worst case, this behavior takes about 4 seconds longer than would be required for a high-speed locate directly to the destination.

- The graph of locate time is nicely linear, except for the extra overhead beyond each wrap turn, and the transition from normal speed to high-speed positioning. Broadly speaking, the locate time is proportional to distance traversed. (Please do not read too much into this statement. The locate time is not a simple linear function of the distance between the source and destination. In particular, given 1000 pairs of randomly-chosen logical blocks `A` and `B`, our measurement of `locate_time(A,B)` differs from `locate_time(B,A)` by more than 30% in 100 cases, and by more than 50% in 50 cases.)

From the graph, it is clear that the key points that parameterize the locate-time model will be the block numbers of the wrap turns, and the block numbers of the spikes about 200 blocks to either side of each wrap turn. Since the Magstar MP has 64 wrap halves, a total of 192 points (3 * 64) characterize the entire tape. Unfortunately, Magstar MP microcode D1I3_27D provides no documented way to ask the drive to state the logical block numbers of these key points. Nevertheless, the wrap turns can be found relatively easily, because the `SCSI REQUEST SENSE` command for this drive is documented to return the wrap half number in a field in the sense data. Thus a simple program can binary search for the exact block number of each wrap turn. First we find all the wrap turns near the load point. Next we find all the wrap turns at one end of the tape. Finally, we find all the wrap turns at the other end of the tape. A program to do this runs in about 20 minutes. Our initial attempt at a locate-time model for the Magstar MP was parameterized only by the 64 wrap turns. We were disappointed with the inaccuracy of this model: If we guess incorrectly about whether the destination is in the slow region beyond a wrap turn, we can incur an error of 4 seconds.

The locate-time model that we have developed is also parameterized by the logical block numbers of the spikes to either side of each wrap turn. To find these blocks, we measure locate times from the other side of the wrap turn, looking for the precipitous drop in locate time beyond the spike. It takes between 1.5 and 2 hours to run a program that characterizes a tape by measuring all 192 key points. C code (580 non-commentary source lines) is available at http://www.bell-labs.com/ hillyer/papers/analyze_3570.c for the body of a completely unsupported program that determines the key points of a 5 GB Magstar MP tape filled with 32 KB blocks.

The locate-time model is an ad-hoc collection of cases that describe a behavioral model of how the drive could be positioning the tape between any two logical blocks. We do not claim that the drive actually performs the tape motions specified in the model, but we do claim that the model predicts most locate times well. The C code for a completely unsupported implementation of this model, specialized for the 32 KB blocksize that we used, is available in http://www.bell-labs.com/ hillyer/papers/locate_model_3570.c (393 non-commentary source lines). A general outline of the model is given in the next paragraph—a full discussion of all the special cases is not worthy of the space that would be required.

The model is given a source logical block number and a destination logical block number. For each, the model searches the table of 192 key points and interpolates to determine which wrap half contains the block, and whether the destination is in the expensive region just beyond a wrap turn. If the source and destination are in the same wrap half, then the locate time is given by one of two linear functions: one models the reading speed used for nearby locates (slope 0.015 seconds per block, y-intercept 0.95 seconds), and one models high-speed tape positioning (slope 0.006 seconds per block, y-intercept 2.11 seconds). The model states that high-speed positioning is used for tape motion further than 230 blocks within one wrap half. Note that all of these constants depend on our 32 KB block size and the particular compression ratio of our data, so they would need to be scaled for general use.

If the source and destination are not in the same wrap half, the model uses the interpolated positions derived from the logical block numbers to calculate the time to traverse the distance at high speed, and adds appropriate amounts of time if the destination is in the expensive region just past a wrap turn, and also if the tape head crosses the load point. In the model, the time cost for a destination in the wrap-turn region is the high-speed time to the wrap turn near the destination, plus 1.6 seconds, plus the slow-speed time from that wrap turn to the destination block, and the time penalty for crossing the load point is 2.4 seconds. In addition, we can see an asymmetry in Figure 1. If the destination is in an in-bound half wrap, i.e., an odd-numbered half-wrap, an extra 0.75 seconds are required.

The model was developed to fit a random walk consisting of a sequence of 3000 locates on one particular tape. After we were satisfied with the accuracy of the model, we tested it by comparing its predictions with locate-time measurements on 4 tapes. We tested a sequence of 10,000 locates on one tape, 5000 locates on a second tape, 20,000 locates on a third tape, and 20,000 locates on a fourth tape. In all four tests, 96% of the locate times were predicted with an accuracy of 1 second or better. For each test, 6 or fewer locates had errors between 2 and 10 seconds. The remainder had errors between 1 and 2 seconds.

3 Scheduling Random Retrievals from Tape

This section first describes a collection of algorithms to schedule a set of retrievals from a single tape, and then mentions a way to schedule a set of retrievals distributed over multiple tapes. The single tape scheduling algorithms are as follows.

One no-effort schedule is FIFO, which simply retrieves the requests in whatever order they are given.

Another no-effort schedule is READ, which reads the entire tape sequentially, discarding blocks that are not in the request list. For the Magstar MP, we found the READ schedule to be the best choice when more than 1200 blocks of size 32 KB are to be retrieved from a single tape.

The SORT schedule is formed by sorting the requested logical block numbers into ascending order. This algorithm works well for a helical-scan tape or a 9-track tape, because the drive will satisfy the requests in a single sweep over the requested blocks. For a modified serpentine tape, such as the IBM 3570 Magstar MP, the sorted schedule will first satisfy all the requests on one side of the load point, and then all the requests on the other half of the tape. For a small number of requests, this approach is a little better than FIFO. For a large number of requests, the SORT schedule converges to the READ schedule.

The SCAN schedule sorts the requests by estimated physical position on the tape, using the table of key points (in particular, the logical block numbers of the wrap turns). The retrieval pattern starts at the load point, sweeps toward one end of the tape in a single pass, then reverses to sweep toward the other end of the tape, and finally back to the load point. In our experiments, we only retrieved blocks when the tape head was moving from the load point outward. A slight variation that we did not test would partition the requests into out-bound (i.e. odd-numbered) wrap halves and in-bound wrap halves. The head would follow the same path as described above, but out-bound requests would only be retrieved when the head is moving outward from the load point, and in-bound requests would be retrieved while the head is moving toward the load point.

The shortest latency time first (SLTF) schedule is the simple greedy schedule. It starts at the load point. From the set of all outstanding requests, it proceeds to the one having the shortest predicted locate time from the current position. Repeat until all requests are satisfied. For some graphs, the SLTF algorithm is too shortsighted: it greedily chooses the shortest edge now, even if that forces the use of a much longer edge in a later step.

OPT is the optimal schedule. We calculate it by using the model of locate time to predict the schedule execution time for every possible ordering of the requests: the fastest one is chosen. The number of possible orderings is exponential in the number of requests, so it is impractical to calculate the OPT schedule for more than 10 or 12 requests.

To approximate OPT for a larger number of requests, we can use the LOSS algorithm, which is a classical heuristic algorithm for the asymmetric traveling salesman problem. (The traveling salesman problem seeks to find the shortest route that passes through a set of cities: here each request is a city, and the locate-time model gives the "distance" between any two cities. The distance is asymmetric in that `locate_time(A,B)` is often significantly different from `locate_time(B,A)`.) The LOSS algorithm is presented in [3], and is also described in some detail in [2], where it is used to schedule retrievals from a Quantum DLT4000 tape. The results in section 4 will show that the LOSS algorithm for the Magstar MP does not generate significantly better schedules than SLTF or SCAN. For this reason, we only give a sketch of the algorithm here.

The basic idea of the LOSS algorithm is as follows. The algorithm processes a graph consisting of cities with directed edges between them. The major cycle of the algorithm searches for one pair of cities to be connected in the schedule. The major cycle is repeated until all the cities are connected. Within a major cycle, the algorithm considers every remaining city, and calculates the difference between the closest and second closest neighbor (separately for inbound edges and outbound edges). This difference, which is called the "loss" for the city, is a lower bound on how much worse the schedule will become if the schedule doesn't use the edge to the closest neighbor. The algorithm finds the city having the greatest loss value, and chooses the edge to its closest neighbor to be part of the schedule. When an edge is chosen, all the other edges going out of the source are removed from the graph, and all the other edges going into the destination are also removed. As soon as a city has only one inbound edge and one outbound edge, that city is fully scheduled, so it need not be considered further.

For the Magstar MP, we have not quantitatively studied the scheduling of requests that are distributed over multiple tapes. But we can mention a result from our unpublished study of another tape jukebox. An intuitively appealing technique to schedule retrievals from a tape jukebox is as follows. Whenever a drive becomes idle, iterate the following three steps. Step 1: for each tape, calculate the best schedule to retrieve all outstanding requests for that tape. Step 2: switch to the tape whose schedule gives the highest effective transfer rate (bytes divided by time). Step 3: execute the schedule for that tape. The interesting result is that for uniformly-distributed random requests, the performance of this algorithm is very nearly equaled by a much simpler algorithm: switch to a tape having the greatest number of outstanding requests, and execute the best schedule for that tape. If it is necessary to prevent starvation of requests to unpopular tapes, we can process the tapes in rounds, such that all the tapes for all current outstanding requests must be processed before any new request is added to the schedule.

4 Random-retrieval Performance of the IBM 3570 Magstar MP

The model of locate time that is described in section 2 gives an accurate estimation of the locate time between any two logical blocks on a Magstar MP tape. Using this model, it is easy to estimate the total tape locate time for a schedule of requests: just sum the locate time from the load point to the first block, and from the end of each block in the schedule to the beginning of the next one. In this first experiment, we do not include the time to rewind to the load point after the schedule is completed, and we do not include time for a tape switch.

The basic experiment evaluates the effectiveness of the scheduling algorithms by calculating the average locate time per request, as a function of the choice of scheduling algorithm and the number of requests in the schedule (denoted N). For one trial, we generate a set of N random block numbers on the tape, use each algorithm to calculate a schedule, and then calculate the total locate time of each schedule. We divide the total locate time by N to obtain the average locate time per request. We repeat this process and calculate the average over many trials. For the OPT algorithm we average over 10,000 trials for N taking each value from 1–9, we average over 1000 trials for N = 10, and we average over 100 trials for N = 12. For FIFO, SORT, SCAN, SLTF, and LOSS, we use 10,000 trials for N ranging over 1–10, 12, 16, 24, 32, 48, 64, 96, 128, and 192; we use 8000 trials for N =

256; 4000 trials for N = 384; 2000 trials for N = 512; 1000 trials for N = 768; 500 trials for N = 1024; 250 trials for N = 1536; and 125 trials for N = 2048.

To confirm that the averages are calculated over sufficiently many trials to be repeatable, the above experiments have been repeated with three different pseudorandom number generator starting seeds. Changing the starting seed alters the average locate time by less than 1% in all cases.

For each algorithm, figure 2 shows the average number of seconds per locate, as a function of N, for a schedule that starts at the load point, and ends with the tape head just beyond the last request in the schedule. We see that for the Magstar MP, the SLTF and SCAN algorithms both work well. We also see that if more than 1200 randomly distributed blocks of size 32 KB are to be retrieved from one tape, the fastest option is to read the entire tape. (Recall from section 2, that for nearly noncompressible data, a tape holds approximately 175,000 such blocks.)

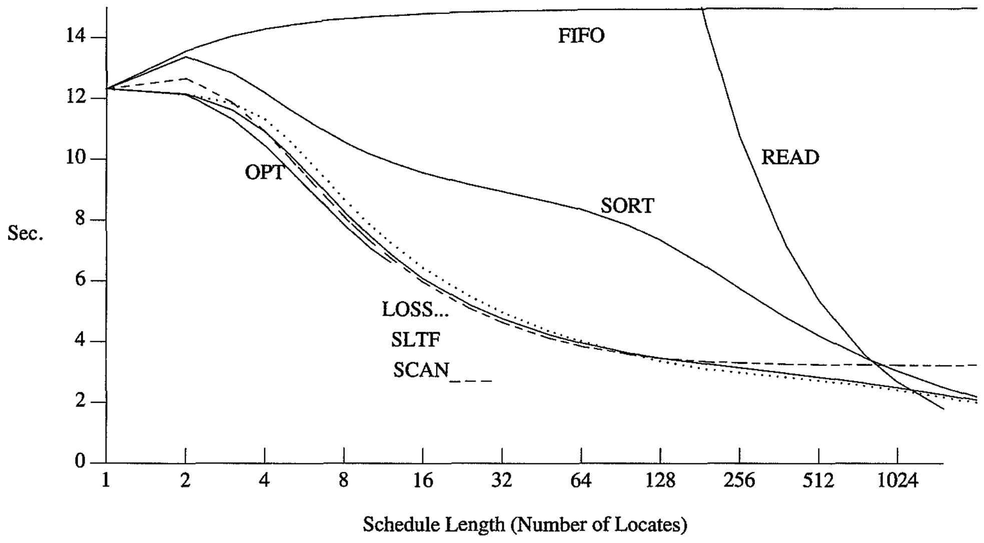


Figure 2: Average Locate Time as a Function of Schedule Length.

From the fact that the performance is nearly identical for SCAN, SLTF, and OPT, we conclude that on the Magstar MP, the speed of a schedule is largely a consequence of where the requests lie on the tape—an obvious order is likely to be a good order. This result is different from the results previously obtained for the Quantum DLT4700 tape library, for which the complex locate time enables a more sophisticated algorithm (LOSS) to give a substantial speedup [2].

The next experiment examines the additional time required when the schedule makes an excursion from the load point, through all the requested blocks, and back to the load point. For this experiment, the OPT schedule is calculated in three ways for comparison. The `OPT_no_rw` schedule is identical to the OPT schedule shown earlier: it starts at the load point, and ends just beyond the last block in the schedule, with no rewind. The second schedule, called `OPT_append_rw`, simply appends a rewind to the `OPT_no_rw` schedule. Since `OPT_no_rw` finds a minimal-time schedule through the last request, one may reasonably suspect that `OPT_append_rw` maximizes the distance of the final rewind. The third schedule, called `OPT_sched_rw`, actually calculates a minimal-time retrieval schedule that starts at the load point, proceeds through all the requests in any order, and then rewinds to the load point. Figure 3 shows the average locate time for these three variations of OPT, for schedules ranging from 1 to 10 requests. The averages are calculated over 10,000 trials for N = 1–6, 2000 trials for N = 7, 1000 trials for N = 8–9, and 100 trials for N = 10. As before, this experiment has been repeated with three different starting pseudo-random number generator seeds. For N = 10 the result varies by 2%; for the other values of N, changing the seed varies the result by 1% or less.

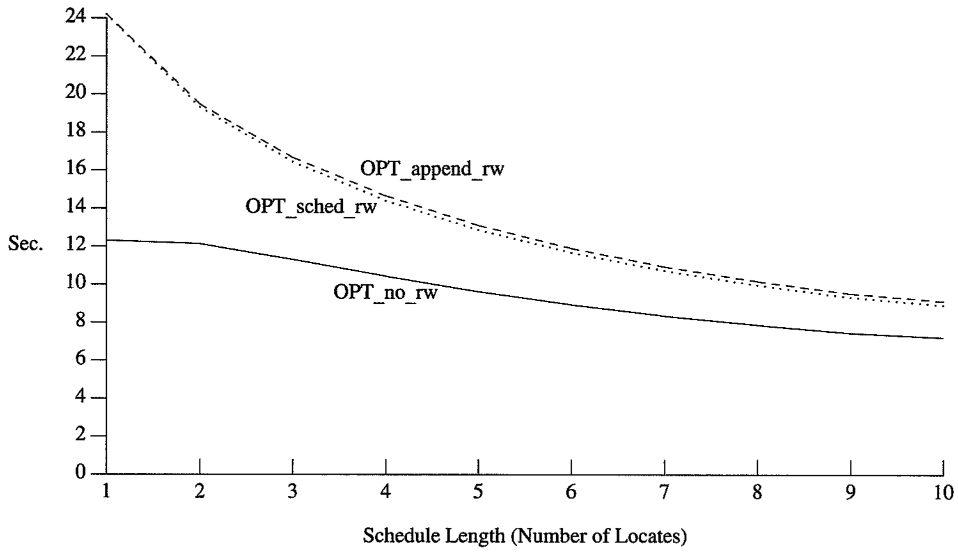


Figure 3: Average Locate Time when Appending or Scheduling a Rewind to Load Point.

From this figure, we see that appending a rewind to the schedule gives very nearly the same result as explicitly minimizing a schedule that ends with a rewind. Our explanation for this observation is that any schedule will need to traverse that subset of the tape that contains the requests—on the Magstar MP, the near-linearity of the locate-time function causes a good schedule to traverse that portion of the tape in a way that approximates a linear sweep. Whether that sweep performs I/O during the outbound trip or during the rewind is unlikely to have a large performance impact. The total rewind time increases with the schedule length, because with a larger number of random requests, there is a greater likelihood that some request will be near an outer edge of the tape, causing the rewind distance to approach its maximum.

The final topic that we study is the effective transfer rate (and hence drive utilization). We determine the relationship between the utilization and two factors: the length of a schedule (i.e., number of requests per tape), and the size of each request (i.e. number of bytes transferred by each retrieval in the schedule). The effective transfer rate is the total number of bytes retrieved, divided by the total retrieval time (the sum of the times to eject the previous tape, switch to the new tape, load that tape into the drive, locate and read through the blocks in the schedule, and rewind to the load point.) The drive utilization is the effective transfer rate divided by the streaming bandwidth. The locate time as a function of schedule length N is obtained from the best schedules in figure 2. Thus we use OPT for schedules of 1–12 retrievals per tape, SCAN for 16–96, LOSS for 128–1024, and READ for 2048–16384. The transfer time is calculated from the streaming transfer rate of 1.93 MB/s that we measured for our slightly-compressible data. The expected rewind time is obtained from the simulation output that supports figure 3. The rewind time ranges from 11.9 seconds for a schedule of length 1 through 17.1 seconds for a schedule of length 10. For longer schedules we use an approximation of rewind time that increases from 17.3 seconds through 22 seconds, which is the largest rewind time that we have measured. For these longer schedules, the rewind time is a small fraction of the total schedule time, so our approximation of the rewind time introduces an error of 2% or less. For the tape switch overhead (eject, switch, load) we use the value 16.3 seconds, as reported in [4].

Figure 4 presents a family of utilization curves for the Magstar MP, using our best scheduling algorithm for each schedule length, and including the overhead of a rewind and a tape switch for each tape schedule. A utilization of 100% would mean that the effective transfer rate equals the streaming data rate of the drive, about 2 MB/s for noncompressible data. The 25% utilization curve shows (for example) that the tape library has an effective transfer rate of about 0.5 MB/s when executing a random access workload that retrieves a single block of size 26 MB after each tape switch. The effective transfer rate is also 0.5 MB/s for a schedule that does a tape switch, retrieves 8 random blocks of size 8 MB, and rewinds.

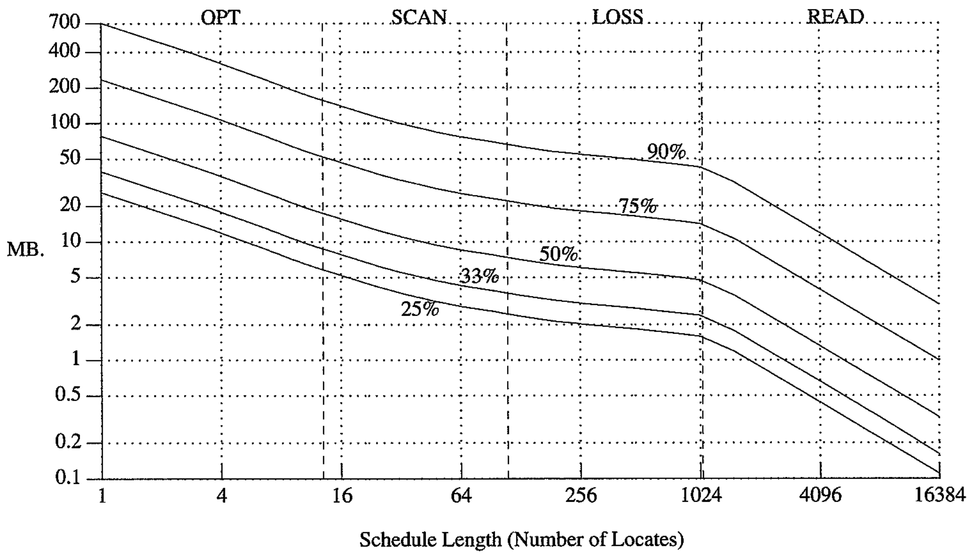


Figure 4: Drive Utilization as a Function of Schedule Length and Transfer Size.

5 Conclusion

For a workload consisting of random retrievals in a robotic tape library, a large component of the service time consists of switching tapes and the fast-forward and rewinding activity needed to access the desired data. For this sort of workload, a traditional figure of merit is the number of retrievals served per hour. We like to think of alternative metrics, namely the drive utilization (`data_transfer_time / (access_time + transfer_time)`), and the effective data rate (`bytes_transferred / (access_time + transfer_time)`). The drive utilization indicates how efficiently the tape library is being used, and the effective data rate indicates whether the library is producing data fast enough to keep the CPU and application busy.

We have studied the performance of the IBM 3570 Magstar MP tape library, which is designed for fast tape switching and positioning times. We have developed a way to predict the positioning time between any two logical blocks on a tape that is filled with fixed-size blocks. Based on this model of locate time, we have studied the performance of several scheduling algorithms for a random retrieval workload on the Magstar MP, and have produced estimates of the effective transfer rate as a function of the two factors: the number of retrievals scheduled from a tape, and the number of bytes transferred by each retrieval. Our estimates are conservative, in that they are based on uniformly-distributed random requests. If the requests for a tape exhibit spatial locality (i.e., if the blocks are somewhat clustered, rather than randomly scattered across the tape), the overhead of tape positioning may decrease.

6 Acknowledgment

The measurements reported in this paper have been conducted on an IBM 3570 Magstar MP tape library loaned to us by IBM. We thank John Gniewek, of IBM Tucson, for arranging this.

References

[1] B. K. Hillyer and A. Silberschatz. On the modeling and performance characteristics of a serpentine tape drive. In *Proceedings of the 1996 ACM Sigmetrics Conference on Measurement and Modeling of Computer Systems*, Philadelphia, PA, May 23–26 1996.

[2] B. K. Hillyer and A. Silberschatz. Random I/O scheduling in online tertiary storage systems. In *Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data*, pages 195–204, Montreal, Canada, June 3–6 1996.

[3] E. L. Lawler, J. K. Lenstra, A. H. G. R. Kan, and D. B. Shmoys. *The Traveling Salesman Problem.* Wiley, Chichester, 1985.

[4] Strategic Research Corporation, 350 So. Hope Ave., Suite A-103, Santa Barbara, CA 93105. *Demystifying tape performance, http://www.sresearch.com/search/105420.htm*, 1996.

Page intentionally left blank

Near-Field Recording
TeraStor's Mass Storage Benchmark Technology for the Next Decade

Gordon Knight, Ph.D.

TeraStor Corporation
3510 North First Street
San Jose, CA 95131-1011
Gknight@terastor.com
+1-408-324-2110

Abstract: TeraStor is developing storage products based on near-field recording that deliver a significant areal density improvement over conventional technologies. The solid immersion lens (SIL) has its origins in microscopy and was refined for use in storage applications at Stanford University. Near-field recording, using the SIL, has a substantial history and the products being developed are the result of many years of research and refinement. The basic properties and advantages of near-field recording are discussed in detail to help the reader understand how density increases are achieved.

Introduction

Near field recording (NFR) is based on several basic principles. NFR uses a combination of first surface recording and a flying optical head to achieve it goals. Digital Equipment Corporation originally developed these concepts in the late 1980s. It is believed that over $60M was invested by Digital in their development. Quantum Corporation acquired the patents on this technology as part of their acquisition of Digital's storage business in 1994. Co-exclusive patent rights have been granted by Quantum to TeraStor for both the flying optical head and first surface recording. The basic technology behind the Solid Immersion Lens (SIL) was developed and patented by Dr. Gordon Kino at Stanford University in 1992. Stanford has granted exclusive patent rights for the SIL technology to TeraStor.

The primary advantage of NFR technology is achieving the highest areal density, or number of magnetically charged bits, that can fit into a given area. Products developed by TeraStor will target markets that value high capacity and random access performance, but have a strong desire to lower their storage cost for a given capacity. These markets include backup, digital asset management, and nearline storage management, among others.

NFR technology is form factor independent and permits both removable and fixed drive applications. TeraStor will commercialize products incorporating NFR in mid-1998. TeraStor has also licensed Quantum to deliver compatible NFR-based products into the market.

The Growing Need for Storage

Data storage needs are rapidly changing, yet the data storage solutions currently being produced with conventional technologies are not growing in capacity as fast as the users' needs are growing. This problem is being created in large part by the rapid convergence of different forms of information or "content" into digital form. This convergence is driven by a desire either on the part of the creators of the data or users of the data to be able to share, transport, replicate and distribute data for mass-market usage, and to ensure the data integrity, longevity, and future restoration capability.

This digital content is becoming vital for applications in areas such as multimedia, graphics design, desktop video, intranets, digital broadcast, and digital photography, in addition to others. This digital content is being driven by a wide range of users including students, business workers, artists, photographers, librarians, journalists, and many others. The convenience of digital data, the ability to easily replicate and protect it all lead to the same conclusions; data is being generated at all levels of the storage market like never before and something extraordinary will need to happen to deal with the growth.

Capacity requirements for mass storage devices range from several megabytes in a handheld computer or digital camera, to gigabytes used in today's personal computer systems, to terabytes and beyond used in massive near-line archival systems or on-line database systems. While all of these storage capacities are achievable today, many different technologies must be used, each having a different set of product, performance and cost issues. These issues make the use of currently available technology in large capacity storage applications either extremely expensive, as in the case of hard disk systems, or uncomfortably slow, as in the case of large off-line and near-line libraries using sequential access media. What is needed is a new mass storage technology providing much higher storage capacities than current technologies, with high-performance random access at a lower cost of storage than other on-line, near-line and offline alternatives. This new storage technology should provide the assurance of an aggressive, continuous path to even greater capacities and performance at lower costs for years to come.

Existing Technology Limitations

Storage technologies being used today present end-users with a set of tradeoffs in determining which product will provide the best solution for their requirements. While one may meet performance requirements, its capacity is wrong for the problem. Another may meet capacity requirements, while not meeting cost requirements. Highlighted below are some of the issues with each of today's available storage product classes.

Hard Disk Drives

Hard disk drives (HDDs) can store from 1 to more than 20 GB. Because of their simplicity of design and relatively high performance, HDDs are ideal for on-line storage of operating systems, frequently used application software and data files. This is because they feature fast random access to data stored anywhere on the disk and allow the data to be retrieved with high transfer rates. In desktop PCs, single HDDs with capacities of 2GB to 4GB are commonly used for this purpose. In larger systems, several HDDs can be used either separately or configured into an array system (RAID) of secure, reliable disks providing tens or even hundreds of gigabytes of storage capacity, but at significant increase in hardware and/or software costs above that of a single drive.

Hard disks usually meet all performance requirements. Where they most often fail to meet requirements is in cost per unit of capacity. This causes end users to architect near-online storage systems, which decrease the cost of storage while increasing access times to some of the data. In environments where data use is infrequent, this can work quite well. In environments where data access is random and constant, near-online storage does not work well. For instance database tables are seldom appropriate for migration to near-line storage. In addition to cost, another attribute missing from hard disks is removability. Today, hard disks are sometimes used in the backup process to hold a mirror image of a production drive, which is then copied to removable media to be stored off site and on low cost media. One thing is for certain; cost makes it impossible to store all of our data in on-line disk drives.

Tape Drives

Tape drives can store from less than 1GB to more than 100GB on removable cartridges or reels of magnetic tape. Their form factors vary widely, as do their performance characteristics. Because of its high storage capacity and low media cost, tape has been the storage medium of choice for backup and archive applications. However, since data must be written and read from the tape sequentially, the amount of time to locate any one piece of data is considerably longer than when using a HDD since the HDD can randomly seek to and access any given data file. With exception at only the very highest end of the product spectrum, data transfer rates are also slower than with HDDs. In addition, when compared to HDDs, tape drives are also perceived as unreliable since tape mechanisms are complex and the tape medium itself is susceptible to a range of environmental threats. For these reasons, tape technology has been limited to data backup and archiving applications (singly or in tape arrays), for temporary file storage and off-line storage in an environmentally controlled vault for disaster recovery purposes.

While its low cost per unit stored is attractive, sequential access tape mechanisms have actually limited the way backup software products can be designed. Backup is looked at as a sequential operation, and being that the backup operation is a streaming function, this is probably OK. Where the sequential access method breaks down is in data restore. Access times to backup copies of data are severely limited by the time needed for a tape drive to locate the required data. In some tape drives this can average several minutes, while the worst case can sometimes be measured in hours. In a better case, it would be possible to handle backups as on-line. But this would require a technology that allows for the low cost of storage, like tape, while presenting the data as on-line like hard disk.

Magneto-Optical Drives

Magneto-optical (MO) drives can store from several hundred MB to several GB of data on a removable disk and come in a variety of form factors. As the need for larger storage capacities on a removable media continues to grow, technologies such as optical storage have emerged to compete with tape and removable HDDs for some applications. All of these optical products employ some sort of laser-based reading and, in the case of writable or rewritable products, laser-based writing technology.

Read-only optical technologies such as CD-ROM and DVD-ROM are very successful as a publishing and distribution medium. Since these products are designed to utilize media that is very low cost but highly reproducible (via a stamping or molding process with a precision master) design tradeoffs are made that are favorable to its use as a tool to access large amounts of published data. These same design tradeoffs, however, become detrimental from a performance standpoint, when these devices are modified to become rewritable mass storage devices and compare unfavorably with higher performance rewritable mass storage products such as HDDs. (see below)

Rewritable optical technologies based on the CD-ROM and DVD-ROM platforms described above, such as CD-R (CD-Recordable), CD-RW (CD-rewritable), DVD-R and DVD-RAM have tried to overcome the performance limitations of a storage technology designed for publishing and distribution applications. These technologies fall short of user needs in the area of performance for use with application software and on-line data files. Thus rewritable products based on CD or DVD technology will continue to fill a market need for short run distribution of high-capacity data files on CD-ROM or DVD-ROM compatible media. The initial appeal of the DVD-RAM disks are simply because they offer a potentially higher capacity random access removable media device than available from any other technology. This appeal is in spite of the fact that performance of these drives is not comparable to hard disk drives. As an example of this, DVD-RAM, when introduced, is expected to offer a storage capacity of just 2.6 GB and transfer rates of just 20 megabits/second (2.5 Mbytes/second) -- much slower than the performance delivered by today's HDDs and unsuitable for serious data storage applications.

Rewritable magneto-optical technology (MO), while overcoming the poor performance of the CD and DVD based platforms, still comes up far short of the performance of HDDs. As an example, the planned extension of MO technology to "ASMO" technology in 1998, while providing capacities as high as 6 GB, will feature access times of 25 to 30 msec and transfer rates of 4 to 40 Mbit/sec, far short of HDDs, yet the product will be relatively expensive when compared with HDDs.

Magneto-optical products continue to face a number of challenges in the market. While they combine two admirable features, random access and removability, the cost of data storage is so high compared to other options that is it usually only used in low capacity applications. Today, the cost of storing data on magneto-optical media is nearly the same cost as keeping the data stored on on-line disks. Furthermore, the use of magneto-optical drives in on-line applications is precluded by their long seek and latency times compared to HDDs.

There are other practical limits to what can be achieved with these conventional mass storage technologies as we go forward in time. In particular, magnetic recording technology used in HDDs faces a potential electromagnetic threshold known as the superparamagnetic limit of the recording material -- the point at which the magnetic domains are so small that their ability to retain a magnetic charge at room temperature is lost. Similarly, the smallest achievable bit cell size in MO technology is limited by the shortest wavelength of light that can be used in lasers, as well as by the highest numerical aperture of a lens that it is practical to achieve in mass volume production. These thresholds present some very real problems that threaten to slow down the rate of increase in achievable areal density of these conventional technologies just at the time when they need to be increasing.

Areal Density - The Key Metric

In the mass storage industry, the metric used most commonly to define achievable storage capacity of any technology is ***areal density.*** The areal density is calculated by multiplying the number of achievable data bits per inch of longitudinal track times the number of data tracks per inch. The resultant areal density is defined in millions or billions of bits per square inch of recording area:

$$\text{Areal Density (bits/in}^2\text{)} = \text{(bits-per-inch) x (tracks-per-inch)}$$

To grow the areal density over time, both the number of bits-per-inch (BPI) and the number of tracks-per-inch (TPI) need to be increased by making the bits smaller and stored closer together. In 1996, the highest achievable average areal density in a shippable product was between 900 and 1,000 Mbits/square inch. In 1997, it will be between 1.3 and 1.5 Gbits/square inch. In 1998, TeraStor's first product is targeted at 15 Gbit/sq in., up to 10 times the areal density of previous technologies.

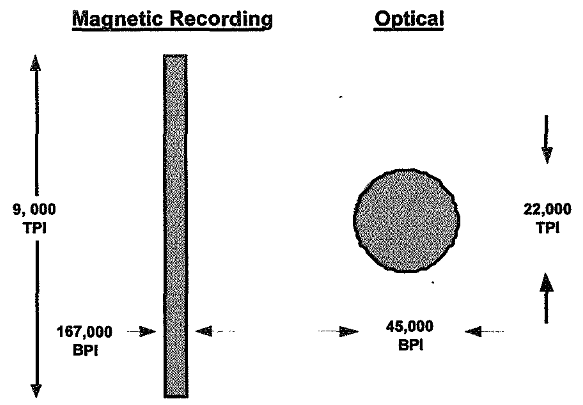


Figure 1. Recorded area comparison between magnetic recording and optical

All storage technologies compete with one another to continually raise the achievable limit of areal density. Each storage technology has particular advantages and disadvantages when it comes to increasing the areal density. In HDDs, BPI is typically much, much higher than TPI because of the properties of the magnetic media and the read/write element of the head, and the tracking methods used. As a result HDDs with 9,000 TPI and 167,000 BPI are expected to be available in 1998. Increases in the TPI are usually more difficult than increases in BPI for magnetic HDDs.

In optical recording, the achievable TPI is usually much higher than magnetic HDDs while the BPI is usually lower. The higher TPI is achievable because high track densities can be stamped or molded with servo tracks in plastic media rather than discrete servo wedges that are recorded by servo-track writers for the HDDs. The laser beam produces a round spot the width of the stamped track. Unlike HDD embedded servo tracks, the stamped tracks allow for continuous track following. The bit density of optical recording is determined and limited by the diameter of the spot size that a laser can be focused to on the surface of the media and is a function of the wavelength of light from the laser. Optical drives with 22,000 TPI and 45,000 BPI are expected to be available in 1998. Figure 1 illustrates the relative recorded area between a magnetic recorded bit and an optical recorded spot.

The Near-Field Solution

TeraStor Corporation has developed a new storage technology that allows an order of magnitude higher areal density when compared to the conventional magnetic and optical technologies that exist today at a significantly lower cost of storage than any current technology. Because of these attributes, users will redefine the boundaries of the storage hierarchy and change user expectations of the capacity, cost and performance features available for any given storage application.

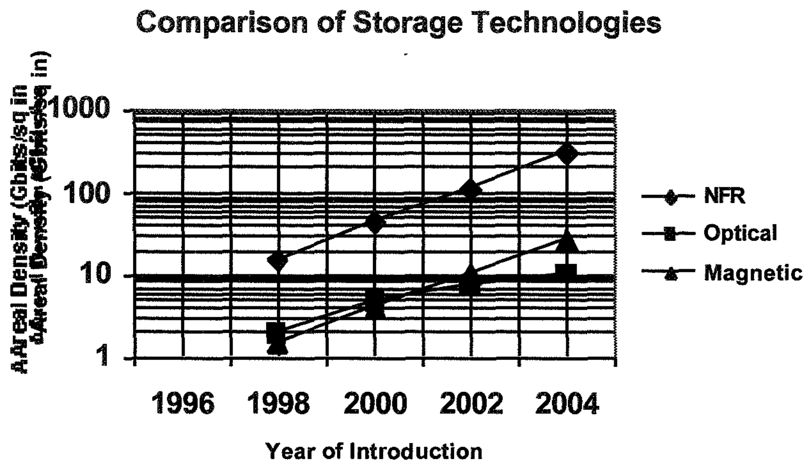


Figure 2. Areal Density Growth Comparison

This new storage technology, called Near Filed Recording (NFR), provides the highest areal density achievable in the storage industry today. The technology provides the capacity and cost of storage comparable to high-end tape drives, but with the performance (seek time and transfer rate) and costs comparable to mid-range HDDs. In addition, TeraStor's well-defined NFR technology roadmap ensures that this areal density advantage will be sustainable for at least the next decade by continuous increases to both the BPI and TPI.

NFR technology is a combination of design elements adapted from several related technology fields, including magnetic recording as in hard disk drives, optical recording, consumer electronics and microscopy. The key elements of TeraStor's *NFR* technology include the flying optical head, the solid immersion lens (SIL), first surface recording, and crescent recording.

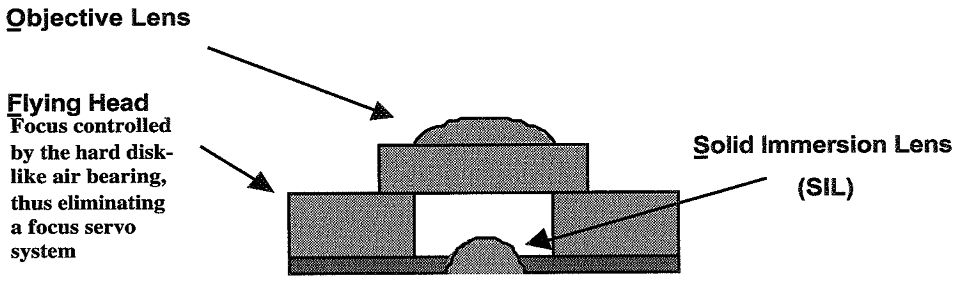
Each of these key elements contributes to achieving the areal density, capacity, performance and cost advantages of *NFR* technology over current technologies. The flying optical head, which is similar to a HDD flying head, provides the simplicity and low cost associated with hard disk drives. This allows the recording element to be placed close enough to the recording media so the distance separating them is less than the wavelength of the laser light, placing it into the near field. One of the key optical components inside the flying head is the SIL, which is used to tightly focus the laser beam to produce an ultra-small spot. The energy from this ultra-small spot is then transferred or coupled onto the first surface (top surface) of the disk in an effect called evanescent coupling. In the near field recording process, a tiny magnetic coil in the flying head writes data to the heated spot. The ultra-small bit domains are written in overlapping sequences, creating a series of crescent-shaped bit domains. This crescent recording effectively doubles the linear bit density, allowing *NFR* to achieve an even higher areal density.

This combination of technology and techniques comprises TeraStor's *NFR* technology, which results in storage densities an order of magnitude higher than current conventional technologies. This areal density advantage is sustainable to capacities of hundreds of gigabytes per disk as the laser wavelength, SIL shape and material, read channels, as well as other elements of the system, improve over time.

TeraStor is the only company that can combine all of these elements into one technology, which leapfrogs the areal density of current mass storage technologies, while maintaining competitive costs and high performance levels. TeraStor products based on NFR will draw heavily on existing MO and HDD materials and processes, allowing low manufacturing costs and rapid production ramps, comparable to those currently seen in high volume HDD products.

Technology Components

The TeraStor flying optical head acts as an aerodynamic, mechanical platform for mounting the objective lens, the SIL lens and the tiny write coil and positions this assembly at a close distance above the surface of the spinning disk. If the distance between the SIL lens and the recording disk is much less than the wavelength of the laser, the resolution of the spot within the SIL is maintained across the air gap through evanescent coupling. Using a red laser as a reference point, the distance between the bottom of the SIL lens and the recording surface would have to be a fraction of the wavelength of a red laser, or less than 0.685 microns. By using a flying head, achieving this close proximity is actually much easier than one might expect. Drawing from the flying heads used in HDD technology, TeraStor is able to produce a laser-based flying head that flies at a distance less than 6 micro-inches or 0.15 microns, well within the 0.685 micron requirement (see Figure 3).


Objective Lens
Flying Head
Focus controlled by the hard disk-like air bearing, thus eliminating a focus servo system
Solid Immersion Lens
(SIL)

Flying Head Cross-section

Figure 3. Flying Head With Objective Lens and SIL

The NFR optical head fly height is less than 6 micro-inches -- significantly higher than heads used in current generation HDDs. In fact, because the air-bearing slider accurately controls the fly height, there is no need for a servo control system to maintain the focus between the lens and the media. The areal bit density resulting from NFR is not directly related to fly height, as it is in HDD systems. In future generations, significant increases in areal density can be achieved without reducing fly height to a distance anywhere close to that of present HDD technologies.

The evanescent coupling or transfer of laser energy heats the spot on the recording surface to the Curie point of about 200 degrees Celsius in roughly one nanosecond. At this temperature, the irradiation heats the molecules in that spot to a finite depth, enabling magnetization when placed within a magnetic field. This magnetic field (positive or negative) is pulsed into the heated spot by a planar coil embedded within the head substrate.

The planar magnetic coil is, as the name suggests, a flat coil that rests on the same plane as the flying head surface. Significantly, this extremely light, small coil resides inside the

flying head assembly, rather than underneath the substrate of the recording media, as is the case with traditional Magneto Optical technology (see Figure 4).
This orientation has two important advantages:
1. Direct overwrite. Through high-speed switching of magnetic pulses, this small head-based coil is able to directly overwrite without going through a complete rotation of the disk. This enables a significant increase in write performance over what is available using MO technology.

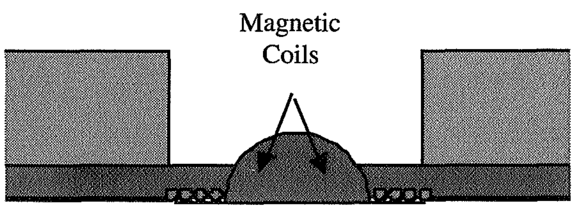


Figure 4. Magnetic Coil in Flying Optical Head

2. Two-sided recording. By embedding the magnetic coil in the head rather than the backside of the disk, TeraStor's solution supports two-sided disks with two heads on line per disk.

In current diffraction-limited optical systems, spot size can be reduced in one of two ways: by decreasing the wavelength of the laser or increasing the numerical aperture (NA) of the objective lens. This approach has clear theoretical limits. For example, in a standard diffraction system, the numerical aperture of a lens in air can never be greater than one.

In the early 1990s, scientists at Stanford University overcame the previous limits on spot size through the use of a new optical system that provides a totally different approach to reducing spot size. This approach increases the effective NA above the theoretical limit of one by adapting an old, but relatively obscure, technique known as liquid immersion microscopy.

In liquid immersion microscopy, the lens and the object to be studied are both placed in a medium such as oil, thereby increasing the magnification of the object beyond what is possible with the microscope alone. The Stanford scientists -- most notably, Dr. Gordon Kino -- simply inverted this effect through the use of a solid lens shaped like a truncated sphere (thus the term "solid immersion lens"). When placed between the standard objective lens and the writing surface, the SIL focuses the incident rays of the laser from the objective lens to a single spot at the base of the partial sphere, as shown in Figure 5. The resulting spot is approximately half the diameter of the spot achieved by conventional means that use only an objective lens.

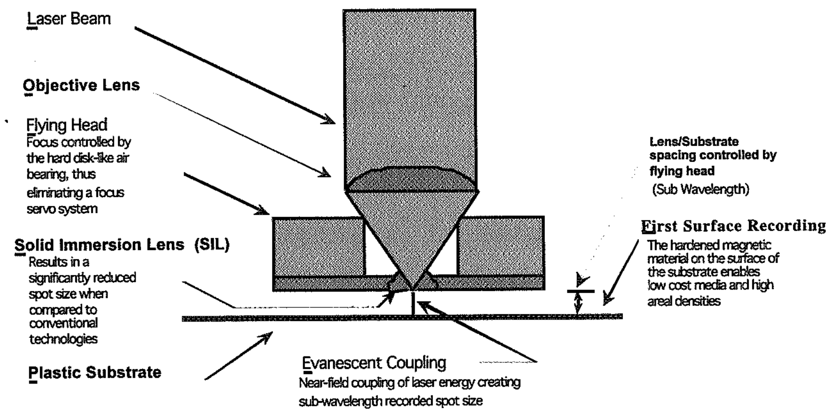

Laser Beam
Objective Lens
Flying Head
Focus controlled by the hard disk-like air bearing, thus eliminating a focus servo system
Solid Immersion Lens (SIL)
Results in a significantly reduced spot size when compared to conventional technologies
Plastic Substrate
Evanescent Coupling
Near-field coupling of laser energy creating sub-wavelength recorded spot size
Lens/Substrate spacing controlled by flying head
(Sub Wavelength)
First Surface Recording
The hardened magnetic material on the surface of the substrate enables low cost media and high areal densities

Figure 5. Architecture of TeraStor NFR technology

In technical terms, the SIL lens slows down the laser light beam to a fraction of its normal speed in air, thus shortening the beam's wavelength and enabling a very fine spot size. Notably, the composition of this SIL lens and its refractive index plays an important role in this process. In particular, materials with a high index of refraction will produce a higher effective NA than those with a lower index of refraction. Air, for example, has an index of refraction of 1.0, while common glass has an index of refraction of 1.5, and diamond has an index of refraction of 2.4. The focused spot size can be calculated using the following equation:

$$\textit{Diameter of Spot} = \frac{\textit{Wavelength of Laser Light}}{\textit{2NA(Objective Lens) x n(Index of Refraction of SIL)}}$$

For example, with a common red laser of 0.685 microns wavelength, and a numerical aperture of 0.65, the focused spot diameter would equal 0.53 microns. If this same focused beam is put into a SIL with an index of refraction of 2.0, the spot diameter is reduced by a factor of two to 0.26 microns.

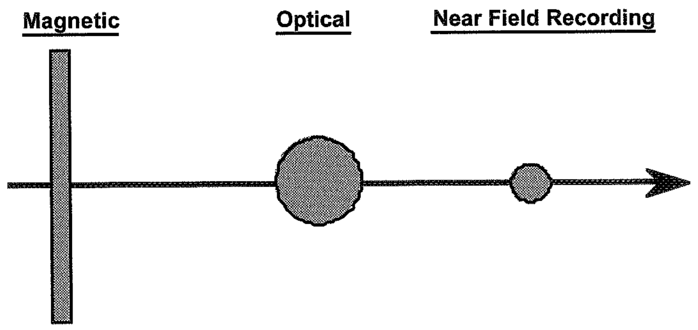


Figure 6. Recorded Area Comparison

The objective lens focuses the laser light in a manner that results in a spot that exists only within the SIL lens. Thus, if the lens were placed near a recording surface, only a small fraction of the light from the laser would appear on that surface (the on access rays). To make the small spot at the bottom of the SIL lens appear on the surface, the Stanford researchers took advantage of another technique known as evanescent coupling.

The concept of evanescent coupling comes from quantum physics and is best introduced by analogy. Let's say, for example, that a doctor puts on a stethoscope and slowly brings the amplifier near a patient's chest. The doctor would begin to hear a heartbeat just before actually coming in contact with the skin. At this point, the stethoscope is within the near field of the heart's sound waves, causing the waves to couple with the stethoscope and produce a startlingly high-quality sound in the instrument even though no physical contact has been made. A similar effect occurs when an energized microwave waveguide (a tube-like device) is placed closely parallel to a second, non-energized waveguide: a portion of the microwave energy begins traveling down the second waveguide. The microwave energy in the first tube couples with the second tube and produces a corresponding energy field in the second tube.

Evanescent coupling was once thought to be impossible, since a sudden discontinuity, such as a wave jumping across barriers, was believed to actually violate the laws of physics. But equations developed in the last century by Scottish physicist James Maxwell indicate that any form of electromagnetic radiation enclosed within a boundary will produce waves outside that boundary that decay at an exponential rate. If an object is close enough to that boundary (within a fraction of a wavelength), it will receive some of the transmitted radiation. In some cases, evanescent coupling can produce a very high-quality effect, inducing more than 50 percent of the energy in the source radiation to couple with the material in the near field. This is precisely what happens in NFR.

In this case, a recording surface is placed within the near field of a SIL lens and a laser spot is focused on the bottom of the lens. The close proximity evokes an evanescent coupling of laser energy, resulting in a "transfer" of the spot from the inside of the SIL to the surface of the disk, creating a small but well-defined spot on the recording disk.

Flying close to the disk surface would mean little if the recording material or storage layer was buried under layers of protective plastic substrate. Traditional MO media uses multiple layers of sputtered material with an intervening substrate layer that places the recording material too far away from the flying head/SIL lens to be practical for NFR. As shown in Figure 7, TeraStor solves this problem by sputtering the magnetic storage layer onto the surface of the disk (and not putting layers of plastic on top). This hardened magnetic material on the top surface of the disk supports NFR and enables high-volume, low-cost production by manufacturers familiar with present MO recording media. The disk itself can be a low-cost plastic injection molded substrate, providing significant cost-savings relative to metal or glass substrate media.

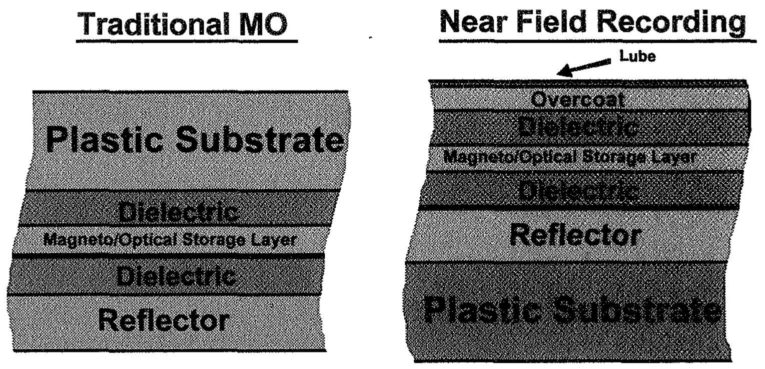


Figure 7. Cutaway view of Magneto/Optical and NFR recording media

Up to this point, the TeraStor solution described has combined a SIL lens, evanescent coupling, a flying head and a planar magnetic coil enabling the production, on first surface recording media, of a bit cell that is smaller in diameter than is attainable using any other recording technology. As the final step in the process of NFR, TeraStor has implemented a crescent recording technique where domains partially overwrite each other, producing distinctive crescent-shaped domains increasing the areal density of the recording surface.

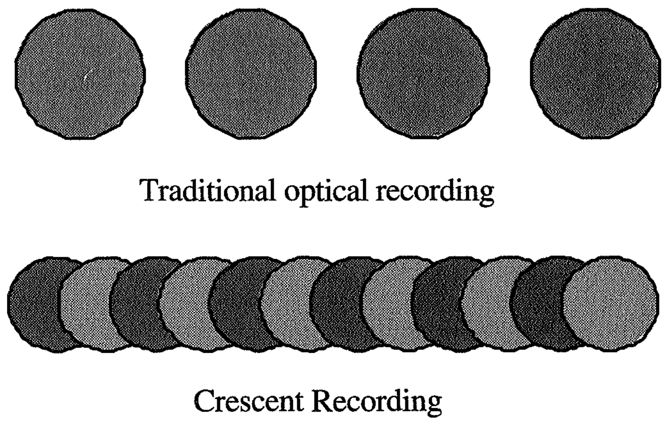


Figure 8. Crescent recording compared to traditional optical recording

The areal density is achieved by the combination of the small spot size produced by the SIL and evanescent coupling along with the overlapping pattern of crescent recording. This results in nearly equivalent linear bit density or BPI as compared to traditional HDDs, but with significantly higher TPI, as depicted in Figure 9.

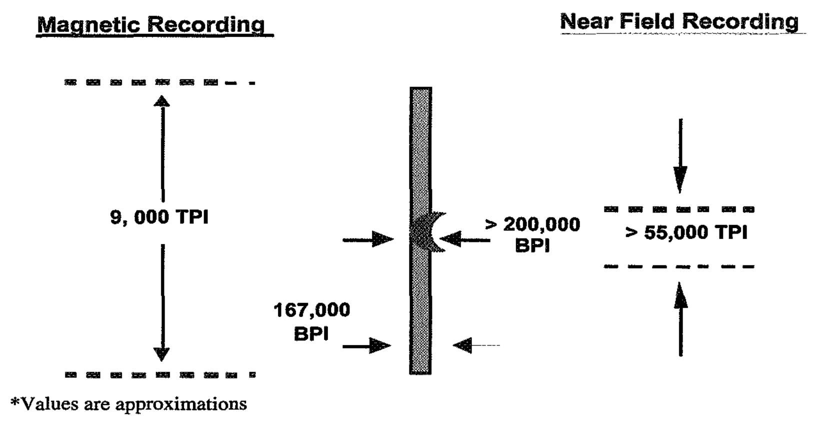


Figure 9. Areal Density Comparison

Conclusion

NFR represents an entirely new category of mass storage technology with distinct advantages over all previous technologies. This breakthrough technology delivers more storage capacity per square inch at a lower cost per gigabyte than any of today's other mass storage technologies, including HDD, tape and MO systems. At the same time, the technology is also easily implemented in both removable media and fixed media devices.

NFR enables direct overwrite and very small bit cells. Further, the technology provides direct access capabilities. All of these features enable the high capacity and performance levels required to serve the needs of high-end data center and workstation environments, as well as the cost savings necessary to support desktop systems. The random access performance of NFR makes it equally suitable for storing system files, applications and actively used data, including video, 3-D images, Internet downloads, CAD/CAM files and more.

With its easy support for standard interfaces such as SCSI, FC-AL, IEEE 1394 and IDE, NFR technology is bound to begin displacing tape, MO and removable HDD technologies in backup and archive applications as soon as products become available.

As the following chart indicates, TeraStor's NFR technology provides a variety of technical benefits compared to HDDs, magneto-optical, and tape drives. Notably, NFR provides a less expensive substrate, a fast direct overwrite capability, and greater bit density per square inch than competing technologies. NFR also allows HDD data transfer rates and does not require an expensive, complex focus servo control to enable accurate recording. In addition, because of its vertical bit cell writing technique, NFR has no known areal density limits, such as the superparamagnetic limit identified for HDD technology.

Compared to tape technology, TeraStor's NFR technology provides equal or greater storage capacities, higher transfer rates, random access, and low-cost, long-life recording media. This high-performance, low-cost solution provides reliable backup and archiving capabilities with a media life of at least 30 years -- well in excess of anything tape drives can offer. NFR provides a clear capacity advantage over MO technologies. In addition, NFR allows much faster transfer rates and access times and much lower drive costs. NFR enables a much higher capacity or lower cost/GB than is possible with the removable HDDs currently available. DVD-RAM, like its older counterpart, the compact disk, is mainly a device for publishing and distributing mass-produced software content, thus performance has never been near that required for a system data storage device. Eventually, NFR technology will be used as the primary, nonvolatile data storage devices designed into systems. Since NFR provides up to a 10x areal density improvement over current HDD technology, it is expected to be an attractive choice for users as an add-on storage technology for desktop systems and workstations.

Table 1 TeraStor NFR vs. Alternative Technologies

	HDD	Optical	Tape	NFR
Random access	x	x		x
Sector addressing	x	x		x
High performance	x			x
Removability	x	x	x	x
Low Cost/GB			x	x

We can only imagine how this technology will change the world of storage management. The value of small tape libraries will diminish with the first product introductions. The changes allowed to backup strategies and architectures are hard to predict, but one can easily imagine making backup, and restore for that matter, an on-line phenomenon. Finally, what contributes to the choice of tape for near-line storage systems will become less clear as high capacity removable and fixed disk products based on near field recording are introduced in the future.

Page intentionally left blank

Charged Particle Technology for Ultra High Density Data Storage

Andrew H. Bartlett, Homi Fatemi, Marc H. Eberle
Norsam Technologies, Inc.
61 Rover Blvd.
Los Alamos, NM 87544
information@norsam.com
www.norsam.com
Tel: +1-505-672-0920
Fax: +1-505-672-0922

Abstract: Long term archival storage of valuable documents is rapidly approaching a critical juncture. The introduction of reliable and acceptable microfilm techniques over sixty years ago allowed preservationists to address the 'slow fires' of deteriorating, acid-based paper documents in a cost-effective manner. Since that time, microfilm standards and techniques have evolved to allow for a relatively secure manner in which to store a variety of document types, ranging from black and white texts to color photographs. However, the advent of digital information creation and retrieval has created numerous new problems that conventional archival storage techniques are hard-pressed to address. Most notably, the capability to create huge amounts of data, often stored only in electronic format, has led to a crisis not only in the need for preservation standards for digital documents, but as well has resulted in impossible demands upon the physical storage capabilities of microfilm storage vaults. With the limited lifespan of organic-based microfilms, particularly older films, the need to replace microfilm with a more durable and dense storage medium is acute.

Norsam Technologies is developing an ultra-long term storage solution in which analog, i.e. eye-readable, images are placed at unprecedented densities into nickel substrates, allowing for near geologic storage lives of data that, as eye-readable images, are not subject to the vagaries of changing formats and media to which digitally stored data is so susceptible. Specifically, using focused beams of Ga^{+1} ions, data can be written at pixel sizes as small as 25 nanometers (nm), allowing, for example, the storage of one million pages of data in less than a cubic inch of material. Readback of this data is direct, requiring no intermediate bitstream interpreter. This paper will outline the process by which the data is written and stored using focused ion beam (FIB) technology. In addition, Norsam is developing a digital product written using focused charged particle technology that will allow for the introduction of a digital technology capable of storing 165 GigaBytes (GB) of data on a CD-sized disc.

Technology

Ion irradiation is commonly used for surface modification of materials.[1] Of particular interest to data storage is ion implantation.[1] Because of fast write rates, implantation and subsequent etch-stop techniques have developed as the method by which data can be efficiently and rapidly written.

Ions are well suited to write data by implantation and etching techniques. Norsam's technology currently uses Ga^{+1} ions implanted at 30 keV into (100) silicon to write the data. By adjusting the ion beam current and the dwell time per spot, the size of the data spot and the dose of implanted ions can be carefully controlled. It has been determined that implantation doses of approximately 10^{13} - 10^{14} ions/cm^2 give a good combination of economical write times and sufficient implantation statistics to reliably and reproducibly write bits of data.[2] Under these conditions, gallium serves as an etch-enhancement dopant, with etchant exposure times as short as 1 minute. Figure 1 is a schematic of the ion implantation process. In this case, the spatial resolution is governed by the beam current, which is a function of the selected aperture. Typical spot sizes, which determine the pixel size of the written data, are about 50 nm. Depth of implantation is a function of incident ion energy, which for 30 keV is about 50 nanometers.

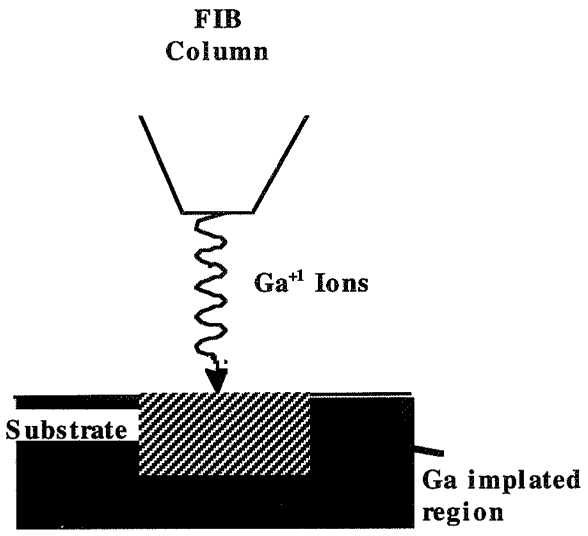


Figure 1

Following implantation, the wafers are transferred to an etch bath and processed for approximately one minute. The regions in which Ga^{+1} has been implanted etch more rapidly than the unimplanted regions, thus the data, in the case of text, is developed and is revealed as topologically distinct regions, essentially engraved into the substrate. Data writing is affected by additional process details. Anisotropic vs. isotropic etching and Si orientation dependency can be exploited to ensure a consistent pit wall profile. Implantation depth is a function of incident Ga^{+1} energy as well as substrate properties; this, combined with etch rate dependency on dose, allows for the writing of variable depth data pits. Different substrate materials can allow for even lower doses, though statistical variations in the number of particles will provide a lower limit to the dose.

The amount of data that can be written into a substrate is a function of spot size. Feature size, density and comparisons with other analog archival storage methods are given in Table I. The time to write each page of data is a function of the number of pixels to be written. For a typical 300 dpi document at 2550 x 3300 pixels, with 10% opacity, actual implantation times are around one second. Software overhead and stage motion increases the total implantation time per page to around two seconds per page. At this rate, nearly fifteen million pages of data per year can be written with each FIB machine. With further improvements in substrate design and software, write rates are expected to decrease to 0.2 seconds per page.

Table I
Comparative weight and volume of paper, fiche and archival substrate for 1M images, assuming Ni archival substrate @ 0.25 mm thickness, and 60 images per 4" x 6" fiche.

Spot size	Ni weight	Ni volume	Fiche weight	Fiche volume	Paper weight	Paper volume
1.0 micron	64 lb.	0.12 ft^3	[a] 93 lb.	[a] 1 ft^3	[a] 10,200 lb.	[a] 206 ft^3
0.2 micron	3.1 lb.	.005 ft^3	Invariant	Invariant	Invariant	Invariant
0.15 micron	1.8 lb.	.003 ft^3	"	"	"	"
0.1 micron	.79 lb.	.001 ft^3	"	"	"	"
50 nm	3.2 oz.	0.6 in^3	"	"	"	"
25 nm	0.8 oz.	0.2 in^3	"	"	"	"

Data writing by FIB implantation is essentially a mastering process. Thus the information on the Si master can be transferred to an archival substrate, such as nickel, by electroforming methods. Figure 2 is a schematic of the electroforming process, showing how the conformal capabilities of electroforming allow for the production of a freestanding substrate containing the data.

Some of the requirements of the archival substrate include: sufficient thickness for durability and strength when handled, mirror flatness, and reproduction fidelity. Electroforming has already been shown to be capable of reproducing features as small as 80 nm, and even better resolution is expected, while the excellent mechanical properties of nickel will allow for durability. Protection against oxidation, corrosion, and fire will require prudent encapsulation. Figure 3 shows a prototype design for a storage box, which, when purged with an inert atmosphere and constructed with sufficiently thick refractories, would allow complete protection against almost any disaster. Bearing in mind that this encapsulator design would replace entire buildings currently dedicated to microfilm preservation, the cost effectiveness of ultra-high density storage becomes apparent.

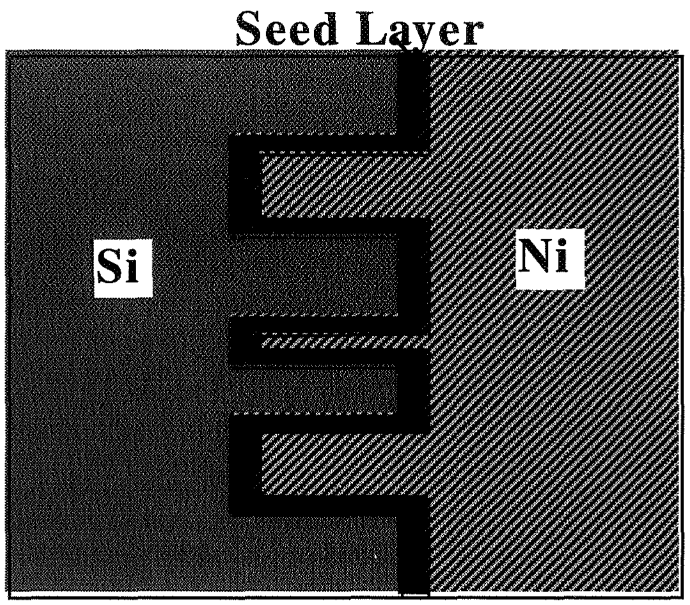


Figure 2

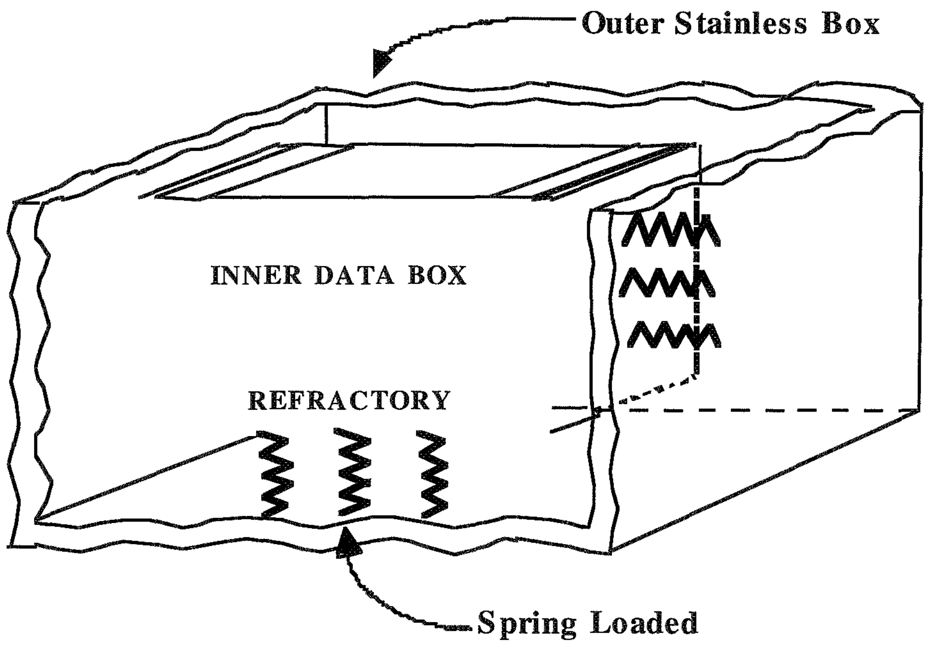
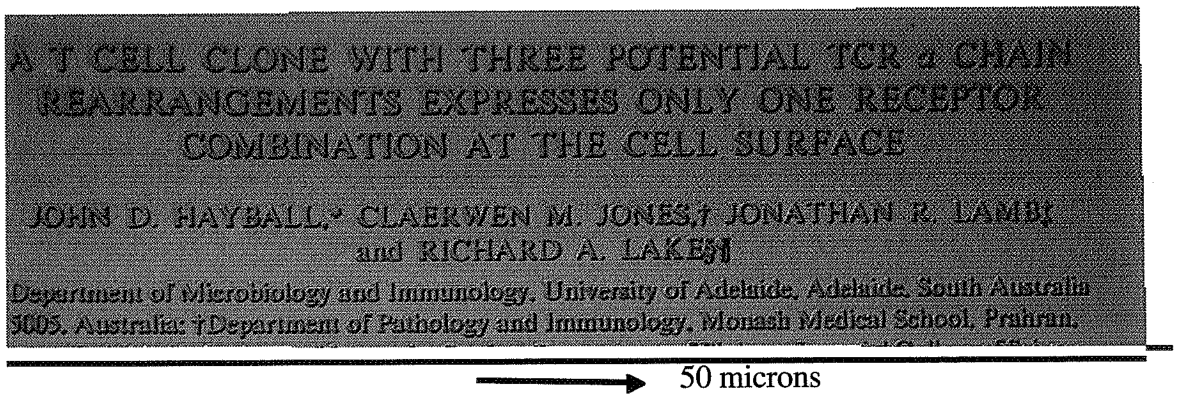


Figure 3

Readback of the data is accomplished by optical microscopy. Figure 4 is an illustration of data supplied by the National Library of Medicine, in this case a portion of a page of text from an electronic journal.[3] For this example, the resolution of the microscope is approximately 0.2 micron, which, in the case of data written at 500x, corresponds to nearly exact pixel by pixel recovery of data. Data written at finer pixel sizes can easily be read by optical techniques, since many hundreds of pixels make up each letter of text, allowing for excellent readability even when written at pixel sizes below the resolution limit of the microscope. However, if exact retrieval is required, scanned probe techniques, such as scanning electron microscopy, have been demonstrated.


A T CELL CLONE WITH THREE POTENTIAL TCR α CHAIN REARRANGEMENTS EXPRESSES ONLY ONE RECEPTOR COMBINATION AT THE CELL SURFACE

JOHN D. HAYBALL,* CLAERWEN M. JONES,† JONATHAN R. LAMB‡ and RICHARD A. LAKE§¶

Department of Microbiology and Immunology, University of Adelaide, Adelaide, South Australia 5005, Australia; †Department of Pathology and Immunology, Monash Medical School, Prahran,

Figure 4

CONCLUSION

Norsam Technologies has successfully demonstrated all aspects of the production technology required to produce its initial analog product, and is planning to begin analog product manufacture by the end of Q2 1998. It is developing beta-site customers, providing initial data writing demonstrations, and proposing data management solutions to several very large clients. Norsam currently owns four FIB milling machines, and is in the process of setting up its first production facilities.

In addition, Norsam Technologies is developing an ultra-high density digital product, HD-ROM, which will use charged particle technology to write at pit sizes at 50 nm and below. Beam blanking technology will allow for write rates starting at 200 Mbps, while near field technology will permit read-back rates at 60 Mbps, at resolutions less than 5 nm.

References

1. Michael Nastasi, James W. Mayer and James K. Hirvonen, Ion-Solid Interactions : Fundamentals and Applications, Cambridge University Press, New York , 1996.

Footnotes

[1] Implantation means that the ions are physically placed into the substrate material under high accelerating voltage. The implantation depth depends upon the voltage.
[2] Writing rates are currently around 3 Mbps, and are expected to increase to about 20 Mbps.
[3] Note that this picture was captured using a 1k x 1k CCD camera. The data was written at 2550 x 3300 resolution, such that nearly 80% of the data was lost in this optical image capture. Regardless, excellent fidelity is still retained.

Page intentionally left blank

Analytic Performance Models of Robotic Storage Libraries: Status Update

Theodore Johnson
johnsont@research.att.com
AT&T Laboratories – Research
Florham Park, NJ 07932
tel +1-973-360-8779
fax +1-973-360-8050

Abstract

Large tertiary storage systems tend to be expensive, so performance models are needed for sizing and for performance optimization. Several researchers (including ourselves) have attempted to supply these models in recent years. In this paper we report on our progress in the development of a previously published model. The improvements we have made include a more accurate model of the robot arm, experimental validation of the model, and the integration of the model into PC analysis tools through a Perl interface.

1 Introduction

In spite of rapidly declining on-line storage prices, many applications, such as scientific archives, scientific databases, data warehouses, digital libraries, and multimedia servers create and serve such large volumes of data that the use of tertiary storage is required. Large tertiary storage systems tend to be expensive, and tend to have unusual performance characteristics. Performance models are needed for sizing and performance optimization studies.

The need for performance models has motivated considerable recent research. Johnson [3, 4], and Menasce, Pentakalos, Yesha, and Halem [7, 8] have developed analytical performance models of tertiary storage systems. Gibson and Miller [1] are developing a configurable mass storage simulator. The NASA Goddard Mass Storage Testing Laboratory [9] is developing a benchmark suite to test and rate HSM solutions.

In [3, 4], we presented a detailed queuing model of a robotic storage library, incorporating the interaction of robot arm, batch arrivals, and multiple file loads from a tape. Since then, we have continued to work on improving the accuracy and usefulness of the model. In particular,

- A weakness of previous tertiary storage models was the lack of detailed information about the performance of model components. We have performed an extensive performance characterization study of common tertiary storage components [6]. We have used this information to make our analytical model more realistic.
- We have refined our model of the robot arm, increasing its range of accuracy.
- We have validated our model using experimental measurements of a Storagetek 9710 with four DLT 4000 drives.
- We have developed an interface between our model and common PC data analysis tools, using widely available public domain software such as Perl.

In this paper, we report on the last three refinements (please refer to our paper [6] in these proceedings for information regarding tape drive performance characterizations).

2 Robot Arm Modeling

In the model reported in [3, 4], we describe how the interaction between queuing for service from the robot arm of robotic storage library and queueing for service from the tape drives can be modeled with an iterative computation.

A *job* in our model represents a user's request for data. Our analysis of usage patterns [2, 5] indicated that a typical request is for a batch of files, possibly distributed over multiple media. In our model, a job splits into a batch of requests, where each request represents the loading of data from a media. The time to serve a request is the sum of the media fetch time, mount time, seek times, and file transfer times. Each request queues for service from one of the media drives, and the job is finished when all of the requests in the batch are finished. The media fetch time depends on the drive utilization, and vice versa, leading to the iterative solution. The robot arm is modeled as a $M/G/1$ queue with an infinite customer population and batch arrivals. While the actual customer population is finite, for light loads the approximation is reasonably accurate.

In the process of performing an experimental validation, we found that the robot arm could easily be made to have a high utilization. The infinite customer model overestimated queuing delays at the robot arm, decreasing the accuracy of the model. To fix the problem, we developed a finite customer population model. In this section, we briefly describe the approach.

The software we use to manage the robotic storage library has a volume manager that does not rewind tapes after use, nor does it return the tapes to the shelf. In addition, the volume manager is single threaded, so only one tape can be changed at a time[1]. As a result, the

[1]If we disable the "fast load" setting on the Storagetek 9710, the robot does not return a ready status until the drives are ready (in case the tape handling software cannot detect when the drive is ready). This setting also results in very long robot service times.

time spent by a request in service by the volume manager can be represented by the robot arm server. The service time of the volume manager includes rewinding the tape in the selected drive, unmounting it, returning the old tape to the shelf, fetching the new tape, and mounting it.

Let c be the number of tape drives in the robotic storage library. If b of the c drives are busy, then the the maximum number of tape load requests pending service by the volume manager is $c - b$. If there are $q < b - c$ load requests pending service by the volume manager, then a new load request must wait for the volume manager to finish the existing q requests.

The model described in [3, 4] assumes that a user fetch request translates into multiple media fetch requests As a result, media load requests also come in batches, but the batch size cannot be larger than the number of idle drives, $i = c - b - q$. Let B be the random variable representing the number of media loads generated by a user fetch request. Let $p_j = \Pr[B = j]$ be the distribution of B. Then if i drives are idle, the size of the batch arrival at the volume manager queue is the random variable B_i with the distribution $p_{i,j} = \Pr[B_i = j]$ defined by

$$p_{i,j} = \begin{array}{cc} p_j & j < i \\ \sum_{k=i}^{\infty} p_k & j = i \\ 0 & j > i \end{array}$$

We can describe the state of the volume manager by specifying the number of busy, queued, and idle drives. Since $b + i + q = c$ and c is constant, we only need to specify two of the three parameters to describe the state. It turns out to be easiest to specify the state of the volume manager with (b, i). The volume manager can be in any state (b, i) such that $b \geq 0$, $i \geq 0$, and $i + b \leq c$.

Next, we specify how the volume manager can change states. The transitions are given in Table 1. In this table, E_{tr} is the expected service time of the volume manager, E_{dr} is the expected service time of drive to fetch the requested data, and p_{qd} is the probability that if all drives are busy and a drive finishes service, there is pending fetch request for the drive to service.

The states and the transition function of the volume manager model define a finite Markov chain model, which we can solve for the steady state behavior. The model has $O(c^2)$ states, and c is small. The robot arm model can be extended to handle deterministic robot arm service times [6] (by using an embedded Markov chain model) and separate media fetch and return requests. We will detail these models in a full paper.

cause	from	to	rate	condition
robot completes service	(b, i)	$(b + 1, i)$	$1/E_{tr}$	$b < c - i$
drive completes service no queued requests	(b, i)	$(b - 1, i + 1)$	b/E_{dr}	$i > 0, b > 0$
drive completes service no queued requests	(b, i)	$(b - 1, i + 1)$	$(1 - p_{qd})b/E_{dr}$	$i = 0, b > 0$
drive completes service queued requests	(b, i)	$(b - 1, i)$	$p_{qd}b/E_{dr}$	$i = 0, b > 0$
job arrival	(b, i)	$(b, i - j)$	$p_{i,j}\lambda$	$j \leq i$

Table 1: State transitions for the robot arm (volume manager).

3 Experimental validation study

We had access to a Storagetek 9710 with four DLT 4000 tape drives. As discussed in the previous section, the volume manager of our tape management software is single threaded, requiring a modification in the definition of the robot service time and the drive service time. We measured the Storagetek 9710 and the DLT 4000 to obtain service time parameters (please see [6] for details on the measurements).

We submitted synthetic jobs to the robotic storage library. A controller script generated requests by waiting for an exponentially distributed length of time and then creating a batch request. A batch request was generated by forking off k processes each of which requested the load of a media, and the fetch of files from the media. The batch size is chosen by sampling an exponentially distributed random variable, while the number of files fetched from a tape has a uniform distribution. Because of the limited number of test tapes (40), we limited the batch size to 15 media. The size of a file fetched from tape can be 1 Mbyte, 10 Mbytes, 50 Mbytes, 200 Mbytes, or 1 Gbyte with a 30%, 30%, 20%, 15%, and 5% chance, respectively, representing a wide range of file sizes. We note that the wide range of file sizes and the emphasis on small files makes obtaining an accurate model more of a challenge.

We varied the arrival rate, the average number of media fetched, and the average number of files fetched per media, and measured the batch waiting time and the batch service time (i.e., by using the waitpid system call). Each experiment was terminated after 50 jobs completed. We assumed that seeks required an average of 75 seconds, the robotic service time is 9 seconds, the mount time is 40 seconds, the rewind and unmount time is 100 seconds, and the transfer rate is 1.5 Mbytes/sec, based on our benchmark measurements [6]. Table 2 lists the results.

The response time predictions are usually within 20% of the observed response time, and in all cases is within 30%. In some cases the predicted response time is quite close to the observed. This good agreement (and also the cases of poor prediction) is due to random chance, as only one sample path was observed. However the general trend is that the model

avg. time between arrivals	avg. media per job	avg. files per media	number of drives	experimental response time	analytical response time	percent difference
1000 sec.	2	5	3	3438 sec.	3571 sec.	3.8%
750	2	3	4	1473	1695	15
1000	2	3	4	1283	1297	1.1
1000	4	3	4	2813	3273	16
1400	3	3	3	2755	2247	18
1000	2	3	3	1293	1340	3.6
1200	4	3	4	3432	2473	28

Table 2: Experimental and analytical predictions of job service times.

gives reasonably accurate predictions. We note that the device utilizations were high, with a drive utilization ranging from 49% to 70%, and a robot arm (volume manager) utilization ranging from 24% to 38%. Accurate response time predictions are difficult when device utilizations are high.

Although we generated a synthetic workload that matches the modeled workload, the processing of the workload was performed by the robotic storage library and its driving software. Therefore we conclude that our model of the mechanics of the robotic storage library are accurate.

4 Interfaces

We have ported the model solver to the Win 95 platform. The model solver is written in ANSI C, so POSIX compliant C development environments (for example, DJGPP, which is available for free) can compile the model and generate executable code.

The Perl scripting language is an excellent tool for driving an external program, parsing the results, and generating reports. Perl is also available for the Win 95 platform for free. Recent versions of Perl (e.g., Perl 5) support OLE calls. As a result it is easy to write a Perl program that makes multiple calls to the model solver, collates the results, passes a table to a spreadsheet package, and causes the spreadsheet to plot the results. We have developed some sample scripts.

5 Conclusions

Understanding the performance of tertiary storage devices is a difficult task, but is necessary for the planning and optimization of large data systems. In this paper, we present our progress in developing an analytical model of a robotic storage library. By developing a more accurate model of the robot arm and by taking measurements of a Storagetek 9710 with four DLT 4000 tape drives, we were able to accurately model the system. To simplify the use of the model, we have developed interfaces to tools available on a PC platform.

Acknowledgements

We'd like to thank the Session Chair, Jean-Jacques Bedet, for his comments on a draft of this paper.

References

[1] T. Gibson and E. Miller. Long-term file activity patterns in a unix workstation environment. In *Proc. NASA Godddard Conf. on Mass Storage Systems and Technologies*, 1998.

[2] T. Johnson. Analysis of the request patterns to the nssdc on-line archive. In *NASA Goddard Conf. on Mass Storage Systems and Technologies*, pages 367–382, 1995.

[3] T. Johnson. An analytical performance model of robotic storage libraries. In *Performance '96*, pages 231–252, 1996.

[4] T. Johnson. Queuing models of tertiary storage. In *NASA Goddard Conf. on Mass Storage Systems and Technologies*, pages 529–552, 1996.

[5] T. Johnson and J. Bedet. Analysis of the access patterns at the GSFC Distributed Active Archive Center. In *NASA Goddard Conf. on Mass Storage Systems and Technologies*, pages 153–178, 1996.

[6] T. Johnson and E. Miller. Performance measurements of robotic storage libraries. In *Proc. IEEE Conf. on Mass Storage Systems / NASA Goddard Conf. on Mass Storage Systems and Technologies*, 1998.

[7] D. Menasce, O. Pentakalos, and Y. Yesha. An analytical model of hierarchical mass storage systems with network attached storage devices. In *SIGMETRICS 96*, pages 180–189, 1996.

[8] O. Pentakalos, D. Menasce, M. Halem, and Y. Yesha. Analytical performance modeling of mass storage systems. In *Proc. 14th IEEE Mass Storage Systems Symp.*, 1995.

[9] R. Venkataraman, J. Williams, D. Michaud, P. Hariharan, B. Kobler, J. Behnke, and B. Peavey. The mass storage testing laboratory at GSFC. In *Proc. NASA Godddard Conf. on Mass Storage Systems and Technologies*, 1998.

Monitoring the Life Expectancy of Compact Discs

Basil Manns
Preservation Directorate
Library of Congress
Washington, DC 20540-4560
bman@loc.gov
+1-202-707-8345
fax: +1-202-707-6449

Abstract: This report measures the condition of 125 CDs that are statistically selected from a collection of over 60,000 discs. This small sample was selected to demonstrate a testing program and to get some feeling as to the general condition of the discs in the collection. Using this small sample, a complete set of ISO measurements were performed. A CD-CATS system was used to perform such measurements. Based upon the findings of these initial tests of 125 CDs, a more detailed test of the complete collection may be recommended. In any case, it is the intent that, at a minimum, these tests be repeated using these same discs in a few years to determine if any and what kind of changes may occur. With these data, information regarding the life expectancy of the collection may be determined.

1. The collection

The Library of Congress (LC) has a large collection of audio CDs and the condition of these discs were never monitored. This small sampling test is to get some data as to the condition of the existing collection, and perhaps develop a baseline for future measurements.

The tester used by the Library of Congress was a CD-CATS system, configured with a 486 PC, printer, and software to output various reports. These reports consist of summary of each parameter measured, a list of each parameter out of specification, and tools to show surface presentations of certain parameter variations. It measures all the ISO parameters, in addition to the jitter and length deviation, which is becoming a significant indicator of errors.

The testing and monitoring program statistically selected a sample of 125 audio compact discs from the LC collection of over 60,000. This 60,000 number is only an estimate of the number in the collection, but a number that most could agree upon. Statistically speaking, a sample size of over 1000 disks would be needed in order to have a confidence level of over 95 %, if discs are normally distributed in defects. However, since we know that the disks are not purely distributive in defects, a successive estimate approach was used, projecting that the true error rate may be less than 5%, the sample size was reduced to 125 disks. This was felt to be a reasonable number of discs to test as a demonstrative testing and monitoring program. The next phase may increase this number to 1000.

2. The test parameters

The test results are grouped, by the CD CATS system, into three categories: static data, dynamic data, and the recommended length deviation and jitter data. Obviously, some of these parameters are of more interest than others in this testing and monitoring program. The parameters are listed below and their definitions and use of the terms are discussed in ISO/IEC 10149, *Data interchange on read-only 120 mm optical disks (CD-ROM).*

The static data includes: SLD (Start Lead In is the optical start of the tracks), SPD (Start of Program Diameter), MID (Maximum Information Diameter), SVY (Scanning Velocity), TRP

(Track Pitch), ECC (Eccentricity), DEV (Deviation), DEFL (Deflection), PP (Push Pull), PPC (calculated Push Pull), and XT (Cross Talk).

The dynamic test data consist of the following: BLER (Block Error Rate), E11, E21, E31, E12, E22, and E32 (error counts), I3, I11, I3R and I11R (reflective indicators), REF (Reflectivity), SYM (Symmetry), BERL (Burst Error Length), CRC (Cyclic Redundancy Check), RN (Radial Noise), BEGL (Burst Errors Greater than Limits), and E32TOT (total number of uncorrectable blocks present within the testing time).

The time interval data, which are not ISO specification, but important to monitor, are the length deviation and jitter. The length deviation is the length of each pit and land area as measured, and compared to the ideal pit and land lengths. The results are compared for each run length, 3T to 11T, (9 values) with variations preset for each run length. Jitter is measured individually for pit and land (3T ...11T), 9 times 2, or 18 measurements. Each pit and land is measured and then jitter is calculated statistically as a standard deviation, (18 more measurements) according to Philips specifications. This is done for the entire track of the disc. As can be imagined, a lot of data are generated for each disc.

The tester also generates an "OUT OF SPEC. LIST", based on the maximum value specified by the user of the CD CATS system. This is a useful reporting mechanism for flagging errors. Based on where the "maximum" value is set the list of errors may not really be errors, but levels outside the reporting boundary. Also, some discs generate numerous "errors" that are not real errors, but only transitions between recording sessions. This depends a lot on how the discs were initially formatted, recorded, and manufactured. Since these are not really errors, it is important, when analyzing the data to determine where the errors occurred. They may have nothing to do with the condition of the disc or the data.

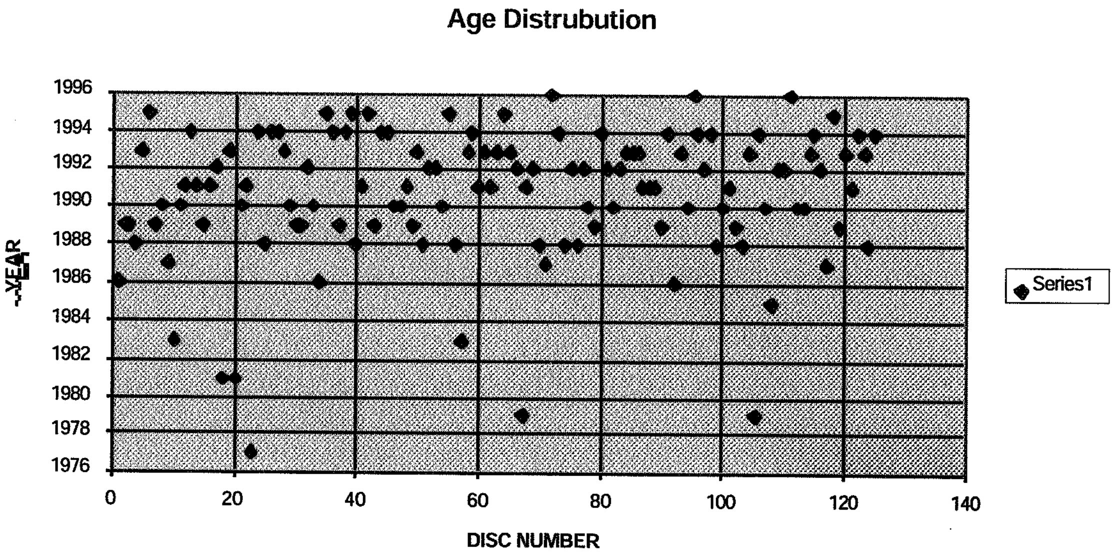


Figure 1. Scatter plot of age distribution.

3. Some test data

The test data have been collected for all the selected 125 discs. The full spectrum of errors has been tabulated for analysis, but are not included in this brief paper. However, a number of scatter plots are provided to present the range of certain parameter.

The Figure 1, represents the 125-disc sample in terms of the age. Since the sample of the test was statistically derived, it is hoped that this scatter plot does indeed represent the age distribution of the CD collection. Each disc is identified by a disc number, which is attached to the jewel case of the CD. It is hoped that the CD can be retrieved at a later date and repeated measurements made.

In looking at the data in the scatter plot, it can be seen that the age distribution ranges from 1977, disc number 23, to 1996 where there are 3 discs in the sample, disc numbers 72, 95, and 111.

It is often interesting or useful to look at specific parameters over the entire surface of the disc. The CD CATS system provides a surface presentation tool to look at how a parameter value may vary over the surface of the disc. This report does not include such presentations, but using the surface presentation of the BLER, for example, it can be used to correlate the variations of levels BLER (max.) with any other parameter, such as the E32 error. This is a useful mapping tool, but performed visually on a CRT.

Figure 2 shows the scatter plot of the maximum Block Error Rate (BLER). Except for the four discs with a BLER over 200, the data are very acceptable. These discs are disc numbers 18, 59, 80, and 93. Most CDs manufactured currently are exhibiting BLERs around 5 or less.

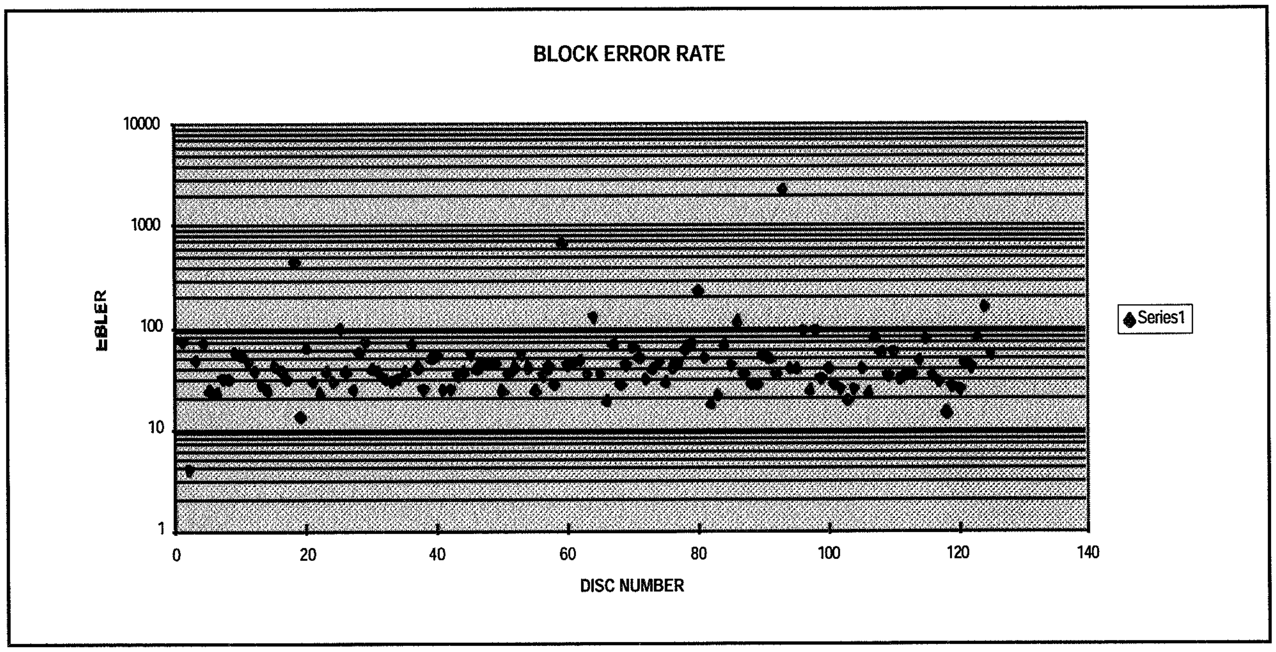


Figure 2. Scatter plot of Block Error Rate (BLER).

A closer look at the BLER for these discs show some interesting findings. The location of the BLER (max.) is shown along with the total playing time of the disc.

Disc number	BLER (max.)	Location time	BLER (av.)	Total playing time
18	444	55' 43"	67	56' 07"
59	1737	00' 04"	65	63' 52"
80	228	50' 32"	4	73' 10"
93	2304	41' 11"	1837	72' 47"

Often, it is said that errors occur near the end or edge of the disc. This is true for three out of the four discs that had BLER above 200. The BLER (av.) level is also too high for discs number 93. However, all these discs also had E32 errors, often near the location of the BLER (max.).

A closer look at the E32 error, which means an uncorrectable error, is in the following table. Discs 57 and 64 had acceptable BLER levels, but exhibited E32 errors. For Disc 57, the pit jitters were out of specification and for disc 64 both the land and pit jitters was out of specification.

Disc number	BLER (max.) at Location time	E32 at Location time	Total playing time
18	444 at 55' 43"	3 at 55' 54"	56' 07"
57	41 at 43' 20"	1 at 42' 52"	49' 40"
59	1737 at 00' 04"	20 at 00' 07"	63' 52"
64	127 at 25' 04"	317 at 25' 04"	64' 52"
80	228 at 50' 32"	409 at 50' 32"	73' 10"
93	2304 at 41' 11"	4 at 26' 11"	72' 47"

There are many other parameters to investigate, but is out of the scope of this short paper. Reflectivity is a critical factor and varies by types of substrates and reflective layers. Noise is also an important key parameters that will be later investigated.

4. Some conclusions

This report provides only the first, baseline testing of a small, 125, sample of discs. The monitoring and investigation of the CD collection will be an ongoing, low level, low priority, and low budget operation. Eventually, the entire collection should be monitored, not just a small sample.

First of all, it is probably desirable to increase the size of the sample, since certain assumptions were made to come up with a small sample size of only 125 discs as the first effort. The size should be increased to 1000 discs, for starts. In the sample of 125 discs there were 5 discs with one or more E32 errors. Since these are all audio discs, one cannot "hear" the "error", but nevertheless there is a measurable data error.

Secondly, the sample that is now measured should be monitored for any indication of deterioration. We have no such real life deterioration data, and coming back to these same discs every 3 to 5 years will be important and a significant contribution to the study of the life of CDs.

And finally, there should probably be a systematic procedure to continue randomly sampling the entire collection to find deteriorating discs. Should such a disc be found, a copy of it should be made. We have not yet progressed to that point.

Page intentionally left blank

On Configuring Hierarchical Storage Structures

Ali Esmail Dashti and Shahram Ghandeharizadeh
Computer Science Department
University of Southern California
Los Angeles, CA 90089
{dashti,shahram}@cs.usc.edu
tel +1-213-740-4781
fax +1-213-740-7285

Abstract

The rapid progress in mass storage technology is enabling designers to implement large databases for a variety of applications. The possible configurations of the hierarchical storage structures (HSS) are numerous and result in various tradeoffs, e.g., storage cost, throughput, initial latency, etc. A naive design might waste system resources and result in high cost. For example, one naive design decision is to add disk caches for those applications whose bandwidth is already satisfied by the tertiary storage device and can tolerate a high latency.

In this study, we introduce a configuration planner for HSS in support of video-on-demand applications. The input to the planner are the number of simultaneous displays required by the target application (i.e., throughput), database size, and expected pattern of access to the objects. It's output are the choice of mass storage devices at each level of the hierarchy. The resulting configuration supports the application requirements at a minimum cost per display. It is possible to extend this study to consider other application requirements, such as: initial latency and storage reliability. Moreover, the presented concepts can be generalized to scientific applications, such as those described in [3].

1 Introduction

Multimedia information systems, such as video-on-demand applications, are data intensive applications that are benefiting the most from the rapid progress in mass storage technology. One of the main design challenges of hierarchical storage structures (HSS), in support of video-on-demand application, is to guarantee continuous display of video objects at a minimum cost. To support continuous display, it is necessary to retrieve and display data at a pre-specified rate, otherwise, the display will suffer from frequent disruptions and delays, termed hiccups. To minimize the system cost, it is necessary to use the correct mixture of mass storage devices for a given target application requirements. For the past few years, techniques have been developed to support continuous display from disks, see [11, 1, 6, 8] for details. This study is an extension of those studies.

The remainder of the paper is organized as follows. In Section 2, we present a display-caching technique to support continuous display using tape libraries. In Section 3, we describe a configuration space of our planner. Finally, in Section 4, we describe a number of possible future extensions.

2 Continuous Display using Tape Libraries

Similar to magnetic disks, tape juke boxes are mechanical devices. However, the characteristics of a tape juke box is different from that of a disk drive [2, 10, 9]. The typical access time to data from a magnetic disk drive is in the order of milliseconds, whereas with a tape juke box, due to the overhead of mounting and dismounting tapes from the drives and its sequential nature, latency times in the order of seconds (minutes) is not uncommon. Due to the long latency time associated with tape access and due to the mismatch between the tape production rate, D_T, and the display consumption rate, C (i.e., Production Consumption Ratio [7], $PCR = \frac{C}{D_T}$), it is necessary to use caching to simulate continuous display from tapes. Otherwise, the display may suffer from hiccups. This display-caching (DC) should not be confused with object-caching (OC). With OC, complete objects are cached at different levels of the hierarchy to minimize the number of references to the slower devices in the hierarchy. Whereas with DC, the cache is used to bridge the gap between the retrieval behavior of tapes and the display requirement of the objects to guarantee a continuous display. Here, we focus on DC to describe: 1) how much data should be cached (prefetched) to guarantee continuous display from tape libraries, 2) where to maintain the cached data?, and 3) when the display should start? In the rest of this section, we address these three topics. We start with a brief description of the physical characteristics of the tape libraries.

Tape devices may operate in either multiplexing mode or streaming mode (non-multiplexing) [7, 12]. In the multiplexing-mode, data transfer is interrupted in the middle of an object retrieval to retrieve portions of other objects. The tape drive must locate the data by performing a sequential seek (T_{Search}) and possibly dismount the tape cartridge in the tape drive and mount another in its place (T_{Switch}) to access other objects. These actions increase the wasteful work performed by the tape drive (i.e., T_{Search} and T_{Switch}), and increase the wear and tear of the tape device. In the streaming-mode, the tape drive retrieves one object at a time, and, hence, at most one T_{Search} and T_{Switch} are performed per object retrieval. Here, we only consider operating the tape device in streaming-mode. (Note: the cost/stream might be lower when using multiplexing-mode.)

We describe display-caching requirements ($DC\_requirement$) for continuous display using tape libraries for two possible cases: the transfer rate of the tape drive is either 1) lower ($PCR < 1$), or 2) higher than the bandwidth required to display an object ($PCR > 1$). For each case, we consider the use of memory display-caching (DC_memory) and hierarchical display-caching ($DC\_hierarchical$), where $DC\_hierarchical$ consists of disk cache ($DC\_hierarchical_D$) and main memory cache ($DC\_hierarchical_M$). In our evaluation, we make the following assumptions:

- Data layout is contiguous on the tape, to facilitate long transfers.
- A tape juke box consists of one read/write drive, one tape switching mechanism (e.g., one robot arm), and a number of tape cartridges.

- Tape access time, T_{Access}, to an object is the sum of average T_{Search} and T_{Search}.
- All objects belong to a single media type, where the display requirement of an object is C and its size is O.

When $PCR \leq 1$, it is necessary to cache $(1 - PCR)$ of the object before starting its display. This is the minimum amount of caching required to guarantee hiccup free displays, as specified by the pipelining technique [7], i.e., $DC\_requirement = (1 - PCR) \times O$. With main-memory caching, $DC\_requirement$ is maintained in the main memory ($DC\_memory = DC\_requirement$); whereas with hierarchical-caching, $DC\_requirement$ is maintained on disk ($DC\_hierarchy_D = DC\_requirement$). The amount of main memory required for hierarchical caching is 4 blocks: 2 blocks for the retrieval from tape onto disk, and 2 blocks for display from disk ($DC\_hierarchy_M = 4$ blocks), where a block size is equal to a disk block.

When $PCR > 1$, the production rate is faster than the consumption rate and the display of the object may start as soon as the first portion of the object is retrieved (the size of the first portion is a disk block size). The climax of the caching requirement for one object retrieval is immediately after the completion of its retrieval. The size of this cache space is the difference between the object size and the portion of the object that has been displayed in parallel to its retrieval, $O - (\frac{O}{D_T} \times C)$.

It might be possible to retrieve other objects from the tape while displaying the first object from cache. Similar to the first case, the climax of the cache requirement for the second object is $O - (\frac{O}{D_T} \times C)$. This is reached after performing a tape access (T_{Access}) and an object transfer ($\frac{O}{D_T}$). Therefore, the total cache space requirement increases by $(O - (\frac{O}{D_T} \times C)) - (T_{Access} + \frac{O}{D_T}) \times C$. The minimum cache space required to guarantee continuous display for some N_T streams is:

$$DC\_requirement = \sum_{k=0}^{N_T - 1} ((1 - \frac{1}{PCR}) \times O - k(T_{Access} + \frac{O}{D_T}) \times C). \qquad (1)$$

The maximum number of simultaneous displays from the cache space, N_T, is: $N_T = Max(1, \lfloor \frac{B_T}{C} \rfloor)$, where B_T is the effective bandwidth of the tape drive, $B_T = \frac{O}{T_{Access} + \frac{O}{D_T}}$.

With main-memory caching $DC\_memory = DC\_requirement$, and with hierarchical caching $DC\_hierarcical_D = DC\_requirement$. The main memory requirement for hierarchical caching is equal to $N_T + 1$ blocks (for the display of the N_T streams from a disk cache), and some memory for the retrieval of the data from tape onto disk. Assuming that the main memory can flush data to the disk at the same speed as the transfer rate of the tape (or faster), then the amount of main memory for object retrieval from tape onto disk is 2 blocks per disk. Therefore, the total amount of main memory required for hierarchical caching is $N_T + 3$ blocks, for a single disk cache.

3 Configuration Space

For the purpose of this evaluation, we make the following assumptions:

N	50 Gbyte	100 Gbyte	500 Gbyte	1 Tbyte	5 Tbyte	10 Tbyte
10	$n_M = 1$ $n_D = 6$	$n_M = 1$ $n_D = 3$ $n_T = 1$	$n_M = 1$ $n_D = 3$ $n_T = 1$	$n_M = 1$ $n_D = 4$ $n_T = 2$	$n_M = 1$ $n_D = 5$ $n_T = 6$	$n_M = 1$ $n_D = 4$ $n_T = 11$
50	$n_M = 2$ $n_D = 6$	$n_M = 3$ $n_D = 11$	$n_M = 3$ $n_D = 24$ $n_T = 1$	$n_M = 2$ $n_D = 10$ $n_T = 2$	$n_M = 3$ $n_D = 16$ $n_T = 6$	$n_M = 3$ $n_D = 19$ $n_T = 11$
100	$n_M = 4$ $n_D = 6$	$n_M = 4$ $n_D = 11$	$n_M = 5$ $n_D = 38$ $n_T = 1$	$n_M = 5$ $n_D = 44$ $n_T = 2$	$n_M = 3$ $n_D = 20$ $n_T = 6$	$n_M = 6$ $n_D = 30$ $n_T = 11$
250	$n_M = 9$ $n_D = 9$	$n_M = 9$ $n_D = 11$	$n_M = 10$ $n_D = 50$ $n_T = 1$	$n_M = 11$ $n_D = 81$ $n_T = 2$	$n_M = 12$ $n_D = 51$ $n_T = 16$	$n_M = 11$ $n_D = 63$ $n_T = 11$
500	$n_M = 17$ $n_D = 18$	$n_M = 17$ $n_D = 18$	$n_M = 18$ $n_D = 56$	$n_M = 19$ $n_D = 99$ $n_T = 2$	$n_M = 22$ $n_D = 101$ $n_T = 32$	$n_M = 22$ $n_D = 101$ $n_T = 32$
750	$n_M = 25$ $n_D = 27$	$n_M = 25$ $n_D = 27$	$n_M = 27$ $n_D = 56$	$n_M = 28$ $n_D = 111$	$n_M = 33$ $n_D = 148$ $n_T = 47$	$n_M = 33$ $n_D = 148$ $n_T = 47$
1000	$n_M = 33$ $n_D = 36$	$n_M = 33$ $n_D = 36$	$n_M = 34$ $n_D = 56$	$n_M = 39$ $n_D = 111$	$n_M = 45$ $n_D = 420$ $n_T = 6$	$n_M = 44$ $n_D = 198$ $n_T = 63$
2500	$n_M = 82$ $n_D = 90$	$n_M = 82$ $n_D = 90$	$n_M = 82$ $n_D = 90$	$n_M = 84$ $n_D = 111$	$n_M = 95$ $n_D = 509$ $n_T = 6$	$n_M = 108$ $n_D = 908$ $n_T = 11$
5000	$n_M = 732$ $n_D = 0$	$n_M = 163$ $n_D = 179$	$n_M = 163$ $n_D = 179$	$n_M = 163$ $n_D = 179$	$n_M = 174$ $n_D = 556$	$n_M = 190$ $n_D = 1022$ $n_T = 11$
7500	$n_M = 732$ $n_D = 0$	$n_M = 243$ $n_D = 268$	$n_M = 243$ $n_D = 268$	$n_M = 243$ $n_D = 268$	$n_M = 261$ $n_D = 556$	$n_M = 278$ $n_D = 1111$
10000	$n_M = 732$ $n_D = 0$	$n_M = 1473$ $n_D = 11$	$n_M = 325$ $n_D = 358$	$n_M = 325$ $n_D = 358$	$n_M = 331$ $n_D = 556$	$n_M = 348$ $n_D = 1111$

Table 1: Configurations space, using Zipf distribution with d=0.271.

- The database consists of single media type, where $C = 0.5$ Mbyte/sec and $O = 3.6$ Gbyte.
- The database adheres to a strict memory inclusion policy; such that objects on main memory are a subset of the objects on secondary memory, and objects on secondary memory are a subset of objects on tertiary memory (i.e., OC adhere to strict inclusion policy).
- When available on multiple memory levels, objects are displayed from the fastest memory level only, even when the other memory levels are idle.
- Main memory consists of 64 Mbyte DRAM modules, where the cost of a module is ≈ 400.
- Secondary memory consists of 9 Gbyte disk drives, where the cost of a drive is ≈ 1500. To support continuous display, we assume that disk drives are configured with 2 Mbyte blocks and each disk supports 28 simultaneous displays.
- Tertiary memory consists of 980 Gbyte tape juke boxes, where the cost of each juke box with its cartridges is $\approx 10,000$. Tape access time is 82 seconds, $T_{Access} = 82$, and transfer rate is 10 Mbyte/sec, $D_T = 10$. We assume hierarchical caching, and the maximum number of streams that can be supported by the cache space is 16 displays.

In Table 1, we show HSS configurations for a range of database sizes (from 50 Gbyte up to 10 Tbyte), and a range of desired throughput (from 10 simultaneous displays up to 10,000 displays). For every given database size and desired throughput, the number of storage modules required to hold the database and support continuous display are shown, where n_M, n_D, and n_T are the number of memory modules, disks, and tape juke boxes. We use Zipf distribution to model the access pattern to the HSS, where the frequency of access to an object j is defined to be $F(j) = \frac{c}{j^{1-d}}$ (c is the normalization constant and d controls access frequency drop off). We set $d = 0.271$ to closely match movie rental access pattern [4].

In Table 1, for a range of databases (i.e., 500 Gbyte to 10 Tbyte) and low throughput requirements (i.e., 10-100 displays, depending on the database size), it is not necessary to use disk caches to store complete objects, i.e, no OC is required. The basic tape bandwidth is sufficient for the application requirement. As mentioned earlier, some DC is required to guarantee the continuous display from the tape libraries. However, for the given databases, as the throughput requirement increases, the references to the working set increases and the tapes might become a bottleneck. In this case, it is possible to either: 1) add more disks and/or memory to cache the working set (i.e., OC), or 2) add more tapes and tape drives (i.e., tape replication). For large databases, the later provides the most cost effective solution for low throughput requirements (up to 1000 displays). However, displaying directly from tapes has the following disadvantages: 1) high initial latency, 2) support for video-on-demand functionality becomes complex. Therefore, the utility of displaying directly from tapes is limited by the type of application being considered. For higher throughput requirements, OC on disks provides the best solution for a wide range of requirements, whereas OC on main memory provides the best solution for small databases and very high throughput requirements (e.g., 100 Gbyte and 10,000 display).

4 Future Work

As part of our future research, we will consider the following extensions. First, our current focus was on optimizing for cost per display. It is possible to extend our configuration planner to consider other application requirements, such as initial latency and storage reliability. For example, we expect that storage reliability to be inversely proportional to the data access from tertiary devices [5], and, hence, tape devices should be accessed less frequently. Second, we plan on studying the effects of a changing data access pattern and database size on the optimal configuration. Finally, we plan on studying the performance effects of data placement on tapes.

Acknowledgments

We thank the referees and, specially, Rodney Van Meter for his insightful comments. This research was supported in part by the National Science Foundation under grants IRI-9258362 (NYI award) and ERC grant EEC-9529152, and a Hewlett-Packard unrestricted cash/equipment gift.

References

[1] S. Berson, S. Ghandeharizadeh, R. Muntz, and X. Ju. Staggered Striping in Multimedia Information Systems. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 79–90, May 1994.

[2] M. Carey, L. Haas, and M. Livny. Tapes Hold Data, Too: Challenges of Tuples on Tertiary Storage. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 413–417, May 1993.

[3] R. Chambers and M. Davis. Petabyte Class Storage at Jefferson Lab. In *Proceedings of the 5th NASA GSFC Mass Storage Systems and Technology Conference*, September 1996.

[4] A. Dan, D. Sitaram, and P. Shahabuddin. Scheduling Policies for an On-Demand Video Server with Batching. In *Proceedings of the ACM Multimedia Conference*, pages 15–23, October 1994.

[5] A. L. Drapeau and R. H. Katz. Striped Tape Arrays. In *Proceedings of the Twelfth IEEE Symposium on Mass Storage Systems*, April 1993.

[6] D. J. Gemmell, H. M. Vin, D. D. Kandlur, P. V. Rangan, and L. A. Rowe. Multimedia Storage Servers: A Tutorial. *IEEE Computer*, 28(5), May 1995.

[7] S. Ghandeharizadeh, A. E. Dashti, and C. Shahabi. A Pipelining Mechanism to Minimize the Latency Time in Hierarchical Multimedia Storage Managers. *Computer Communications*, 18(3), March 1995.

[8] S. Ghandeharizadeh, R. Zimmermann, W. Shi, R. Rejaie, D. Ierardi, and T.-W. Li. Mitra: A Scalable Continuous Media Server. *Multimedia Tools and Applications Journal*, 5(1):79–108, July 1997.

[9] B. Hillyer and A. Silberschatz. On the Modeling and Performance Characteristics of a Serpentine Tape Drive. In *Proceedings of the ACM Sigmetrics International Conference on Measurement and Modeling of Computer Systems*, pages 170–179, May 1996.

[10] S. Sarawagi and M. Stonebraker. Reordering Query Execution in Tertiary Memory Databases. In *Proceedings of the 22nd VLDB Conference*, pages 156–167, September 1996.

[11] F. Tobagi, J. Pang, R. Baird, and M. Gang. Streaming RAID-A Disk Array Management System for Video Files. In *Proceedings of the First ACM Multimedia Conference*, August 1993.

[12] P. Triantafillou and T. Papadakis. On Demand Data Elevation in Hierarchical Multimedia Storage Server. In *Proceedings of the 23rd VLDB Conference*, pages 226–235, August 1997.

The Berkeley-San Francisco Fine Arts Image Database

Nisha Talagala, Satoshi Asami, David Patterson
{nisha,asami,patterson}@cs.berkeley.edu
Computer Science Division
University of California at Berkeley, Berkeley, CA 94720
Tel: +1 510 642-1845

Bob Futernick, Dakin Hart
{bfuternick, dakin}@famsf.org
Fine Arts Museums of San Francisco
233 Post Street, San Francisco, CA 94108
Tel: +1 510 415 750-3508

Abstract

This paper describes the world's largest on-line art collection, provided by a collaboration between the Fine Arts Museums of San Francisco and the University of California at Berkeley. The collection contains over 70,000 images of artworks and is accessible over the web. Each image will be available at a resolution of upto 3,000 by 2,000 pixels. This paper describes the storage system and the image database that make up the web site.

1. Introduction

This paper describes a collaboration between the University of California at Berkeley and the Fine Arts Museums of San Francisco, to provide a large collection of high-resolution images over the web. Over the past few years, the Fine Arts Museums have photographed over 70,000 art objects and stored the images on CD-ROMs in PhotoCD format. Each image is 3,000 x 2,000 pixels and about 4.5 MB in size. Because of limited storage, the museum offered only low-resolution versions of the images online: 2KB thumbnails and 20-50KB JPEGs. The larger versions of the images were kept only on CD-ROMs.

Visitors to the web site can search the image database by artist, title, or description of an artwork. The image database is the largest on-line art collection in the world. Since the web site's launch in November 1996, many users have requested larger versions of the images. To satisfy this demand, the museum is collaborating with UC Berkeley to create an http server for the larger images.

The high-resolution images are stored on the Tertiary Disk prototype [2], a three terabyte disk storage system developed at UC Berkeley. The name of the prototype comes from twin goals, to achieve the cost/megabyte of tape systems and the performance of disk arrays. The Tertiary Disk prototype is a large distributed storage system created from commodity components.

The following sections describe the storage system and the image server implementation. Section 2 describes the storage system design. Section 3 describes the image database and http servers. Section 4 gives the current status and future work.

2. The Storage System

Over the past five years, disk drive prices have been falling at nearly a factor of two per year [1]. This rapid decrease makes large scale disk systems a feasible alternative to tape libraries. However, commercially available disk storage systems, like hardware RAID arrays, have a much higher cost per megabyte than the underlying disks. Finally, new disks for RAID arrays usually have to be obtained from the manufacturer, to maintain the warranty. This practice increases the cost of expansion and the lag time to incorporate new disks.

The performance of a disk array is limited by its host connection(s). In most cases the bandwidth of the host connection does not match the bandwidth of the disks. Disk bandwidths are increasing dramatically due to higher rotation speeds, higher areal densities, and better electronics [1]. If these trends continue, the host interconnect will become an increasing performance bottleneck for RAID arrays.

Tertiary Disk is a storage system design that avoids these problems. The prototype is a three terabyte disk system with a cost/megabyte close to that of the underlying disks. The basic idea is to use commodity hardware to lower cost and software to manage and monitor the system. The design is based on a group of relatively independent storage nodes. Independent nodes make the system more scalable and allow incremental expansion. Also, this design increases the number of links from the storage system to the outside world, improving bandwidth. The prototype proves by example that a large storage system can be built using off-the-shelf hardware.

The design uses PCs connected through a switched network, with each PC hosting a group of disks. The prototype contains 20 PCs and 368, eight gigabyte disks. The PCs are interconnected through 10 Mbit/s switched Ethernet. (We will soon upgrade to a 100 Mbit/s network). The disks are connected to the host machines using Fast-Wide SCSI. To improve availability, each disk is connected to two PCs. Each SCSI bus is shared, with two SCSI controllers on the bus. If one host in unable to service requests, all the disks can be accessed from the second host. In normal mode, each host accesses half the disks (i.e., there is no disk sharing). More details about this design are available in [2].

The 20 hosts are organized into 10 nodes of 2 machines each. There are two types of nodes, *light* and *heavy*. Each light and heavy node contains 32 and 56 disks, respectively. Figure 1 shows a light node. The light and heavy nodes were used to study the cost/performance trade-offs in changing the number of disks per host. A performance study of the nodes is also available in [2]. The prototype contains eight light nodes and two heavy nodes.

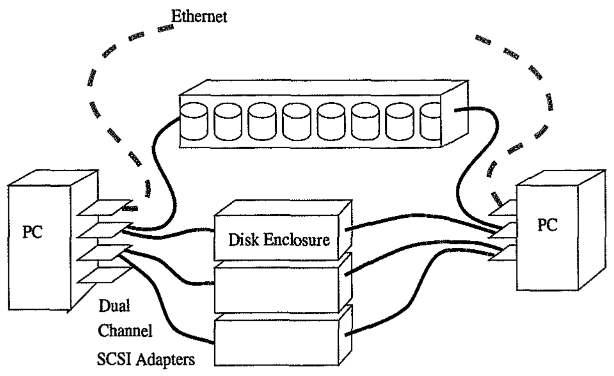


Figure 1. Design of a light node.

Management is one of the biggest challenges for a system like this. To simplify management and monitoring, we use programmable disk enclosures. Each enclosure has a serial port interface which supports a set of commands. Using this interface, the status of all enclosures can be monitored from a central location.

The application for this storage system is the Fine Art Image Database. Over 70,000 images, in different formats, are stored on this prototype. The next section describes the image database.

3. The Image Database

The prototype is used to serve the high-resolution images of each artwork over the web. Before the images are put on-line, they are processed by a half-dozen undergraduate students. This processing removes problems that may have occurred while the images were being photographed. The images are cropped, rotated to the correct orientation, and enhanced using color correction. Since this process is very subjective, it is hard to automate, and each image is processed by hand using Adobe PhotoShop. Once the images are processed, they are converted to a layered format that is presentable over the web.

When the museum first launched its web site, they chose not to put the larger images on-line for two reasons. The first was the lack of disk space. The second was the lack of a proper user interface. It is not practical to serve the user an entire 3,000 by 2,000 pixel image. Many users do not have the network bandwidth to support such transfers (it would take a user half an hour to download a 6 MB image with a 28.8k modem). Also, no display would be large enough to hold the full-size image. Finally, there are copyright restrictions that prevent high quality images of many of the art works from being freely distributed. These problems are avoided by converting the images to a layered format that allows zoom-in.

The images are stored in the GridPix format [3], a special format developed by the Tertiary Disk group. In this format, an image is stored as a series of tiles, each tile encoded in JPEG. A GridPix file contains header information and a tile index, followed by a series of tiles in various resolutions. The GridPix viewer is a CGI-script that generates an HTML page describing an array of tiles. A second CGI-script retrieves the requested tiles from the GridPix file. Figure 2 shows the viewer. The page contains tiles inside a fixed size window. At lower resolutions, the image contains only a few tiles and fits completely inside the window. At higher resolutions, only a part of the image fits inside the window. Hence the user can study parts of an artwork in detail. Only the part of the image that is visible to the user is transferred over the network. One of the biggest advantages of GridPix is that caching is done entirely by the browser. When a user scrolls around an image that is too large to fit entirely within the viewer's window, only the new tiles that appear need to be retrieved from the server.

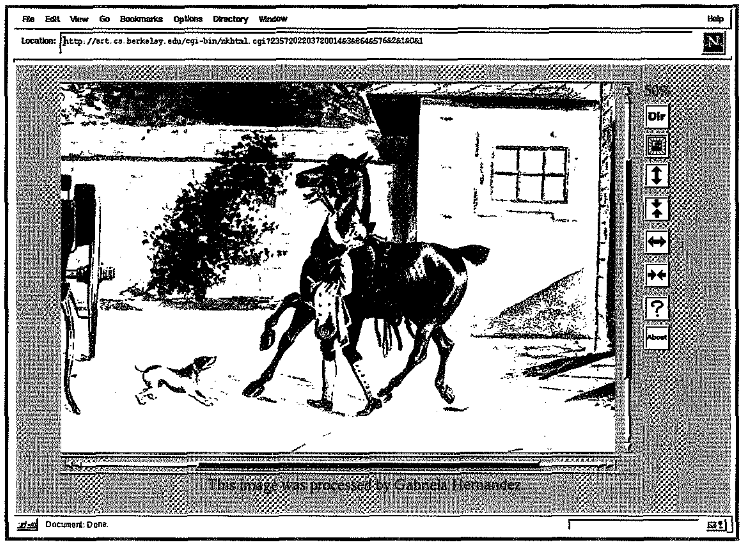

File Edit View Go Bookmarks Options Directory Window Help
Location: http://art.cs.berkeley.edu/cgi-bin/zkbtml.cgi?23572022037200143&864&576&2&1&0&1
50%
Dir
?
About
This image was processed by Gabriela Hernandez
Document: Done.

Figure 2. The GridPix Viewer

GridPix was developed after considering an alternative format, FlashPix [4]. FlashPix is a proprietary format developed by Eastman Kodak, Microsoft, and others. FlashPix also allows multiple resolutions of tiled images, but has several disadvantages. First, the current implementation requires a server and browser plug-in, limiting the types of machines that can be servers and clients. Second, since FlashPix is a general format that provides many features in addition to zoom-in, the files are quite large. Each of our images occupies 4-6 MB of space in FlashPix and 1-2 MB of space in GridPix.

The images are stored in several formats. Only the GridPix images are available to the outside world. Each image is also stored in TIFF format and PhotoCD format. The GridPix and TIFF versions are mirrored, i.e. two copies of each image are kept. Only a single copy of the PhotoCD images is kept because of space considerations. The GridPix images are generated from their TIFF counterparts. Since GridPix images cannot be edited using PhotoShop, the TIFF images make it possible to make changes later if necessary.

The homepage and search engine for the museum are located at the Fine Arts Museum (http://www.famsf.org/), while the large images are served by a group of http servers at Berkeley (http://art.cs.berkeley.edu). Each host in the prototype is an http server. The images from each PhotoCD form a *folder*. Each folder contains around 100 images and has a unique ID. Each image is contained in a single GridPix file. Concatenating the ID number of the folder and the image's number within the folder creates the unique key for each image. An image's key is translated into a URL for the image.

4. Status and Future Work

We are currently processing the images and converting them into GridPix format. The processed images are available to users through a web site at Berkeley (http://art.cs.berkeley.edu/). By February 1998, this site will be linked to the museum's main web site, with approximately 30% of the collection available in the layered format. From then on, images will be added to the site as they are processed. The search index at the main web page will allow users to search for those images that have the high-resolution counterparts available.

We also plan to offer a Java applet viewer as an alternative to the GridPix viewer. Once the web site is available to all users, we will study usage patterns and load balancing problems for this workload.

5. Acknowledgements

The image processing work is being done by UC Berkeley undergraduate students Gabriela Hernandez, Nicholas Huynh, Tony Le, Shankara Lowe, Victor Wong, and Wen-Kai Zhong. The art works were photographed by Sue Grinols of the Fine Arts Museums of San Francisco.

We wish to thank IBM for donating the disk drives used in Tertiary Disk and Intel for donating the PCs. The Tertiary Disk research is performed in conjunction with the Network of Workstations Project at UC Berkeley and is funded by the DARPA N00600-93-K-2481 Roboline-Storage contract.

References

[1] Grochowshi, E. Hoyt, R. Future Trends in Hard Disk Drives. *IEEE Transactions* on Magnetics, Vol. 32, No. 3. May 1996.

[2] Talagala, N. Asami, S. Patterson, D. Anderson, T. Tertiary Disk: Large Scale Distributed Storage. University of California at Berkeley, February 1997. Available at (http://now.cs.berkeley.edu/Td/papers/).

[3] Asami, S. The San Francisco-Berkeley Fine Arts Project, October 1997. Available at (http://now.cs.berkeley.edu/Td/talks/).

[4] FlashPix Specification. Kodak Corporation, 1997.

Page intentionally left blank

Eliminating the I/O Bottleneck in Large Web Caches

Alex Rousskov and Valery Soloviev
Computer Science Department
North Dakota State University
Fargo, ND 58105-5164
{rousskov,soloviev}@plains.NoDak.edu

Abstract

This paper presents a technique for eliminating the disk bottleneck in large Web Caches. Our approach objective is twofold. First, the presented algorithm substantially decreases disk activity during peak server load. Second, it maintains the hit ratio at the level of traditional caching policies. We evaluate the performance of the algorithm using trace driven simulations based on access logs from several top-level Web caches.

1 The Problem

Caching servers or *caching proxies* are now standard tools for handling the exponential growth of the Web traffic. Individual caching proxies can boost their performance by joining cache *hierarchies*. Several large hierarchies or *cache meshes* are currently operational [1, 2].

In a cache mesh, intermediate servers have to serve data to all proxies they cooperate with. To be effective, an intermediate cache server must handle gigabytes of traffic per day and maintain a large storage of cached documents. The traffic volume is rapidly increasing with the Web growth and as new servers join the hierarchy. Due to the system overhead, it may take about four disk I/Os to cache or retrieve a single document. It is no surprise that the disk subsystem becomes a serious bottleneck in a large caching proxy [3, 4].

In the nearest future, multimedia data such as audio and video clips are expected to become a major part of the Internet traffic. Caching proxies could be naturally used for caching and *smoothing* the delivery of delay-sensitive media. This new media, along with the Web traffic growth, will increase the burden on caching proxies, especially on their disk subsystem.

Current caching schemes rely on algorithms originally designed for *in-memory* caches in database and file systems [5, 6]. Thus, traditional algorithms ignore performance problems associated with maintaining large *disk-resident* archives of cached documents.

This paper demonstrates that traditional caching algorithms create excessive load on a caching server and do not scale with the increase in the volume of the Web traffic. We present a new caching algorithm designed to minimize I/O activity on a server while maintaining the hit ratio at the level of traditional algorithms.

2 Traditional Approach

Many Web caching algorithms have been proposed. Most algorithms adopt techniques found in database and file systems [5, 6]. A few recent studies take the specifics of the Web traffic into account [7]. All existing algorithms optimize the hit ratio and ignore I/O activity.

A common pattern among traditional algorithms is to store *every* incoming cachable document while purging the previously cached ones to free disk space. Thus, there is a persistent flow of data to the disk-resident cache. Clearly, the volume of disk traffic is proportional to the Web traffic. That is, when the number of requests to a caching proxy increases, so does the number of documents written to the disk cache. Disk queues grow exponentially and so does disk response time [4]. This results in an I/O bottleneck system.

The I/O bottleneck leads to the performance degradation of the entire caching proxy. When the disk subsystem cannot handle incoming requests, the response time of a single request increases. A longer response time means more concurrent requests in the system. The latter leads to shortages in buffer memory, CPU slices, file descriptors, etc. As the queuing theory suggests, these resource shortages increase *exponentially* when the server load is high. Thus, the performance problems are most severe during the peak server load.

To compensate for the increase in the load during peak hours, the system must have excessive disk, memory, and CPU resources. These resources are not utilized for the rest of the time.

3 Proposed Algorithm

This section presents an algorithm designed to eliminate the I/O bottleneck in large caching proxies. Our design objective is twofold. First, we want to substantially decrease disk activity during peak server load. Second, we want to maintain the hit ratio at the level of other caching algorithms.

The traditional approach is to write *every* new cachable document to disk. In a large hierarchy, the majority of Web documents are requested from an intermediate caching proxy only once. Thus, these documents are written to the disk but are never read back. If we could predict future requests, we would never write such documents and would drastically reduce disk activity.

A precise prediction of the Web traffic is, of course, not feasible. However, our analysis shows that most documents requested today were requested in the past as well. Thus, by maintaining an *active set* of previously requested documents, we could "predict" the future traffic. We can rebuild the active set only once per day when the server load is negligible. We propose `Static Caching` algorithm that works as follows.

- Once per day, when the server load is negligible, scan the *log file* of a caching proxy to determine a set of URLs (names) of previously accessed documents.
- Form an active set by selecting URLs of most *valuable* documents among those scanned in the first step. A value of a document is determined by its contribution towards the total number of hits weighted by the document size. Such a value would guarantee an optimal hit ratio if the future traffic matches the past precisely.
- If a document from the active set is not currently cached, then it can be either prefetched when the active set is formed or fetched during the first corresponding request. We assume the first scenario.
- During the day, the active set of URLs remains unchanged or *static*. A request to a cached document from the set produces a *hit*. A request to a document outside of the active set gives a *miss*.

`Static Caching` shifts all major disk and CPU activities from peak load hours to the time when a caching proxy is idle. During the day, the disk activity is triggered by *hits* and *updates* of active documents only. The former are essential to maintain a high hit ratio. The latter are rare and have negligible impact. Thus, there is no excessive disk activity during peak load.

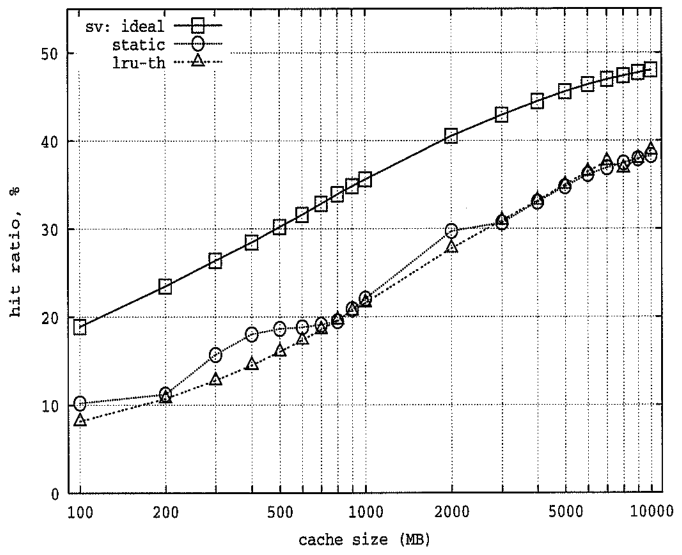
As a side effect, the CPU load is minimal during peak hours. A single hash table lookup is required to process a request. No work on maintaining the hash table or any other meta data is required. A single hash table is all that has to be stored in memory buffer. Thus, memory requirements are also minimal. Other benefits of the approach include a potential for exchange of active sets among cooperative caching proxies to optimize the cooperation and for compression of cached documents.

4 Performance Analysis

Using trace driven simulations, we have compared the performance of the `Static` algorithm with the `LRU-TH` algorithm [8]. `LRU-TH` and its variations are traditionally used in caching proxies [9, 3]. Due to cache sizes exceeding daily traffic volume, many other known algorithms will mimic the behavior of `LRU-TH`. In our previous work with *primary* Web servers, we observed a similar performance of many algorithms on large caches [10].

We used traces from six *root* servers of a large cache hierarchy maintained by a National Laboratory for Applied Network Research [1]. Traces were produced by the Squid caching proxy, a freeware successor of Harvest [9]. Squid is currently the best freeware proxy accounting for about 70% of all European caches [11]. Traces' duration was about two months. Each server studied has distinct traffic patterns. For example, the `SV` server handled mostly international traffic and processed more than 6 GB per day (more than 600,000 requests). On the other hand, the `LJ` server had an open access list and processed almost 2 GB per day (about 150,000 requests). NLANR servers had 6-8GB allocated for disk cache.

For each URL request, Squid logs the time of the request, the URL, the size of the document, and the performed action [1]. The last modification date of a document is not logged, but updates can be usually detected by analyzing the action field. Furthermore, for every request, we compared the size of the requested file with the one cached by an algorithm. A change in the file size indicates a modification and results in a "miss".



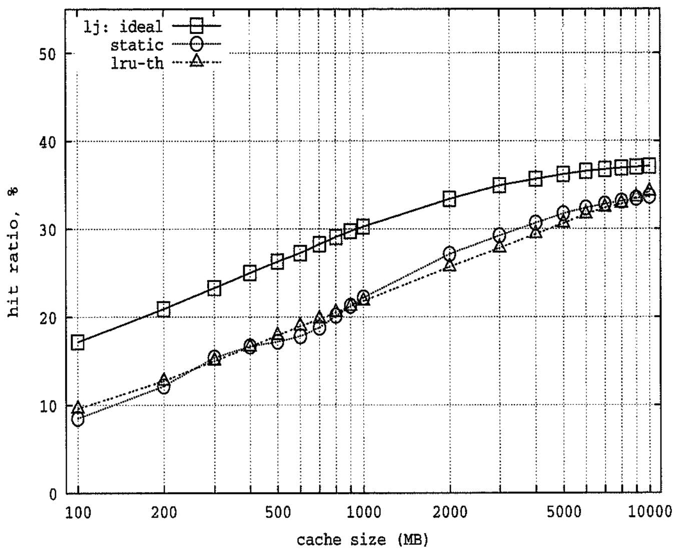


Figure 1: `Static` vs. `LRU-TH` on `SV` and `LJ` servers

Figure 1 compares the performance of the `Static` algorithm with the `LRU-TH` algorithm on the `SV` and `LJ` servers. Performance on other servers studied is similar. `LRU-TH` is shown with the *best* threshold for each cache size. Solid lines show an upper bound on `Static` performance in a hypothetical case when the future is 100% predictable (i.e., we know what *old* URLs will be accessed again; there are still *new* URLs that can be cached by `LRU` but cannot be cached by `Static`).

Clearly, elimination of excessive I/O load by `Static` did not sacrifice the hit ratio. We also measured the number of disk I/Os produced by the algorithms. `Static` consistently reduced daily disk traffic by about 35%. The savings would become even bigger when only peak load is considered. Such a reduction would eliminate disk bottleneck in a real proxy environment.

5 Conclusions and Future Work

The expansion of the Web and increasing presence of multimedia information result in the I/O bottleneck on caching proxies. Traditional caching algorithms ignore the performance of the I/O subsystem and do not scale with the increase in traffic volume and intensity. We have presented the `Static Caching` algorithm which focuses on saving disk bandwidth. `Static Caching` eliminates the I/O bottleneck while preserving a high hit ratio.

Currently, we are investigating the application of `Static` algorithm to *leaf* proxy caches installed in large universities and corporations. Leaf caches handle more *dynamic* traffic than intermediate caching proxies. An interesting *symbiosis* of the `Static` algorithm and a traditional dynamic caching policy may be needed to maintain a high hit ratio with minimum resource requirements.

Acknowledgements

This work was supported, in part, by NSF grants `OSR-95-53368` and `RIA IRI-94-09845`.

References

[1] National Laboratory for Applied Network Research.
the lab: `http://www.nlanr.net/`, cache project: `http://ircache.nlanr.net/`

[2] Europe Caching Hierarchy
`http://www.terena.nl/projects/choc/`

[3] Carlos Maltzahn, Kathy Richardson, and Dirk Grunwald. Performance Issues of Enterprise Level Web Proxies. *In Proceedings of the ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*, Seattle, June 1997
`http://www.cs.Colorado.edu/ carlosm/sigmetrics.ps.gz`

[4] Alex Rousskov and Valery Soloviev. On Performance of Caching Proxies. Submitted for publication.
`http://www.cs.ndsu.nodak.edu/~rousskov/research/cache/squid/profiling/papers/`

[5] E.O'Neil, P. O'Neil, and G. Weikum. The LRU-K Page Replacement Algorithm For Database Disk Buffering. *In Proceedings of the ACM SIGMOD Int. Conf. on Management of Data*, Washington, May 1993.
`http://paris.cs.uni-sb.de/public_html/papers/LRU-k_report.ps.Z`

[6] R. Karedla, J. Love, and B. Werry. Caching Strategies to Improve Disk Performance. *IEEE Computer*, pp. 38-46, v.27(3), March 1994.

[7] J. Pitkow and M. Recker. A Simple Yet Robust Caching Algorithm Based on Dynamic Access Patterns. *In Proceedings of the Second International WWW Conference*, Chicago, October 1994.
`http://www.ncsa.uiuc.edu/SDG/IT94/Proceedings/DDay/pitkow/caching.html`

[8] M. Abrams, C. Stanbridge, G. Abdulla, S. Williams, and E. Fox. Caching Proxies: Limitations and Potentials. *In Proceedings of the Fourth International Conference on the WWW*, Boston, December 1995.
`http://ei.cs.vt.edu/~succeed/WWW4/WWW4.html`

[9] A. Chankhunthod, P. Danzig, C. Neerdaels, M. Schwartz, and K. Worrel. A Hierarchical Internet Object Cache. *USENIX Annual Technical Conference*, San Diego, January 1996.
`http://excalibur.usc.edu/cache-html/cache.html`

[10] Igor Tatarinov, Alex Rousskov, and Valery Soloviev. Static Caching in Web Servers. In proceedings of *the IEEE International Conference on Computer Communications and Networks, Las Vegas, September 1997.*
`http://www.cs.ndsu.nodak.edu/~tatarino/pubs/static.ps`

[11] European Caching Task Force Survey Results:
`http://w3cache.icm.edu.pl/survey/results/`

Page intentionally left blank

Building and Managing High Performance, Scalable, Commodity Mass Storage Systems

John Lekashman
NAS Systems Division
NASA Ames Research Center
Mail Stop 258-5
Moffett Field, CA, 94035
lekash@nas.nasa.gov
Tel: +1 650-604-4359
Fax: +1 650-604-4377

Abstract

The NAS Systems Division has recently embarked on a significant new way of handling the mass storage problem. One of the basic goals of this new development are to build systems at very large capacity and high performance, yet have the advantages of commodity products. The central design philosophy is to build storage systems the way the Internet was built. Competitive, survivable, expandable, and wide open.

The thrust of this paper is to describe the motivation for this effort, what we mean by commodity mass storage, what the implications are for a facility that performs such an action, and where we think it will lead.

We believe the implications for the future are that it will become much easier to use storage, resulting in extreme growth in deployed massive storage capacity,

1. Introduction

This paper is about managing commodity, high performance, long term storage.

Commodities are things that you can get from a lot of different places.

Management is something you can hardly get anywhere.

High Performance is an elusive target. In cars, it often inspires a sense of pleasure to be driving a high performance vehicle. In computing systems, it, usually meaning that the thing in question is performing its intended function well enough that something else gets on your nerves.

2. Our work

We feel our work is very useful, because we have figured out how to put all three of these together.

We're the first ones to build such a system, where the proprietary nature of the system is limited to a given system unit, such that it no longer will annoy us.

2.1. Our data is ours.

No one can hold it hostage. The key point of building data storage systems, after all, is to store your data. Since we are the ones doing the data storage, it only makes sense that its our data, and not subject to random external constraints, licensing arrangements, or other encumbrances that make facility management irritating.

This means that all the hardware and software interfaces have to be wide open. Things like TCP/IP, RAID, SCSI, Fibre Channel, OpenVault, DMAPI, and so on, are key, and must be the total definition of the system.

No vendor proprietary file systems, no charging per byte stored, no vendor specific implementation can exist in the system across an interface boundary. And, the whole part that any vendor provides must conform to the interfaces that are defined to the other parts.

So, it is perfectly ok for the vendor of a host platform that accesses your data to have their own native, proprietary file system. Its not ok for your data to be stored in that file system in such a way that it takes more than a day or so to read all the data out of that file system. More than a day or so will annoy you, and then its not high performance.

2.2. Scale matters, both up and down.

Table 1 shows some current choices for the various parts of the hardware system.

Figure 1 shows how to connect these parts, in a relatively generic fashion.

Its important to note that there is wide dynamic range in what can be done. At the low end, a storage system with a capacity of 1/2 a terabyte can be built for less than $30K, using PCs, ethernet, and 4mm tape changers. At the high end, a storage system could be built for $15M with a capacity of 3 Petabytes. The hardware cost of the low end system is about 2 cents/MB, the high end 5 cents/MB. One can continue to scale up the size, however, there are few cost reductions. This kind of cost pattern, where the size has a range over five orders of magnitude, the overall cost per unit drops some, due to large purchasing power, shows the hardware involved in these systems is truly commodity based.

This then, is the hardware commodity base. The software, however, in current large scale storage systems, is anything but a commodity. There are a number of different vendors, each pursuing their narrow customer base. Their data formats are incompatible, their file systems proprietary, or only run on one kind of very specific hardware. When one discusses transitions with operations managers, they talk in terms of years, and hundreds of thousands of dollars to effect the transition. This then, is nothing like a commodity, and really, nothing like high performance. Remember, high performance things are not annoying. Years of transition, and much cost, are.

In order to win, the software needs to play the same tune as the hardware components. There are several key ideas to follow to be able to get there. The environment is now correct for this to occur, with a few key steps.

Interconnect Fabric	RAID Storage	Deep Archive	Archive Server	File, Program, Data Servers
HIPPI	Megadrive	STK	Pentium PC	DEC Alpha 8200
Fibre Channel		IBM		SGI Origin 2000
Fast Ethernet	Kingston	4mm SCSI	Sun desktop	Pentium PC
...	...	...	...	...

Table 1. Component parts.

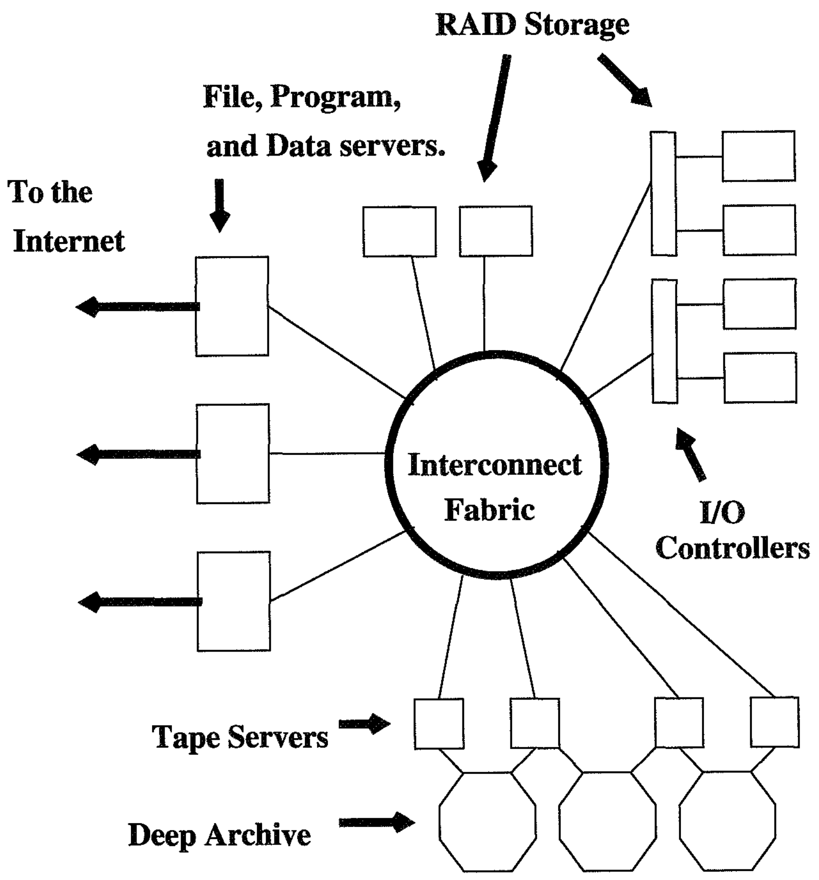


Figure 1: The hardware components of a mass storage system. After this, its a mere matter of software.

2.3. Key enabling mindset.

There are two key ideas that are needed to be able to do this. The first is to adopt a network point of view. The second is to recognize what that means to the software.

2.3.1. You must have a network centric point of view.

The basis model for the Internet is that components are designed to be able to plug into each other, virtually without limit. As shown in Figure 1, the various hardware components are designed to do this, and can easily be so connected. However, the business model of the site management needs to understand this as well. It is a common thought process to think of 'my storage machine's disk drives', or my storage machine's tape drives'. One must lose this mindset, and think of peripherals that happen to be hanging around the network. They are a common resource, all of them. No machine owns anything that has a long lifetime. Only by doing this can you avoid the deadly, time consuming and costly problem of large system transition.

2.3.2. Its the API, stupid.

Here we steal a phrase from our president some years ago, and mangle it slightly. This software interface drives what it is you can do. One can make all the comments one likes about simply plugging hardware components together, but if the software doesn't play together well, nothing actually works. So, once again, the overall storage system mindset has to go, and the Internet survivable and interoperable mindset needs to come into play.

Here is the key idea: You need to have your software laid out such that you are prepared to throw everything away, at a moments notice, within a given scope.

To reinforce it, I will say it again, slightly differently: In order to save your data, you must be prepared to abandon any part of it, at any time.

That scope can vary widely, depending on the business model of your facility. A small company, for example, might agonize over a $3000 tape drive. A large national facility may be prepared to roll a $300K network fabric out the door, because a vendor is no longer responsive. This kind of scoping is an essential part of the business management that needs to take place, in order to effective build the facility. Exactly where your facility exists financially is an essential part of the scoping effort.

3. How do we get there?

There are a host of implications to conducting such a development.

Open software interfaces have to be defined, where they do not exist. This process has in fact been going on for years, with the development of RAID, the definition of DMIG and DMAPI, the creation of the Openvault tape specification, IPI channel interfaces, SCSI and Fibre Channel disk interconnect, and the whole Internet standardization process. The time is now right to build a mass storage system that takes advantage of all of them.

4. Related and Future Work

There has been much related work in the IEEE Mass Storage Reference Model, and in particular in the development of HPSS (High Performance Storage System) [1]. Many of the ideas being developed are similar, such as network centric storage, and well defined APIs to store data. The key factor in this work is that the storage system software becomes completely unencumbered.

There is software glue to be created to be prove this concept. The NAS facility has been putting together a reference implementation for the past year, showing how it can be done.[2]

5. Conclusions

Really big storage systems are going to cost a lot less. Products will have to compete on capacity pricing with the cost of an IDE disk drive down at the local computer store. For disk, this is on the order of 10 - cents a megabyte, in mid 1997. Tape is of course 2 - 3 orders of magnitude cheaper.

Customers will be less nervous about deploying such systems, because they don't have the long term capital outlay and investment.

We believe that the most significant result of this type of work will be a great deal of growth in the marketplace.

Our basic design philosophy is mirroring that which we put into building the Internet, many years ago. Big storage systems will eventually be as easy to put together as that has now become, and because of that, people will do it. There is no question that this will take many years to get such ubiquity to occur, and the end results will certainly look nothing like what we first build. But it will happen.

References

In keeping with the network centric view, the major references used in this work are on the net. There are, no doubt many related papers as well. They are also shown at these locations. However, on the net is where the action is taking place., and where information for this paper was created.

[1] http://www.sdsc.edu/hpss/ - The web home for the High Performance Storage System.

[2] http://science.nas.nasa.gov/Groups/NAStore - The web home for storage at the NAS facility.

Page intentionally left blank

The Designs of RAID Systems with XOR-Capable Disks

Tai-Sheng Chang, Sangyup Shim†, and David H.C. Du
Department of Computer Science, University of Minnesota, Minneapolis, MN 55455
Email: {tchang,du}@cs.umn.edu Phone: +1 612 626-7522 Fax: +1 612 625-0572
† Department of Computer, Information, and Systems Engineering,
San Jose State University, San Jose CA 95192-0180
Email: sishim@email.sjsu.edu Phone: +1 408 924-4058

Abstract

Recently, the exclusive-or computation capability was added on high performance disks to provide an alternative way of implementing RAID (Redundant Arrays of Independent Disks). This alternative approach reduces the traffic on the storage channel and eliminates the XOR computation load on the host or RAID controllers. Therefore, it may be a better approach than the traditional RAID design to implement RAIDs in a mass storage arena. In this paper, we propose two new approaches with the XOR capability on disks to implement RAID systems. Simulation results are also provided.

1 Introduction

There have been a variety of RAID (Redundant Arrays of Independent Disks) systems proposed to improve I/O performance. It improves I/O performance by increasing concurrent accesses to disks. By using extra parity information, the RAID systems also provide fault tolerance when a disk fails. Data on a failed disk can be reconstructed with data and parity information from other disks. As a result, it improves data availability. Nonetheless, the update of data (e.g., write operations) requires extra tasks (exclusive-or computations) to update on the parity information.

The traditional RAID implementations rely on a centralized device to perform all the RAID management tasks and the exclusive-or (XOR) computations. Such a centralized device could be either a RAID controller for hardware RAID or the host CPU for software RAID systems. (For convenience, we call this traditional approach as the "host-based XOR" approach). Since the XOR computation is performed on a centralized device, the old data and old parity have to be transferred over the storage channel to a host or a RAID controller when data updates (*write* operation) or data reconstruction in a case of the disk failure. It results in not only more data traffic on the storage channel, but also higher buffer requirements to store all the temporary data. Also, the XOR computation could be a potential bottleneck with high bandwidth storage channel. In reality, the hardware based RAID controllers are expensive and can only connect to a relatively limited number of disks. On the other hand, the software RAIDs do not perform as well as the hardware based RAID controllers although they are much cheaper. It is because the XOR computation and other RAID management functions need to be carried out on the host and will compete with all the other applications on the system resources such as CPU, host memory, system bus, and etc.

As an alternative solution to the host-based XOR approach, the XOR computation can potentially be performed on disk drives. (We call this "disk-based XOR" approach.) Current Seagate Barracuda 9 FC-AL disk drives are indeed capable of supporting XOR SCSI commands proposed in [1]. With this disk-based XOR approach, XOR computation can be performed independently on disks. Also the old data and parity blocks do not have to be transferred to a central device to accomplish the XOR computation. It reduces half of the data traffic on the storage channel for *write* operations. Therefore, disk-based XOR seems to be a better solution for supporting a large scale storage systems compared to the traditional RAID implementations.

In this paper, we investigate the performance of the disk-based XOR approach. We proposed two new approaches with disk-based XOR. The details are described in Section 2. In Section 3, we will show the simulation results for these two approaches. In Section 4, we conclude out study and suggest some future works.

2 Disk-based XOR approaches

The new SCSI command proposed in [1] to accomplish disk-based XOR is called *XD Write Extend* (denoted by XDW-EXT). When a disk (target disk) executes an XDW-EXT, it will read the corresponding old data from the disk. It will then do the exclusive-or computation on this old data and new data (to be sent from host). It keeps the XOR result in the buffer and write the new data to the disk. At the same time, the target disk will send another command called *XP Write* (denoted by XPW) to the associate parity disk. When a parity disk executes such a command, it will read the old parity from the disk. When the parity disk is ready, the XOR result on the target disk will be sent to the parity disk. After the new parity is calculated, it will be written to the parity disk.

There are two implementation alternatives to process an XDW-EXT command. The first is to let the target disk wait for the completion of the operation on the parity disk before it can process the next command. That is, the target disk will not execute the next command in its command queue before the current XDW-EXT is completed. This approach is simpler and needs no other extra function. But it may result in lower disk utilization. The other approach is to allow the target disk to proceed the next command once the XOR result of the new and old data has been calculated and the new data has been written to the disk. This approach allows multiple commands pending on each disk. It would improve the disk utilization but require some mechanism to protect all the temporary data from overwritten.

Since the disks serve commands independently, a deadlock may also happen with such XDW-EXT operations with either approach. This is because it needs two disks to complete its tasks and may cause a deadlock with a circular waiting condition. One major challenge for implementing RAID with the disk-based XOR approaches is to prevent deadlock from happening while maintaining acceptable performance. We propose two different deadlock prevention approaches in this paper. The first approach is to avoid the deadlock condition by re-arranging the location of the parity blocks. For example, when all the parity blocks are stored on one disk (RAID-4), the circular-waiting condition will never happen. Our first approach is based on this observation and the idea of the RAID-5's parity block distribution among a group of disks to improve the performance. We partition the disks into several groups and store all the parity blocks of one group onto another group of disks. We call it "Grouping" approach. For example, if there are n groups, G_1, G_2, to G_n, we may store the parity blocks of group G_1 on the disks in group G_2. And store the parity blocks of group G_2 on the disks in group G_3 and so on. In general, we store the parity blocks of group G_{i-1} in group G_i where i is from 1 to $n-1$. Figure 1 shows an example of this approach with three disks in each group (two disks in Group 1). The advantage of this parity placement approach is that there is no need for change on disks. Neither the applications nor the disks require any change. The disadvantage of this approach is that, some disks in the last group G_n will have lower space utilization. To compensate this problem, we can use less number of disks in the last group since the last group is exclusively used for parity block in this approach. The load will be less since there is no additional load from data update on these disks.

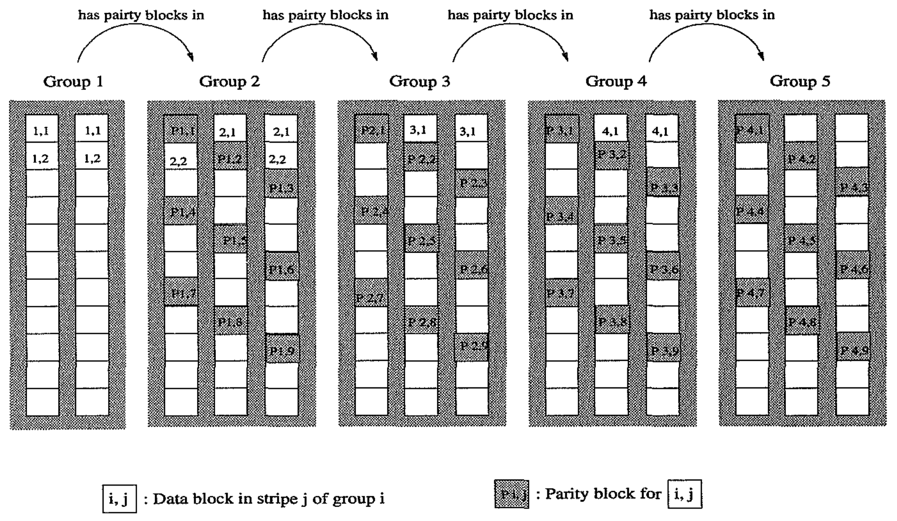


Figure 1: An example of Grouping Approach with 3 disks in each group

In the second approach (we call it the XPWF approach), we resolve deadlock by giving the XPW a higher priority and also ensure that the XPW will be completed once it is sent to the parity disk. The higher priority can be ensured by putting the XPW command at the head of the command queue on the parity disk. To ensure the XPW can be completed, each disk sends the associated XPW first when it starts an XDW-EXT command. After the XPW is sent to the parity disk successfully, the target disk starts processing the associate data update and XOR calculation (We call this portion of XDW-EXT command to be XDW). If an XPW from another disk arrives while a disk is trying to send out an XPW, then the disk with the arriving XPW should abort its current XDW-EXT command and stop the intent to send the associate XPW. The aborted XDW-EXT should be put back to the command queue. Figure 2 shows a control flow of this approach. Since the XPW has not been sent yet, the XDW-EXT has not been processed yet. And the overhead of aborting this command can be minimized since no data has been read and processed at the time of abortion. This XPWF approach works if the transmissions on the storage channel are serialized. That is, only one transmission is allowed at any time. Both SCSI and FC-AL belong to this category. In the case when multiple concurrent transmissions are allowed, such as in SSA (Serial Storage Architecture [3]), some modifications to provide acknowledgments from the parity disk are necessary. The advantage of this approach is that no modification on the applications is required. Also, we can use RAID-5 or whatever parity placement approaches. The disadvantage is that it requires some modification on the protocol of executing an XDW-EXT command. Also this approach is considered pessimistic because it does not necessarily result in a deadlock by sending out an XPW while there is an XPW from other disks waiting for service in its command queue.

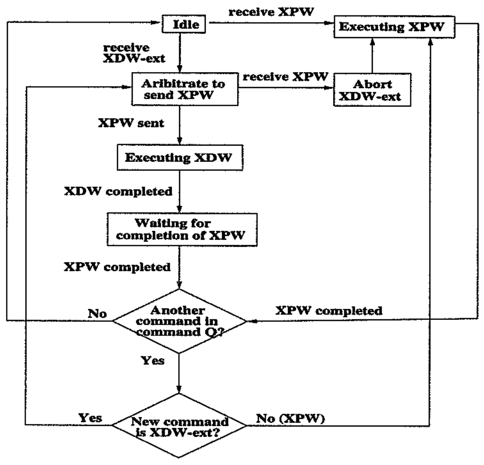
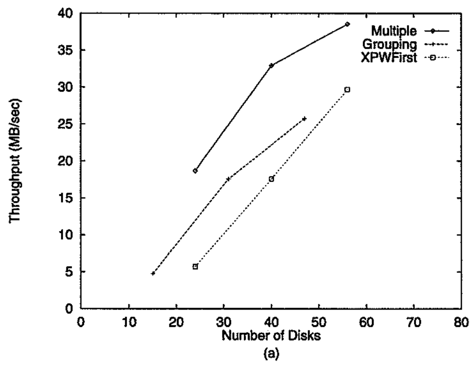


Figure 2: Control flow for XPWF Approach

3 Simulation Results

In this section, we will show the simulation results for the Grouping and XPWF approaches we proposed in previous section. The simulation model is based on the FC-AL model used in [3]. We keep a fixed number of outstanding commands on a host. The number is set to eight times the number of disks attached. That is, each disk has eight commands on average. It represents a case when the disks are highly loaded (each disk is either waiting for XPW to complete or reading/writing data).

The simulation results are shown in Figure 3. The disks were assumed to wait for the XPW's completion before it can process the next command. We also compare the results when multiple outstanding commands are allowed on each disk with the RAID-5 parity block placement. The results show that before the FC-AL link starts to be saturated (≤ 40 disks in Figure 3), both approaches achieved as much as two thirds of the throughput of the method allowing multiple concurrent commands on each disk. This is because in the single command case, the disks have to wait for its parity update before it can process the next command. Therefore, disk utilization is lower. When the number of XPW's are the same as that of XDW-EXT on a disk, the disk utilization will be less than 67%. This is because that each XDW-EXT needs to wait for the completion of its XPW. That is, the disk will be idle at least for the period of time executing an XPW. Assuming data and parity update need the same amount of time on average and the time is one unit, the disks will be idle for at least one time unit (waiting for XPW to complete) for every three units (each XDW-EXT needs 2 units and each XPW needs one unit). That is why the upper bound for the disk utilization is two thirds. When the delay on the loop becomes longer, the disks utilization will be lower.



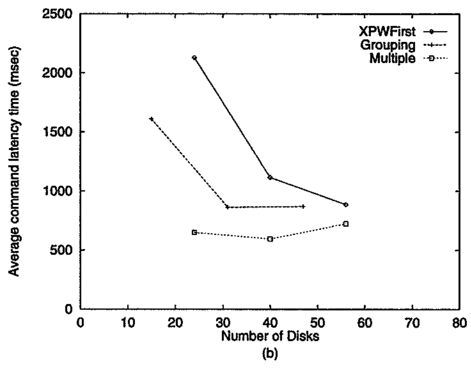


Figure 3: (a): The aggregate throughput comparison; (b): Access latency comparison

Figure 3(b) shows the average access latency for each request. The latency time include the command waiting time in the disk command queue, the disk service time and the data transfer time on the loop.

4 Conclusions

In this paper, we have proposed two different approaches to implement the Disk-based XOR. The simulation results showed that the Grouping approach is slightly better than the XPWF approach. Both approaches achieved about two thirds of the throughput of the multiple outstanding command approach.

In this paper, we have proposed two approaches to implement a RAID system by taking advantage of the XOR capability on disks. Further studies are necessary to investigate its performance with large *writes*, when rebuilding a failed disk, and its buffer requirement.

Acknowledgements

This work is partially supported by Seagate Technology, Inc.

References

[1] Gerry Houlder, Jay Elrod, and Mike Miller, *"XOR Commands on SCSI Disk Drives"*, X3T10/94-111r9.

[2] Sangyup Shim, Yuewei Wang, Jenwei Hsieh, Tai-Sheng Chang, and David H.C. Du, *"Efficient Implementation of RAID-5 Using Disk Based Read Modify Writes"* Technical Report, Department of Computer Science, University of Minnesota, 1996.

[3] David H.C. Du, Jenwei Hsieh, Tai-Sheng Chang, Yuewei Wang and Simon Shim, "Performance Study of Serial Storage Architecture (SSA) and Fibre Channel - Arbitrated Loop (FC-AL)", *to appear in IEEE Parallel and Distributed Technology*

Optimizing Configuration for Hierarchical Storage Based Continuous Media Server

Youjip Won
Server Architecture Lab
Intel Corp.
CO3-202 15220 NW Greenbrier Prakway
Beaverton, OR 97006
FAX: +1 503-677-6700
yjwon@co.intel.com

Jaideep Srivastava and Zhi-Li Zhang
Computer Science Dept
University of Minnesota
200 Union Street S.E #4-192
Minneapolis, MN 55455
FAX: +1 612-625-0572
{srivasta|zhzhang}@cs.umn.edu

1 Introduction

Recent advances in computing and communication technologies have made it technically feasible and economically viable to provide on-line access to a variety of information services over high speed networks. Particularly, convergence of various technological factors, namely in network access and in video coding and transmission, have recently brought a rapid growth of interest in on-line access to *multimedia* services. The problem of large-scale provision of services has several implications for the design of a storage server. Voluminous nature of a multimedia file is one of the primary reasons which makes the design of the server non trivial. With MPEG-2 compression technique a movie file of 110 min length requires more than 3 G bytes of storage space. Storing thousands of files of this size on a disk subsystem requires huge amounts of disk space. According to user surveys[7], file-access frequency is strictly biased in favor of a small number of popular titles, while the rest of the files are rarely accessed in commercial video rental. This characteristic of the user-access pattern enables the storage architecture designer to exploit the hierarchical storage structure in managing a large volume of information.

The advantage of using hierarchical storage architecture is to exploit the popularity or *hotness* of a file and assign space the appropriate storage hierarchy to each file, thereby maximizing the cost-performance ratio. The efficiency of hierarchical storage architecture is maximized when the capacity of each hierarchy is well balanced and the behavior of each storage hierarchy is well harmonized. Thus, in configuring hierarchical storage, these system parameters have to be carefully taken into account.

From the server's point of view, maximizing throughput is primary concern in various aspects of the design. Throughput can be thought as the number of requests which can be handled or *serviced* by the server per unit time. When user request needs to be serviced immediately, the number of requests which can be handled per unit time, *throughput* is inversely proportional to the probability that the incoming request is blocked. Thus, *Blocking probability* is an important concern in achieving a desired cost-effectiveness. In hierarchical storage architecture which consists of tertiary storage and secondary storage, the request blocking can be due to congestion in tertiary storage or due to congestion in secondary storage

In this work, the effort is concentrated on finding a minimum amount of storage resources for each hierarchy to achieve a given throughput. Storage capacity and bandwidth capacity of a unit storage device, e.g. *a disk drive*, *a tape drive* is fixed. *Blocking probability* is a metric for throughput.

There have been a number of studies about using hierarchical storage in continous media server[2, 4, 6, 5, 9]. Also, extending the notion of *hierarchy* to distributed system has been propsed[1, 10]. However, none of these literatures have dealt with the issues of *What capacity for each storage hierarchy is required to satisfy given performence metric.*

2 Problem Formulation

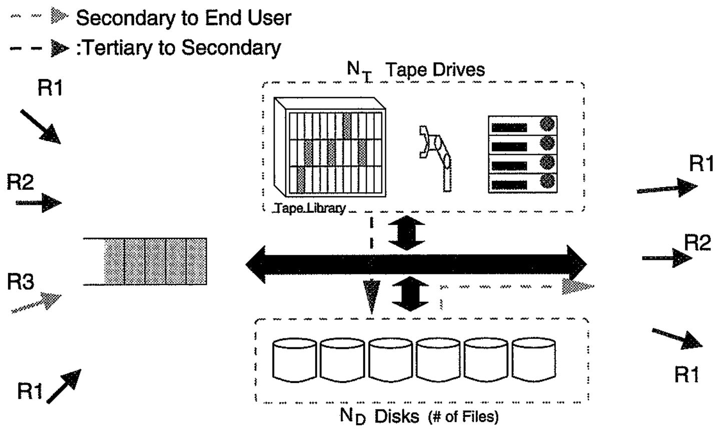


Figure 1: Service Mechanism

Fig. 1 illustrates the hierarchical storage architecture and its service mechanism. When requests arrive at the server, some of them cannot be serviced immediately due to limitation on system resources. All files reside permanently in the tertiary storage, which can be in the form of tape cartridges. A file is loaded from the tape library to the disk in an *on-demand* basis. When a user request arrives at the server, a mechanical device loads the respective tape cartridge into the tape drive unless it is already loaded on the disk or being loaded by the preceeding request. Then, the requested file is loaded from the tertiary storage into the secondary storage, which is a disk subsystem and then is sent to the user. It is also possible that incoming request observes that the requested file is either in the tape drive being loaded on the disk or loaded on the disk. In former situation, the request waits until the file is loaded on the disk and then starts to be serviced to user. In latter case, user is serviced immediately. In case all the tape drives are transferring files to the disk or disk subsystem is full, the server cannot accommodate new file, *Blocking situation.* Detailed queuing analysis of hierarchical storage behavior can be found in Won[8].

The objective of this work is to find minimum cost storage capacity to satisfy the given performance metric. The performance metric adopted in this work is *Blocking Probability*, probability that the incoming request is not serviced immeidiately. Given the performance specification of the individual disk subsystem and tape drive, e.g. *read/write bandwidth, head movement overhead*, we determine the amount of storage resources to preserves the blocking probability.

3 Approach

We develop an efficient methodology to find the optimal solution for configuration problem. In our hierarchical storage structure, server consists of two storage hierarchies - secondary and tertiary storage. Request is blocked when either of the hierarchies cannot accommodate the incoming request. Hierarchical storage system is modeled using the closed queuing network to compute the blocking probability. The customer in the closed queuing system corresponds to the files in the tape library. Thus, each customer repeats the cyclic transition among *Idle*, *On Tape Drive*, and *On Disk* status. The figure in the bottom of Fig. 2 illustrates the closed queuing network representation of the hierarchical storage system.

Given the blocking probability, e.g. δ in Fig. 2, it is of interest to determine the lower bound of the number of tape drives or the capacity of the disk subsystem to achieve the blocking probability. Fig. 2 illustrates the approach to find the configuration of storage

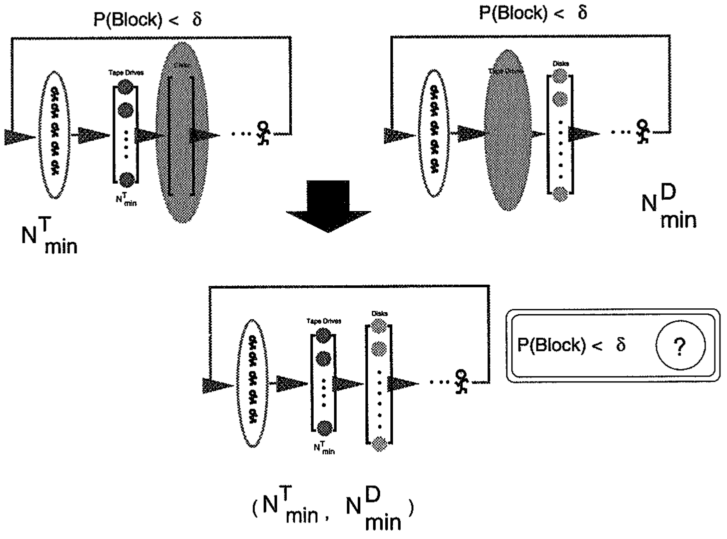


Figure 2: Partial Optimal Configuration and its Integration

hierarchy via partial optimal configuration and its integration. The configuration methodology is described in the following.

Given the user access pattern, aggregate user request arrival rate, data rate of the tape drive and data rate of the disk,

Step 1 Find minimum number of tape drives given that there is sufficient capacity secondary storage with blocking probability $\mathcal{P}$.

Step 2 Find minimum capacity of the disk subsystem given that there is sufficient tape drives with blocking probability $\mathcal{P}$.

Step 3 Build a hierarchical storage system with the amount of resources found in Step 1 and Step 2.

The amount of resources found in Step 1 and Step 2 is called *Partial Optimal Configuration.* Obtaining the partial optimal configuration is an import problem to make the entire framework useful. Cohen[3] found the famous result that in *Quasi Random Input, loss* *system*, the probability for the server state is determined by the *idle time* arrival rate and *expected service time*. Based on the Cohen's framework, we developed queuing system which precisely models the system where one storage hierarchy has sufficient capacity[8].

We use partial optimal configuration for disks and tape drives to obtain the optimal configuration. However, the server with N_T^{min} number of tape drives and N_D^{min} of disks may not preserve the blocking probability δ. This is because N_T^{min} and N_D^{min} is obtained given that there are sufficient amount of disks and sufficient amount of tape drives, respectively. Algorithm Opt_Config in table 1 is used to find the optimal storage hierarchy for a given probability δ.

Algorithm 1 Opt_Config

```
▷ Finding Optimal Configuration for given $\mathcal{P}$
$Opt\_Config(C_T, C_D, \Im, \mu_T, \mu_D, \mathcal{P})\{$
    Find partial optimal configuration $\Re_D$ with respect to blocking probability $\mathcal{P}$ ;
    Find partial optimal configuration $\Re_T$ with respect to blocking probability $\mathcal{P}$ ;
    Let starting configuration $\Re$ be $\langle N_T^{min}, N_D^{min}, \mu_T, \mu_D \rangle$ ;
    $N_T \leftarrow N_T^{min}$ ;
    $N_D \leftarrow N_D^{min}$ ;
    $C \leftarrow C_T N_T + C_D N_D$ ;
    while $(\mathcal{F}(N_T, N_D, \mu_T, \mu_D) > \mathcal{P})\{$
        ▷ Find the next closest point to $C_T N_T + C_D N_D = C$
        for $X \leftarrow N_D^{min}$ to $\lfloor \frac{C-(N_T-N_T^{min})C_T}{C_D} \rfloor + N_D$ ;
            $Y \leftarrow \lfloor \frac{C-C_D(X-N_D)}{C_T} \rfloor + N_T + 1$ ;
            ▷ Compute Distance between $(X, Y)$ and $N_T C_T + N_D C_D = C$
            $Dist \leftarrow \frac{|C_D X + C_T Y - C|}{\sqrt{C_T^2 + C_D^2}}$ ;
            if $(Dist < MinDist)$ {
                $MinDist \leftarrow Dist$ ;
                $X^{min} \leftarrow X$ ;
                $Y^{min} \leftarrow Y$ ;
            }
        }
        $N_D \leftarrow X^{min}$ ;
        $N_T \leftarrow Y^{min}$ ;
        $MinDist \leftarrow \infty$ ;
    }
    ▷ $(N_T, N_D)$ is optimal configuration
    return $(N_T, N_D)$ ;
}
```

Table 1: Algorithm for Opt_Config

4 Summary

Hierarchical storage system is being proposed to effectively utilize the popularity of each file providing cost effective solution for massive data storage system. Multimedia application is one of the context where data volume is much larger than text based application. In this work, we are interested in finding optimal configuration of storage hierarchy while maintaining a certain level of throughput, *blocking probability*. Given the speed of the tape drives and disk data rate, capacity of the hierarchical storage architecture is represented by the number of tape drives, N_T and the capacity of the secondary storage, N_D. Optimal configuration is a minimum number of N_T and N_D with a given data rate at each storage hierarchy and user access characteristics. In designing a hierarchical storage architecture, the user access profile needs to be carefully incorporated to avoid waste of the resources. In the method developed in this work, we first find the partial optimal configuration for each storage hierarchy. Then, from the integration of the partial optimal configuration, we obtain the optimal configuration. Queuing representation of the original hierarchical storage architecture is a two stage queuing network with blocking. By assuming the sufficient capacity in one storage hierarchy, the queuing model of the hierarchical storage system becomes single stage queuing model which is *finite capacity*, *finite customer*, *proper loss* queuing system. Cohen's result for *Generalized Engset Formula*[3] enables us to build a fine queuing model for partial configuration. The resulting analytical model for partial configuration becomes much simpler than original two stage queuing network to analyze. In this work, we propose an algorithm and its probabilistic framework to find a optimal configuration by integrating the partial optimal configuration.

References

[1] D. W. Brubeck and L. A. Rowe. Hierarchical storage management in a distributed VOD system. *IEEE Multimedia Magazine*, 3(3):37–47, Fall 1996.

[2] A. L. Chervenak. *Tertiary Storage: An Evaluation of New Applications*. PhD thesis, University of California at Berkeley, December 1994.

[3] H. W. Cohen. The Generalized Engset Formulae. *Philips Telecommunication Review*, 18(4):158–170, November 1957.

[4] Y. N. Doganata and A. N. Tantawi. A cost/performance study of video servers with hierarchical storage. In *Int'l Conf. on Multimedia Computing and Systems*, Piscataway, N.J., 1994. IEEE Press.

[5] Ghandeharizadeh, S. and Shahabi, Cyrus. On Multimedia Repositories, Personal Computers, and Hierarchical Storage. In *Conf. Proc. ACM Multimedia*, pages 407–416, 1994.

[6] W. T. M. Kienzle, D. Sitaram. Using a storage hierarchy in movie-on-demand servers. Technical report, IBM T.J. Watson Research Center, Hawthorne, NY, 1994.

[7] V. S. Magazine. Distributions of video viewing frequencies. Video Store Magazine, Dec. 13 1992.

[8] Y. Won. *Issues in Designing a Distributed Hierarchical Storage System For Continuous Media Service*. PhD thesis, Department of Computer Science, Univeristy of Minnesota, Aug 1997.

[9] Y. Won and J. Srivastava. Parametric Evaluation of Performance Behavior in Hierarchical Storage Architecture. *Submitted to International Journal of Performance Evaluation*, 1997.

[10] Y. Won and J. Srivastava. Strategic Replication of Video Files in a Distributed Environment. *International Journal of Multimedia Tools and Application*, To Appear.

Page intentionally left blank

The Mass Storage Testing Laboratory at GSFC

Ravi Venkataraman, Joel Williams, David Michaud, Heng Gu, Atri Kalluri, P C Hariharan
Systems Engineering and Security, Inc
7474 Greenway Center Dr, Suite 700
Greenbelt MD 20770-3523
{ravi, joelw, michaud, hengg, atri, hari)@ses-inc.com
Tel: +1-301-441-3694
Fax: +1-301-441-3697

Ben Kobler, Jeanne Behnke, Bernard Peavey
NASA Goddard Space Flight Center
Greenbelt MD 20771-1000
{ben.kobler, jeanne.behnke, bernie.peavey} @gsfc.nasa.gov
Tel: +1-301-614-{5231, 5326, 5279}
Fax: +1-301-614-5267

Abstract: Industry-wide benchmarks exist for measuring the performance of processors (SPECmarks), and of database systems (Transaction Processing Council). Despite storage having become the dominant item in computing and IT (Information Technology) budgets, no such common benchmark is available in the mass storage field. Vendors and consultants provide services and tools for capacity planning and sizing, but these do not account for the complete set of metrics needed in today's archives.

The availability of automated tape libraries, high-capacity RAID systems, and high-bandwidth interconnectivity between processor and peripherals has led to demands for services which traditional file systems cannot provide. File Storage and Management Systems (FSMS), which began to be marketed in the late 80's, have helped to some extent with large tape libraries, but their use has introduced additional parameters affecting performance. The aim of the Mass Storage Test Laboratory (MSTL) at Goddard Space Flight Center is to develop a test suite that includes not only a comprehensive check list to document a mass storage environment but also benchmark code. Benchmark code is being tested which will provide measurements for both baseline systems, i. e. applications interacting with peripherals through the operating system services, and for combinations involving an FSMS.

The benchmarks are written in C, and are easily portable. They are initially being aimed at the UNIX Open Systems world. Measurements are being made using a Sun Ultra 170 Sparc with 256MB memory running Solaris 2.5.1 with the following configuration:

- 4mm tape stacker on SCSI 2 Fast/Wide
- 4GB disk device on SCSI 2 Fast/Wide
- Sony Petaserve on Fast/Wide differential SCSI 2

Description of the Benchmark Code

The program exercises the I/O system of the machine on which it is running by processing (writing, reading, and copying) a specified number of files of specified file size and block size combinations. The code can exercise just the disk or it can exercise the disk and one tape drive. The files used are generated by the benchmark program during the test, and do not pre-exist the test.

A system load that is typical for the user may be evaluated by choosing files of different file and block size combinations. The tests are run on a stand-alone system to isolate user, network, and other overhead that may affect the measurement. The result thus obtained would be the ideal (maximum) throughput or the baseline throughput. By running the tests over the network the network overhead may be computed. If required external loads may be applied

The same benchmark could be run with the system subject to other load on CPU and I/O or run under Client/Server mode to measure the response times under typical user operating conditions.

In order to exercise a Hierarchical Storage Manager (HSM), it is only necessary to write a number of files to the fill the (staging) disk that forces HSM to migrate the files out to tape. Any attempt to read the files back will force the HSM to migrate (in) the files to the staging disk. HSM migration parameters are measured by special tests that force migration of files from the staging disks to the tape using the vendor provided APIs.

The disk test is performed in the following way:

(1) Write a file to disk
(2) Read that file from disk
(3) Copy that file from disk to disk
(4) Read the copy

The tape test is performed in the following way:

(1) Write a file to tape
(2) Read the file from tape
(3) Copy the file from tape to disk
(4) Copy the file from disk to tape
(5) Read the copy from tape

Whenever a file is written, there are three words at the end of the file which are used to mark the file. These words contain a checksum generated when the file is first written, a file marker, and the length of the file. Whenever a file is read, except in the case of the copy operation, these three words are checked to make sure that the file has been properly identified and read.

When benchmark tests were first run, the effect of the file system memory cache was immediately evident. Small files remained in memory, and hence were "read" back very quickly. In order to minimize this effect, the benchmark was modified to do a pre-test preparation. This simply reads and writes a sufficient number of files to fill up the file system memory cache.

The benchmark may be run in one of two modes: rotational or sequential. In the first mode, a file is created and written to the disk. Then that file is read back in and checked. Next the file is copied, and afterwards the copy read and checked. This is repeated for the specified number of files, and then for each block size and file size combination specified. In sequential mode, however, all of the writes are done in succession for the specified number of files. Then they are all read back in and checked, next copied, and finally the copies read and checked. As expected, the effects of the file system cache are more evident in rotational mode than in sequential mode, since in rotational mode there is no processing intervening between the write (or copy) and the subsequent read of a given file.

As the program executes, it collects statistics and writes them into a comma-separated-variable file that may be easily read by a spreadsheet program for further analysis.

Some Test Results

A fairly lengthy series of tests are required to properly exercise the HSM. Such a series of tests produce much more data than can be presented in this paper. For this reason, only a small, but representative, sample will be given here. The tests exercised the Petaserve system and the disk drive system only, the 4mm tape system was not used in these tests.

Figures 1-8 show results for files of size 7,962,624 bytes, using a block size of 66,536 bytes. The first four figures show the write, first read, copy, and second read for the test in Rotational mode; the next four have the corresponding results for Sequential mode. Both of these test runs were accomplished by running the disk test on a sufficient number of files to cause the HSM to have to roll files out to tape in order to make room for new files that were being written to the disk. In Sequential, but not Rotational mode, files also had to be brought in from tape.

These tests are not intended to represent any workload, and in fact are somewhat unrealistic in that they exercise the HSM more than would be likely or desirable in a production system. The limiting factor in throughput in each of the tests is the disk drive, which can run at a maximum of around 6MB/sec, including the overhead of the benchmark program. This is clear at the beginning of the Rotational mode test, for the first 85 or so operations where all that is being measured is the disk drive performance, since the HSM and the tape system have not yet come into play. Note that for the copy operation, throughput is calculated by doubling the file size, since there is both a read and a write operation to accomplish the copy. Since there was only one disk partition used in the test, there is some disk contention in the copy operation.

On all of the charts, the short downward spikes on write, read, or copy operations are attributed to the file system cache. Periodically it fills up and some delay occurs while it is cleared. The long downward spikes are attributed to the action of the HSM, when it becomes necessary to roll files out to tape to make space for additional files being written to the disk. Note that these spikes occur at regular intervals.

In Rotational mode, a file is written, then read, then copied to a new location, and then the copy read in again. This is repeated for the specified number of files. The HSM comes into play whenever a write or a copy causes the disk usage to reach the high water mark. Hence there are downward spikes either on the write or the copy. The reads, occurring immediately after the write or copy, are fairly fast, and may even come directly from the file system cache. The HSM does not come into play directly, although it may still be making more space on the disk at that time, and thus cause overall throughput to slow because of contention for the disk.

In Sequential mode, all of the files are first written. Hence the downward spikes occur whenever the disk usage reaches the high watermark. By the time that a file has to be read, it is certain that it has been rolled out to tape. Hence the HSM is active in moving the file back to disk on every read. In addition, the HSM may have to be simultaneously rolling files out to tape in order to make room for the files it needs to bring in from tape. These same comments hold for the copy operation and the following read. This explains the significantly lower throughput in Sequential mode as opposed to Rotational mode.

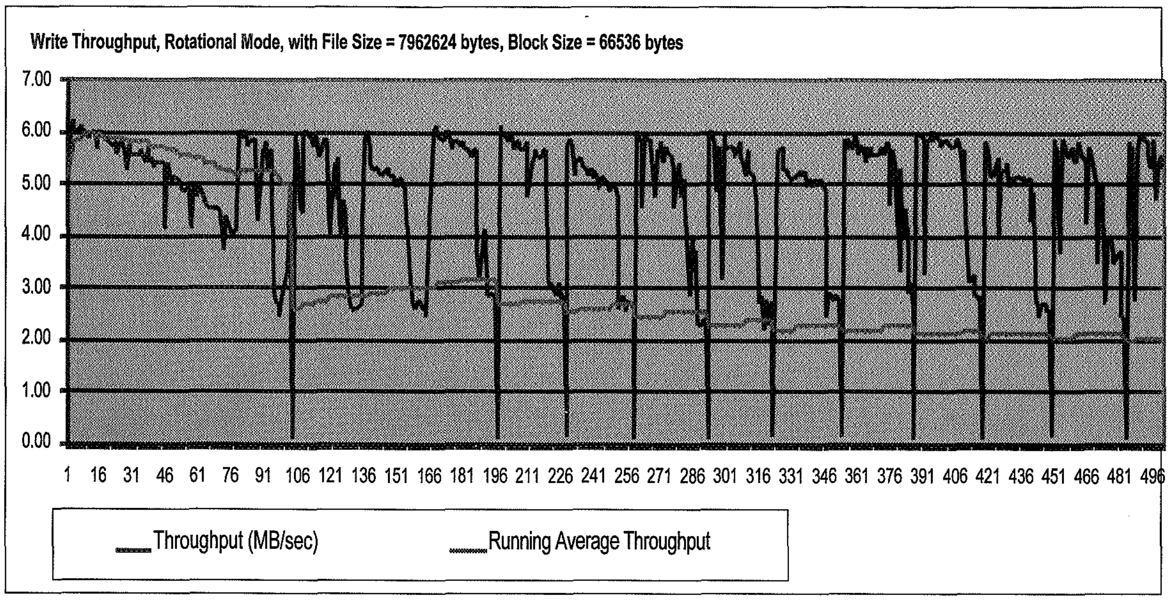


Figure 1

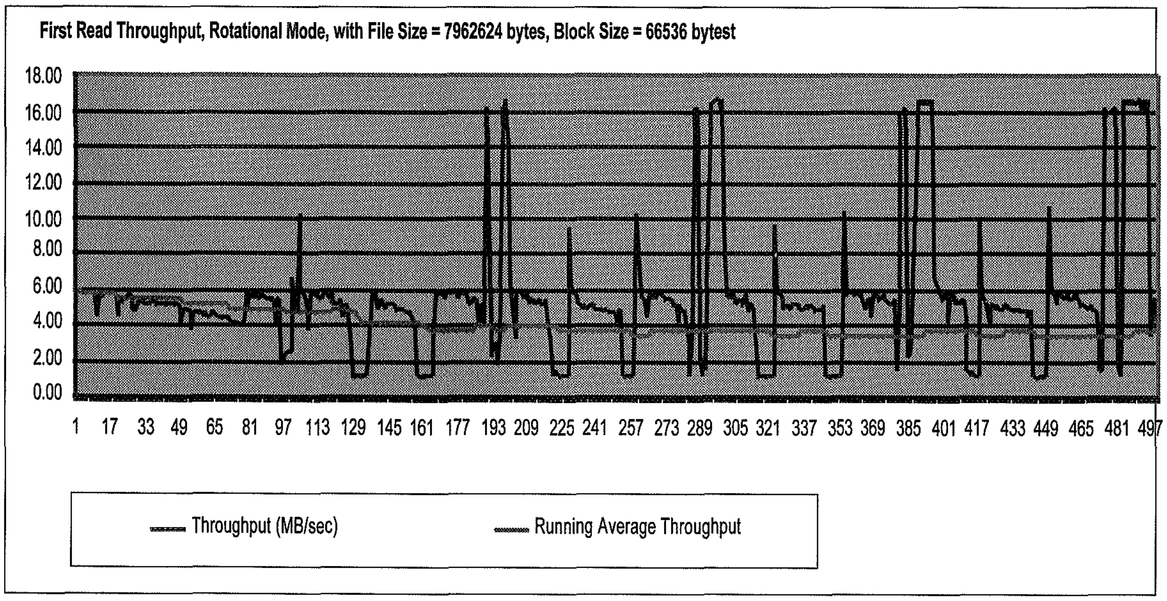


Figure 2

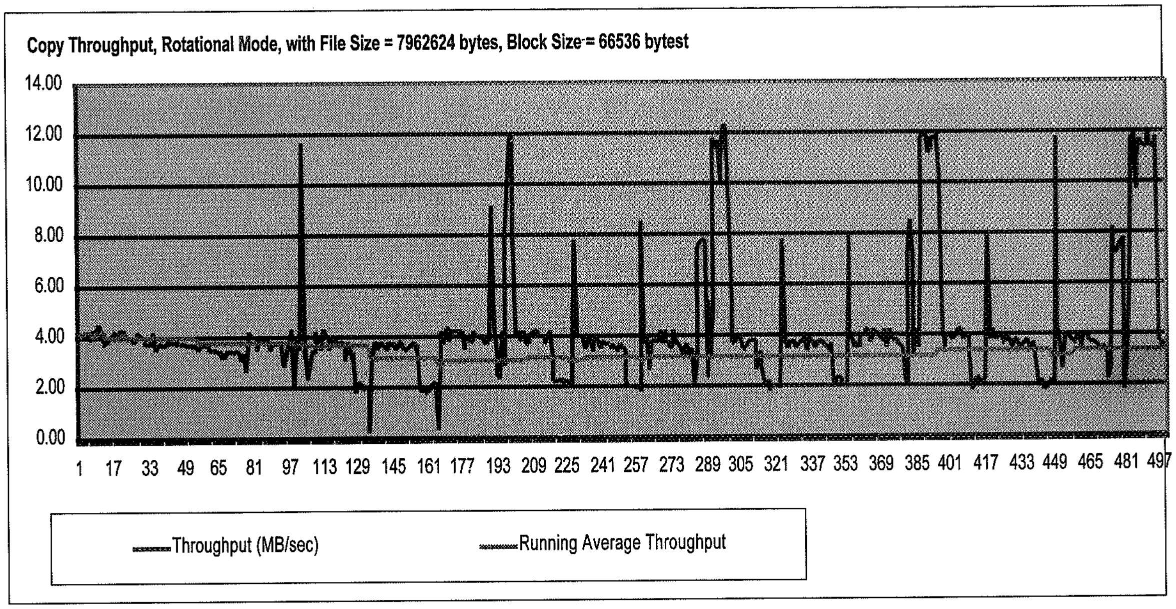


Figure 3

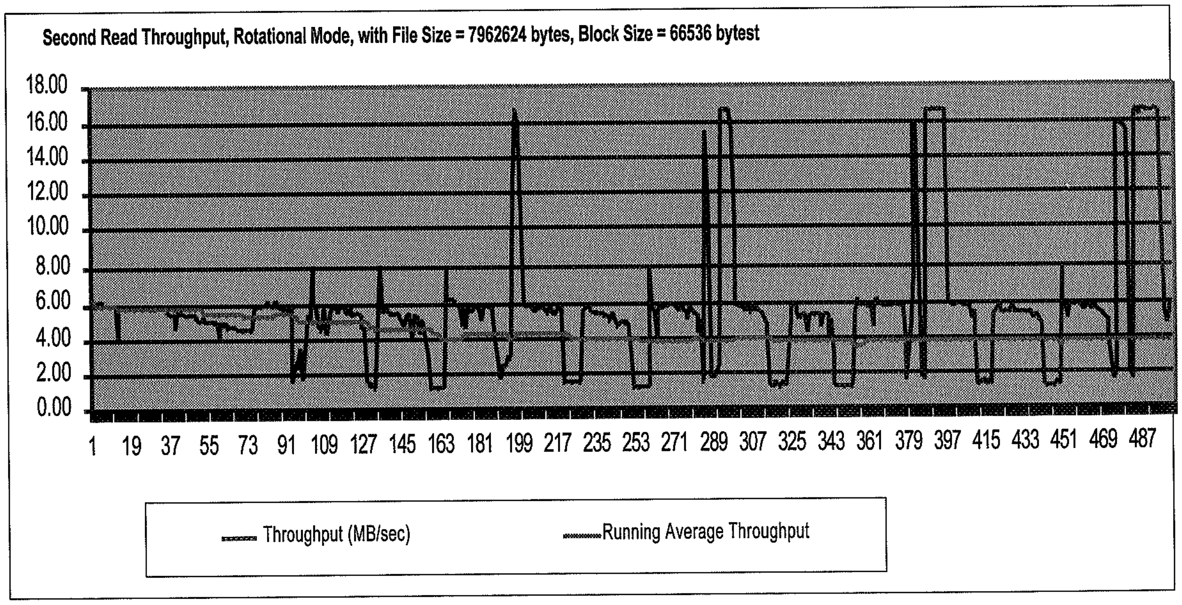


Figure 4

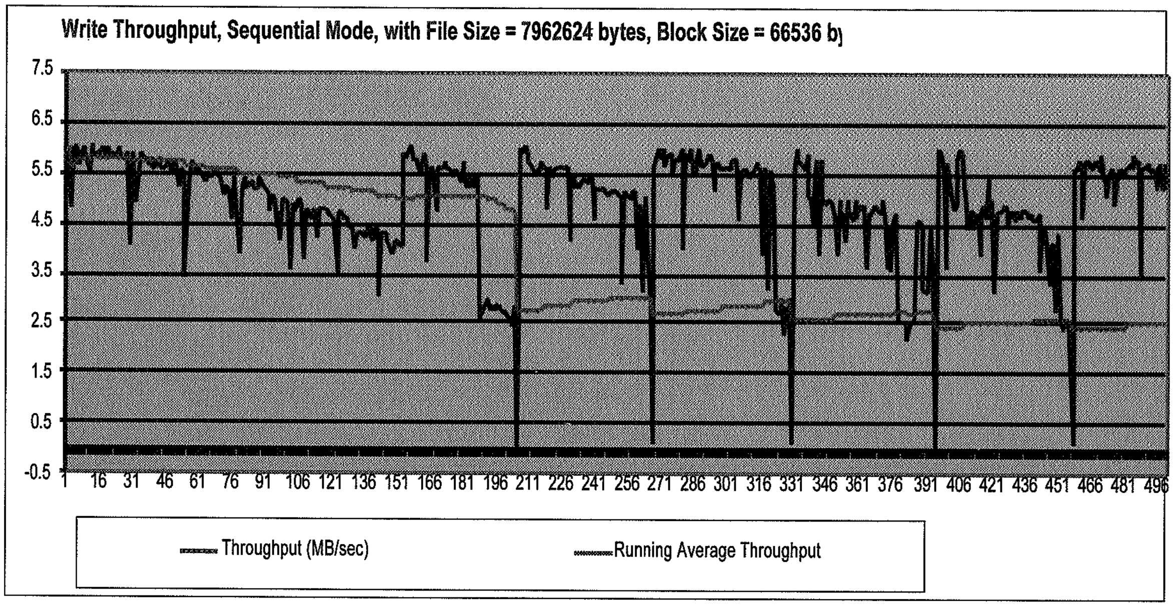


Figure 5

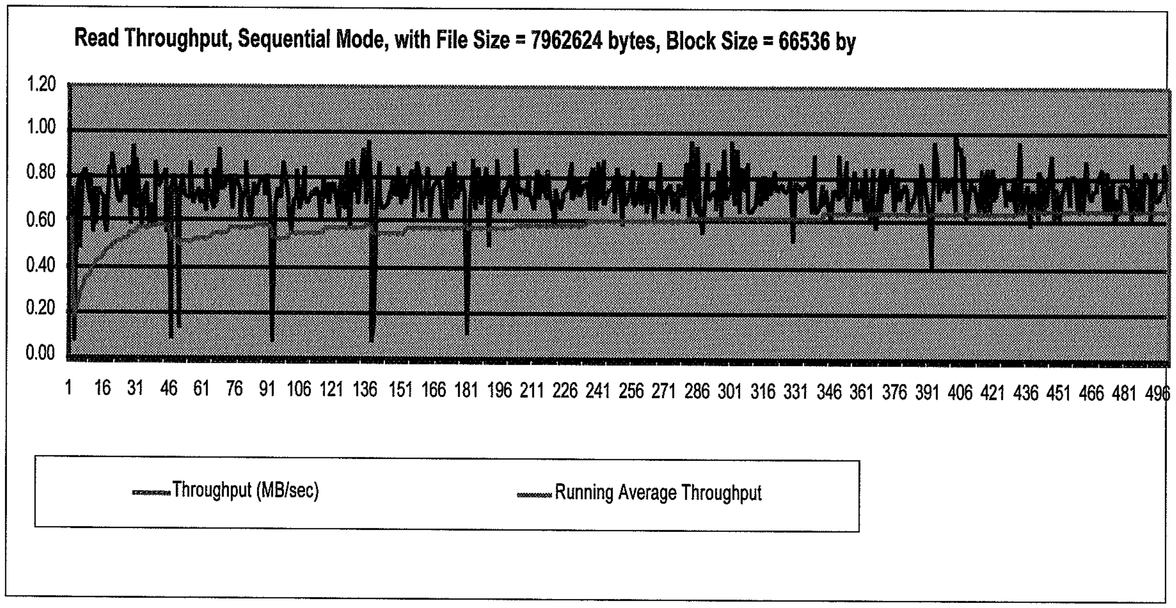


Figure 6

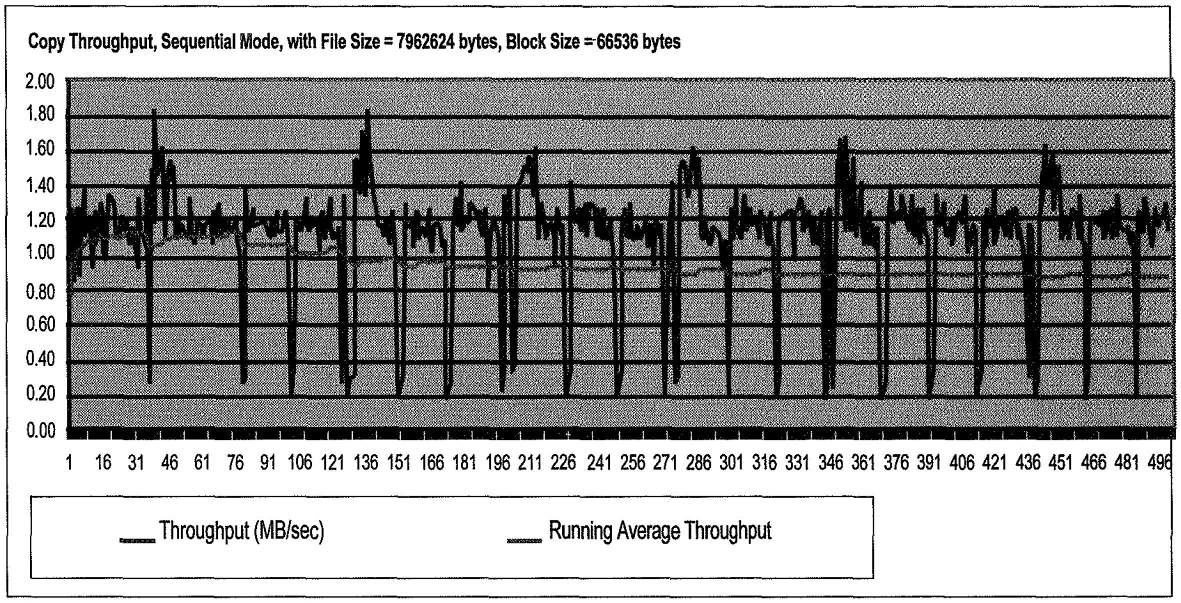


Figure 7

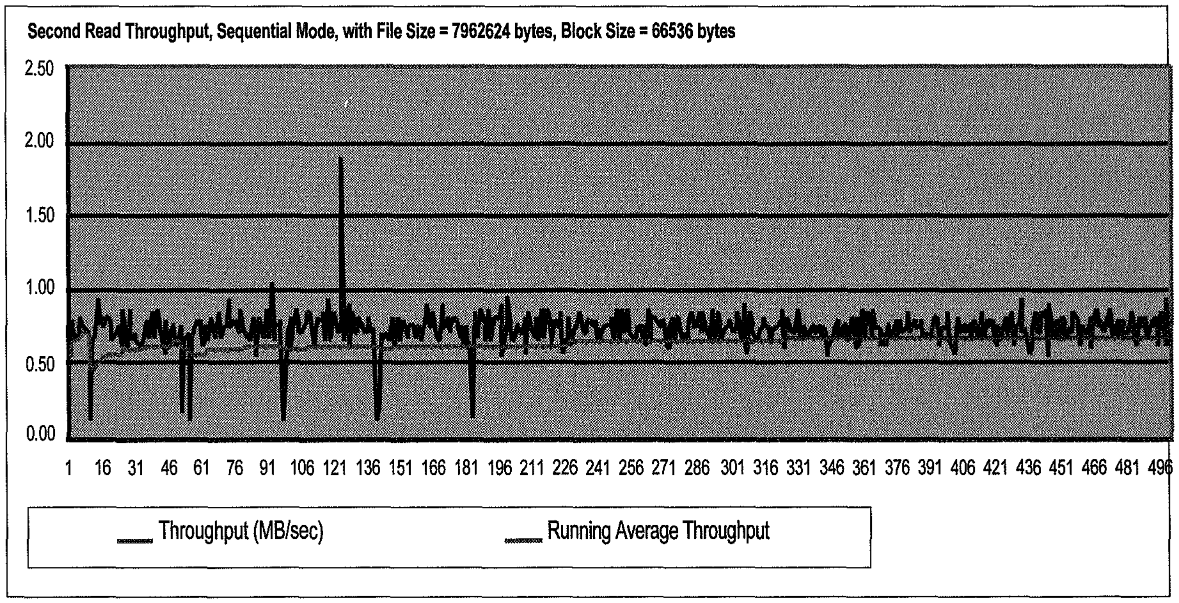


Figure 8

The HSM bases its operation on a high and a low watermark. When the disk drive is filled to the high watermark (90% of capacity in this test run) the HSM rolls files out to tape until the drive is only filled to the low watermark (70% of capacity in this test run). During the test run, it was observed that when the disk drive reached the high watermark, the HSM could not roll files out fast enough to keep the disk from filling up completely. *The* *watermarks, however, are completely user-configurable, though these tests were run with them only at the values mentioned.* The tape system would probably be better used if the high watermark were set to a lower number.

Future Directions

We expect to run additional tests in the future, using different hardware and software. These tests will allow us to determine the effect of the HSM software and the parameters used to configure it, the disk drive system, and the computing platform on system performance. We will also be testing the code on other environments to test for portability. In addition, we will begin including code for testing client-server connections to the mass storage system.

Acknowledgements

We thank Sony Corporation for the loan of the library unit and Petaserve; in particular, we are grateful to Frank Jones, Stu Reeder and Madhu Reddy for sharing their expertise, and for being available to answer our numerous questions.

EuroStore[1] – a European Mass Storage Management Project

Martin Gasthuber
Deutsches Elektronen Synchrotron (DESY)
Notkestrasse 85 22607 Hamburg/Germany
FMartin.Gasthuber@desy.de
tel +49-40-8998-2564
fax +49-40-8998-4429

Abstract

A European consortium formed by science and industrial partners have started the EuroStore project to develop and market a Hierarchical Storage Management System (HSM) together with a high performance parallel file system (PFS). The EuroStore project aims to design and develop a high performance file store. It will combine the features of a Hierarchical Storage Manager (HSM) with the performance of a parallel file system to provide a system capable of meeting the requirements of the most demanding applications in industry, commerce, and science. The few existing HSM and parallel file system products do not fully address all the needs of the user community, and in some cases represent a major investment that users are not willing to undertake. This paper will outline the current state of the design and implementation ideas.

1 Introduction

The complete EuroStore project is divided into two major components – the HSM at the bottom and the parallel file system as its client on top. Figure 1 shows the basic overview of the complete system.

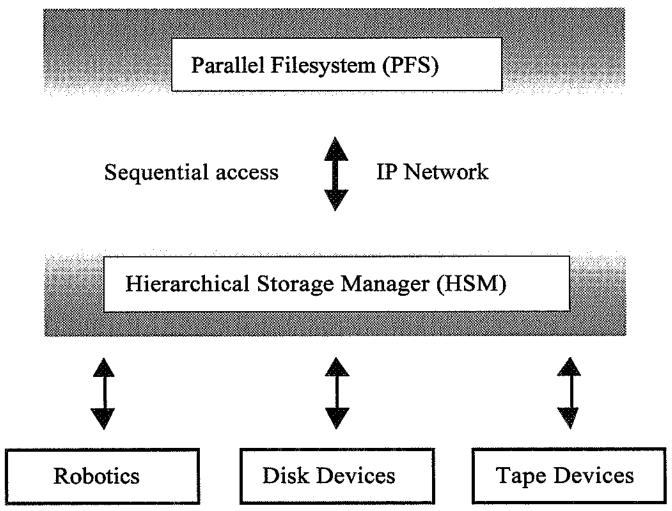


Figure 1: Basic component overview

This simple separation into two major building blocks leads to the separation of development tasks where the parallel file system and the HSM will be developed by two different teams with minimal dependencies on each other. The complete EuroStore collaboration includes the following companies and science institutes.

- QSW (Industry / UK)
- CERN (Science / French)
- DESY (Science / Germany)
- HCSA (Industry / Greece)
- STAR (Industry / Italy)
- HNMS (Industry / Greece)
- AMC (Industry / Greece)

DESY and QSW will be the development sites for the HSM and parallel file system respectively. The proposed project was approved by the European Commission and will start on March 1, 1998, with a duration of 2 years for the first stage.

2 The two components

The following section describes the proposed development and/or extensions to the base products that will be done during the EuroStore project. Section 2.1 will focus on the parallel file system called PFS and section 2.2 focus on the Hierarchical Storage Management System called HSM.

2.1 Parallel Filesystem - PFS

The parallel file system PFS was initially introduced in the MEIKO CS2 (Computing Surface) massive parallel computer system a few years ago, and allows versatile configurations of hardware and software components among all nodes on the system. This leads to a high performance and high capacity capable implementation.

Due to its implementation (on top of standard VFS layer in the UNIX kernel) any legacy application can use PFS and its additional features like file migration, without any changes. The current implementation runs in the Sun Solaris environment and is compliant with the Sparc ABI.

Throughout the EuroStore project PFS will be enhanced and extended in the following areas:

- HSM integration – file migration capabilities
- Performance improvements – exploiting CS3 hardware capabilities
- RAID-5 capabilities in software
- Easy administration tools and interfaces
- Portability to other OS
- Globally accessible character devices (like tapes) from all nodes in a system and the ability to define logical tape devices mapped to any number of (cheap) physical tape devices connected to any node.

The HSM integration will be the main extension and will be based on the HSM API designed in cooperation with the HSM development team. The approach here is to make use of the already existing stub-file mechanism inside PFS and add a user level migration daemon plus some administration tools. To achieve high performance file access through the whole chain of components (process – PFS – network – HSM – device) PFS will make use of the already striped layout of files and will access the HSM from all nodes with striped data from those files in parallel. This design makes complex software, dealing with striped tapes etc., obsolete.

2.2 Hierarchical Storage Management System - HSM

Unlike the PFS component, the HSM will be built from scratch exploiting the experience of DESY and other partners during the past years. The overall design goal is to make the software simple and smart. We do not rely on any third party component (except a commercially available database for persistent metadata) like DCE, nor did we plan to introduce new complexities by supporting parallel tape I/O or similar features. The HSM should work in small and large environments, supporting a large number of concurrent data streams with the maximum bandwidth the slowest component in that stream can sustain. The following simplistic list gives an overview of the proposed characteristics:

- Support of device hierarchies of any device type (random or sequential)
- Based on the IEEE Reference Model V5
- Network-centered architecture (*i.e.* unlimited number of 'mover' hosts) in a heterogeneous computing environment
- Full security for all control connection through Idea and other encryption mechanism (no DCE installation required)
- Easy integration/adaptation of new robotic and/or storage devices through simple and open API
- Easy portability (hopefully no work)
- Implementation in JAVA
- Administration/Monitoring/Controlling through HTTP/Java and SNMP. Any WWW browser can be used to connect to the HSM in a nice and secure environment, and already installed network management tools using SNMP can be used to monitor any running HSM components.
- Initially support only for sequential file access through TCP streams
- Support for automated (robot) and manual libraries
- Use of IP protocols for control and data communication
- Logical (and also physical) separation of control and data communications

Besides the listed characteristics we plan to add features in a second stage of the project based on the experience end-users gain while using the initial version. Due to its loosely coupled architecture both components can easily be configured to run without the other complementary component (*e.g.* through the open HSM API).

3 Future Work

Though the official project start is March 1, 1998, we have already made numerous design decisions reflecting basic ideas and implementation strategies. For increase of availability and performance it is planned to add another component to the HSM for caching purposes only. By the time of the conference we will show more detailed and fixed design and implementation strategies.

4 Conclusions

EuroStore will have all required features needed in a high performance and data intensive application in a small, scaleable, manageable size and complexity. By requiring minimal features from the host computer system, the use of modern object oriented languages and extensive use of standards, the system will enforce a smooth deployment in a heterogeneous computing environment.

---

[1] Funded by the European Commission ESPRIT PROJECT 26317

Transparent Continuous Access to Infinite Data: A New Vision for Enterprise Data Management

Don Banks and Christina Mercier
Auspex Systems
5200 Great America Parkway
Santa Clara, CA 95054
dbanks@auspex.com and cmercier@auspex.com
+1 408-986-2000
FAX: +1 408-986-2020

Abstract: This paper presents the "Transparent Continuous Access to Infinite Data" (TCAID) architecture, developed at Auspex Systems, which addresses the enterprise data management challenge. The model presented here separates policy from mechanism and treats the solution space as a bounded area described by simple data management parameters which encompasses a wide spectrum of possible data management solutions. Using this model existing data management point products become a series of customer-specified policies implemented by pre-packaged mechanisms that operate on customer-specified target data.

1. Introduction

Organizations are centralizing control of their distributed data in order to share and exploit the content. The Transparent Continuous Access to Infinite Data (TCAID) architecture addresses the centralized control of globally distributed data by supporting a set of data management solutions capable of implementing many policies over the data irrespective of its location. The architecture makes the set of services offered by the "cloud" of data management platforms function as a single cohesive solution.

The TCAID model provides access to client data so that even in the event of failures and/or loss of the primary copy of the data, client programs are unaware of the failure and continue to access it as if the primary is still the source of the data and no failure has occurred. No logical limits exist on how large the total collection of data can grow for either individual files or entire file systems including copies of those files and file systems.

2. The Problem with Current Data Management Solutions

Data volumes supporting the enterprise, and even single applications, have expanded to tens of terabytes of data. Data Management products such as HSM, disaster recovery, archive, and high availability, are showing signs of strain managing such enormous data sets and are reaching their breaking point. These products are often limited by file size, internal database performance, recovery mechanisms, etc. The backup window has disappeared as file systems have grown to tens of terabytes and terabyte file transfers for backup and migration are saturating network bandwidth. Data management products, dependent upon the availability of tape drives or other media devices to deliver their service, time-out or "hang" critical client requests. Products that have worked well for managing data in the enterprise have suddenly slowed down, or worse, stopped working all together as thresholds, queues, and/or resources have been exceeded.

3. The Vision: A New Paradigm for Data Management

Transparent Continuous Access to Infinite Data (TCAID) is an architecture which addresses the limitations encountered with current Enterprise Data Management products. TCAID separates policy from mechanism and presents a model that treats the solution space as a bounded area described by simple data management parameters, encompassing a wide spectrum of possible data management solutions. Individual solutions, such as backup, migration, as well as new ways of managing data not available in current products, can be described by applying customer-specified policies to the data to be managed. TCAID technology provides high-speed replication and a simple, manageable way to describe policy over a set of data. The TCAID architecture allows growth to hundreds of terabytes. It provides an integrated solution which is only constrained by the amount and type of data storage hardware available.

4. The TCAID Architecture

The TCAID architecture is an "embedded" model in the sense that its internal architecture is completely transparent to clients during operation. For the network filesystem client, the data is available over a logical connection to a server. Where and how the data is physically stored and accessed is not visible or of concern to the client or the client network filesystem protocol. TCAID is an architecture directed at the fileserver environment, not a local filesystem implementation. This is an underlying assumption that permeates the design. The fact that all requests (whether NFS, SMB or some other supported network file system protocol) come from the network rather than through a local API or local application request, supports and constrains the TCAID design.

There are two ways the model must be viewed. The first is the data-centered view (Figure 1), which illustrates the concepts by which data is managed to achieve policy-driven data management services. The second is the client-centered view (Figure 2), which represents the infrastructure and logical presentation of the TCAID-enabled file servers to clients.

4.1 The Data-Centered View

The TCAID model encompasses data management and high availability products by describing data replication as a three dimensional model. The first dimension (shown on the x-axis in Figure 1) is the logical distance from the source copy that the replicated data resides (local disk or tape versus disk or tape on another server). The second dimension (shown on the y-axis) is the time allowed to elapse between the time the source copy is modified and the time that the subsequent target copy is created (mirroring delay). The third dimension (shown on the z-axis) represents some number of coherent replicated copies in time that are to be managed and made available as historical views of the data (e.g., today's version, yesterday's version, etc.). While the x and y axes represent in a logical fashion the physical topology over which the data is spread, the z axis allows time to progress backwards (e.g., last Tuesday's data, last month's data or last year's data) to represent the illusion of infinite versions of data kept in the system managed "archives".

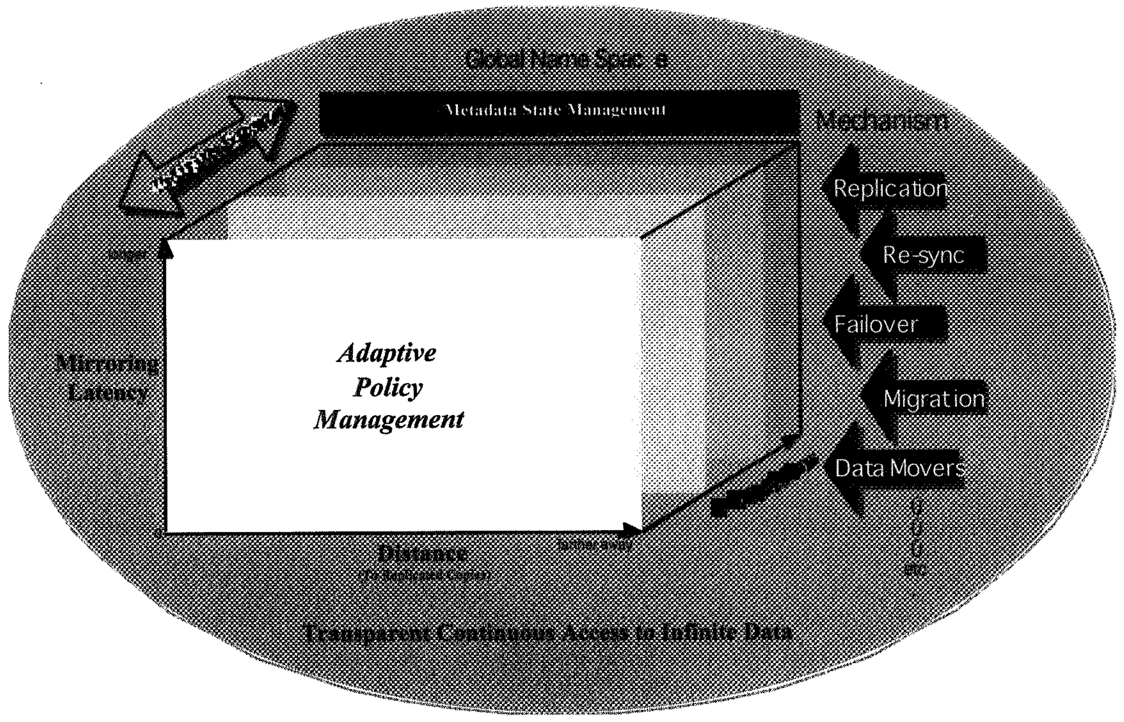


Figure 1: The Data-Centered View

The intent here is to provide sufficiently rich and powerful data management mechanisms such that a custom solution can be assembled and executed on demand based on the policy or policies specified by the administrator for particular data sets. By applying a rich common set of mechanisms, any policy can be implemented. This is the classic separation of policy from mechanism to allow a dynamic, adaptive implementation.

4.2 The Client-Centered View

The second view of the TCAID model is the client view of the TCAID-enabled server (Figure 2). A key element of this architecture is that it distributes data in a controlled and managed fashion that is transparent to the network client. In essence, the data-centered model is overlaid on this infrastructure in order to provide the illusion of consolidated data while at the same time providing the reliability and convenience of automatic distributed replications.

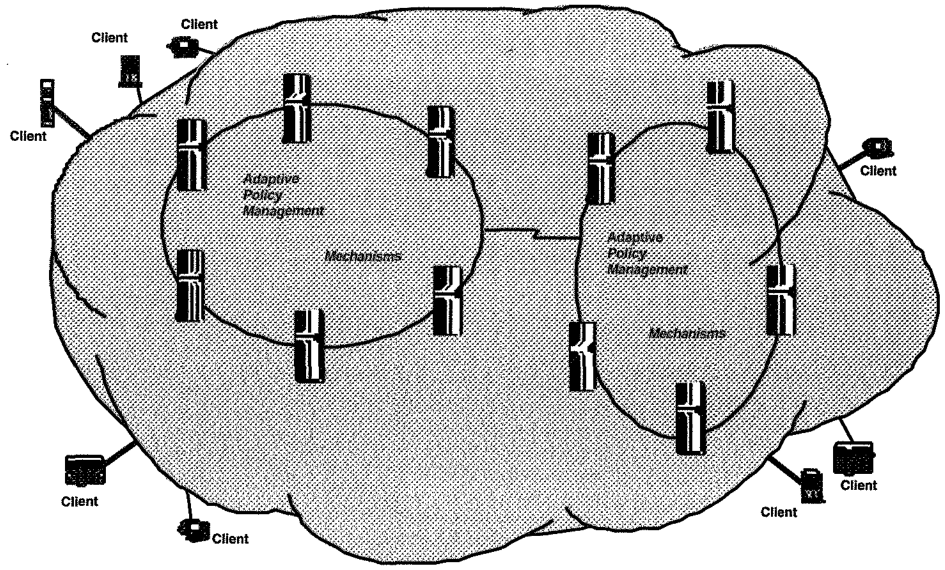


Figure 2: The Client-Centered View

Network clients accessing files through standard network protocols have a logical (rather than a physical) view of a "cloud" of TCAID-enabled servers; the client views a single data server providing a single connection point. Physically this cloud may be composed of any number of file servers capable of taking over for a failed server or filesystem and making the replicated data available in a manner transparent to the client. By specifying a policy that creates one or more replications of the data within the cloud, administrators guarantee that the physical failure of a server providing the primary copy of the data causes little or no effect to the client.

One of the important attributes of this architecture is that it allows the construction of a distributed filesystem within the cloud while presenting a simple single server view to the client. The client does not have to run any special software in order to gain the advantages of the distributed implementation. Location-transparent file naming along with a global name space provides the flexibility required to manage the distributed data.

5. TCAID Technology

The separation of policy and mechanisms in the TCAID architecture allows the system to provide an implementation that can dynamically build a solution to carry out one or more policies specified by the administrator. The mechanisms are transparent and the interface by which the policy is specified is simple to understand and administer. The mechanisms represent the technology while the policies represent the presentation of that technology in an implementation-neutral fashion to the system administrator who must describe how the data is to be managed.

Policy is the means used to define the ways in which data items (e.g., a file, a directory of files or a file system) are managed with respect to creating and maintaining replications. Policy management is adaptive in both a static and dynamic way. Static adaptation occurs as the system administrator defines the policies through the provided GUI. This is termed static because it occurs once at definition time. The system dynamically chooses "intelligent" defaults for not yet specified parameters and defines/chooses the mechanism(s) required to implement the policy (or policies) specified by the administrator in order to build a custom data management application. Conflicts are recognized at the time they occur and conflicting selections are prevented. Dynamic adaptation occurs as the implementation of policy (i.e., the mechanism) adjusts to changing conditions and/or configuration so that the system can continue to carry out the desired policy in the most efficient and effective way. For example, if asynchronous replication is specified with the copy being no more than 15 minutes out of date but the network bandwidth changes (either up or down), the replication mechanism adjusts by changing the replication window. When policies cannot be carried out as specified, the system provides a notification alert and logs the untoward event(s).

Mechanisms implement policy. They are comprised of independent, modular "building blocks" that provide capabilities required to fully address the policies. Mechanisms work together as a cohesive unit by sharing and operating on common metadata, either locally or distributed across the network. Mechanisms represent both the platform-independent and platform-dependent parts of the system. Depending on the base system infrastructure, strong interfaces (i.e., APIs and/or protocols) allow these mechanisms to be highly portable by enabling alternative shallow-stack or deep-stack implementations of function.

6. Status

Work is under way at Auspex Systems to implement solutions using the TCAID architecture. Auspex is currently early in the prototyping and development phase and expects to have concrete findings and conclusions with respect to the implementation within the next six months. At this point, the TCAID architecture has provided Auspex with a solid foundation from which to build solutions to the data management problem for the consolidated network file server environment.

Page intentionally left blank

A Scalability Model for ECS's Data Server

Daniel A. Menascé
Dept. of Computer Science
George Mason Univ., MS 4A5
Fairfax, VA 22030
menasce@cne.gmu.edu
tel: +1 703 993 1537
fax: +1 703 993 1710

Mukesh Singhal
Dept. of Computer and Info. Sc.
The Ohio State Univ.
Columbus, OH 43210
singhal@cis.ohio-state.edu
tel: +1 614 292 5839
fax: +1 641 292 2911

Abstract

This paper presents a model for the scalability analysis of the Data Server subsystem of the EOSDIS Core System (ECS). The goal of the model is to analyze to determine if the planned architecture of the Data Server will support an increase in the workload with the possible components upgrade. We provide a summary of the ECS's Data Server architecture and of a high level description of the Ingest and Retrieval operations in the ECS's Data Server. This description forms the basis for the development of our scalability model. Then we present details of the scalability model and the methodology used to solve it. We describe the structure of the scalability model, input parameters, expressions for computing parameters of the scalability model solver, and algorithms for solving the scalability model. The scalability model is very general and allows the modeling of data servers with numerous configurations.

1 Introduction

Perhaps one of the most important examples of large-scale, data-intensive, geographically distributed, information systems is NASA's Earth Observing System (EOS) Data and Information System (EOSDIS). EOS is a NASA mission aimed at studying the planet Earth. A series of satellites with scientific instruments aboard will be collecting important data about the Earth's atmosphere, land, and oceans over a period of 15 years. This mission will generate an estimated tera bytes/day of raw data which will be processed to generate higher level data products [2]. Raw data received from the satellites is first stored as Level 0 (L0) data which may then be transformed after successive processing into levels 2 through 4 (L2 - L4). Data received from the satellites and the data products generated from them will be stored at various Distributed Active Archive Centers (DAACs) located throughout the United States. An important component of a DAAC is the Data Server—the subsystem that stores and distributes data as requested by EOSDIS users.

The Data Server stores its information using a hierarchical mass storage system that uses a combination of automated tape libraries and disk caches to provide cost-effective storage for the large volumes of data held by the Data Server. Performance studies and workload characterization methods and software for hierarchical mass storage systems are reported in [3, 5, 6, 7, 8].

In this paper, we present a model for the scalability analysis of the Data Server subsystem of the EOSDIS Core System (ECS). The goal of the model is to analyze if the planned architecture of the Data Server will support an increase in the workload with the possible upgrade and/or addition of processors, storage subsystems, and networks. This analysis does not contemplate new architectures that may be needed to support higher demands.

The remaining sections of this paper are organized as follows. Section 2 provides a summary of the architecture of ECS's Data Server as well as a high level description of the Ingest and Retrieval operations as they relate to ECS's Data Server. This description forms the basis for the development of the scalability model of the data server. Section 3 presents the scalability model and the methodology used to solve it. This section describes the structure of the scalability model, input parameters, algorithms for computing parameters of the scalability model solver, algorithms for solving the scalability model, and the assumptions and rationale behind these assumptions. The scalability model takes into account the proposed hardware and software architecture. The model is quite general and allows the modeling of data servers with numerous configurations.

2 Ingest and Retrieval Operations

This section provides a high level description of the Ingest and Retrieval workloads of the ECS's Data Server. This description forms the basis for the development of a model to analyze the scalability of the Data Server. The scalability analysis entails determining whether the current architecture of the ECS Data Server supports an increase in the workload intensity with possibly more processing and data storage elements of possibly higher performance.

2.1 Subsystems of the Data Server

The following subsystems of the Data Server will be considered for the purpose of the scalability analysis considered in this study:

Software Configuration Items:

- **Science Data Server (SDSRV):** responsible for managing and providing access to non-document earth science data.
- **Storage Management (STMGT):** stores, manages, and retrieves files on behalf of other SDPS components.

Hardware Configuration Items:

- **Access Control and Management (ACMHW):** supports the Ingest and Data Server subsystems that interact directly with users. Of particular interest here is the SDSRV.

- **Working Storage (WKSHW):** provides high performance storage for caching large volumes of data on a temporary basis.
- **Data Repository (DRPHW):** provides high capacity storage for long-term storage of files.
- **Distribution and Ingest Peripherals (DIPHW):** supports ingest and distribution via physical media.

2.2 Ingest Data Operation

The diagram in Figure 1 depicts the flow of control and data for the Ingest process. We have not included Document Repository nor the Document Data Server due to their small impact on scalability if compared with ingest of L0 data. Circles in the diagram represent processes. The labels in square brackets beside each process indicate the hardware configuration item they execute on. Bolded labels indicate hardware configuration items that belong to the Data Server.

The main aspects of the diagram of figure 1 are discussed below:

- Incoming L0 data is first stored into the system on the Staging Disk, and then into AMASS's cache—the hierarchical mass storage systems' disk cache for files. The metadata are extracted and entered into a Metadata database managed by Sybase and the actual data are archived. This is depicted in figure 2.
- The SDPF (Science Data Processing Function) represents the users of the Ingest system and negotiates with the Ingest Request Manager for coordination of transferring data into the Ingest system.
- Data are initially entered through an interactive GUI interface, or, most of the time from external data providers through ftp or direct transfer of files, if that is done on the same local network, into the Staging Disk.
- The actual data is then transferred into AMASS' disk cache. From the cache, the data migrates to robotically mounted tapes managed by AMASS. The metadata extracted from the data is stored into a Metadata Database managed by Sybase.
- The SDSRV (Science Data Server) gives the metadata templates to the Ingest Request Manager for it to extract metadata.
- There are two and sometimes three SDSRV's and one STMGT (Storage Management) processes. The Ingest Request Manager process selects which SDSRV to use.

The scalability analysis will among other things determine possible performance bottlenecks. The staging disk, the AMASS disk cache, and the metadata extraction process are likely candidates for bottlenecks.

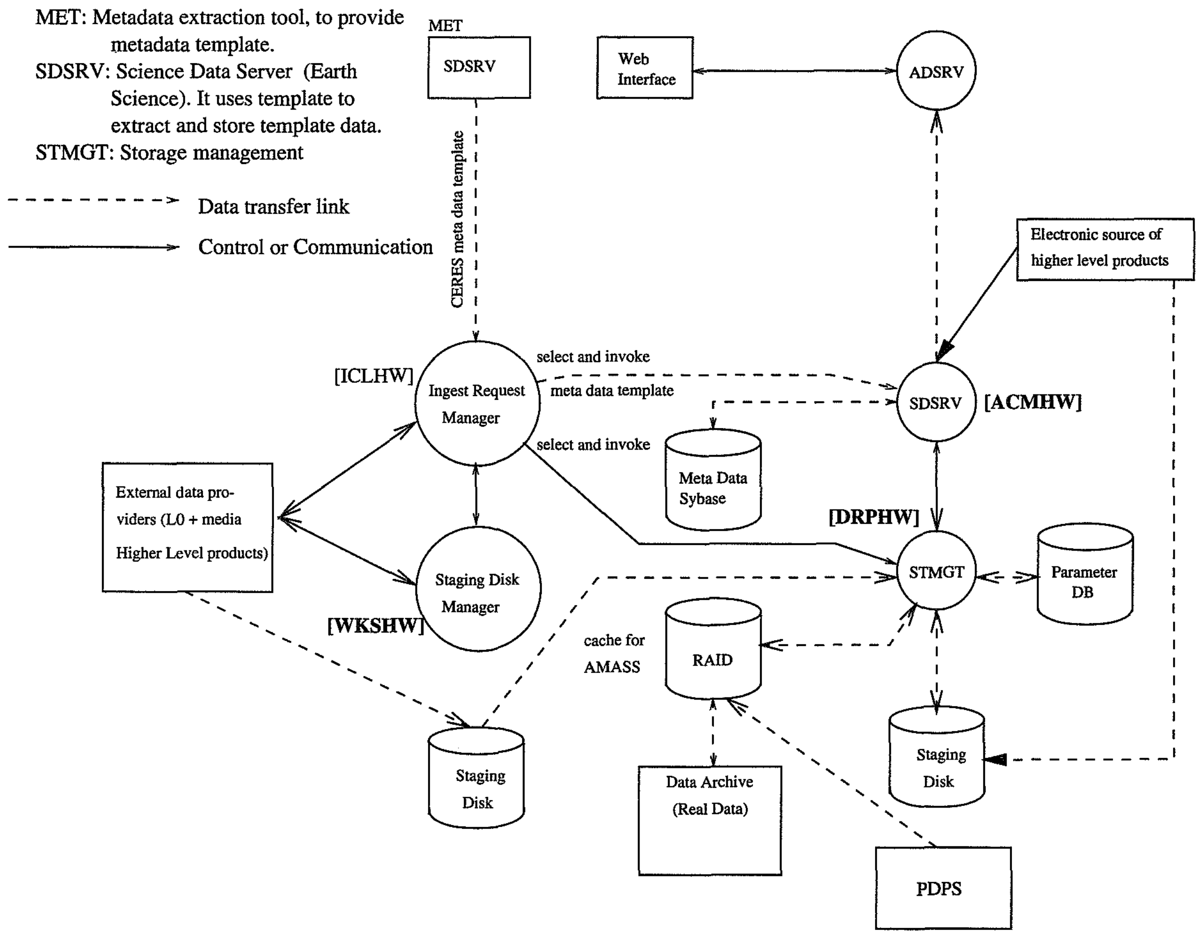


Figure 1: L0 Ingest Control and Data Flow.

2.3 Retrieval Operation

This section examines the retrieval and processing operation on L1+ data. Figure 3 depicts the flow of control and data for this operation. Circles in the diagram represent processes. The labels in square brackets beside each process indicate the hardware configuration item they execute on. Bolded labels indicate hardware configuration items that belong to the Data Server.

The retrieval operation proceeds in the following three stages:

Stage 1: Checking data and deciding what processing is required:

- SDSRV initiates the retrieval process by notifying the Subscription Server of the new data arrival.
- The Subscription Server performs a subscription checkfor this data and performs an appropriate notification, e.g., email notification, etc.

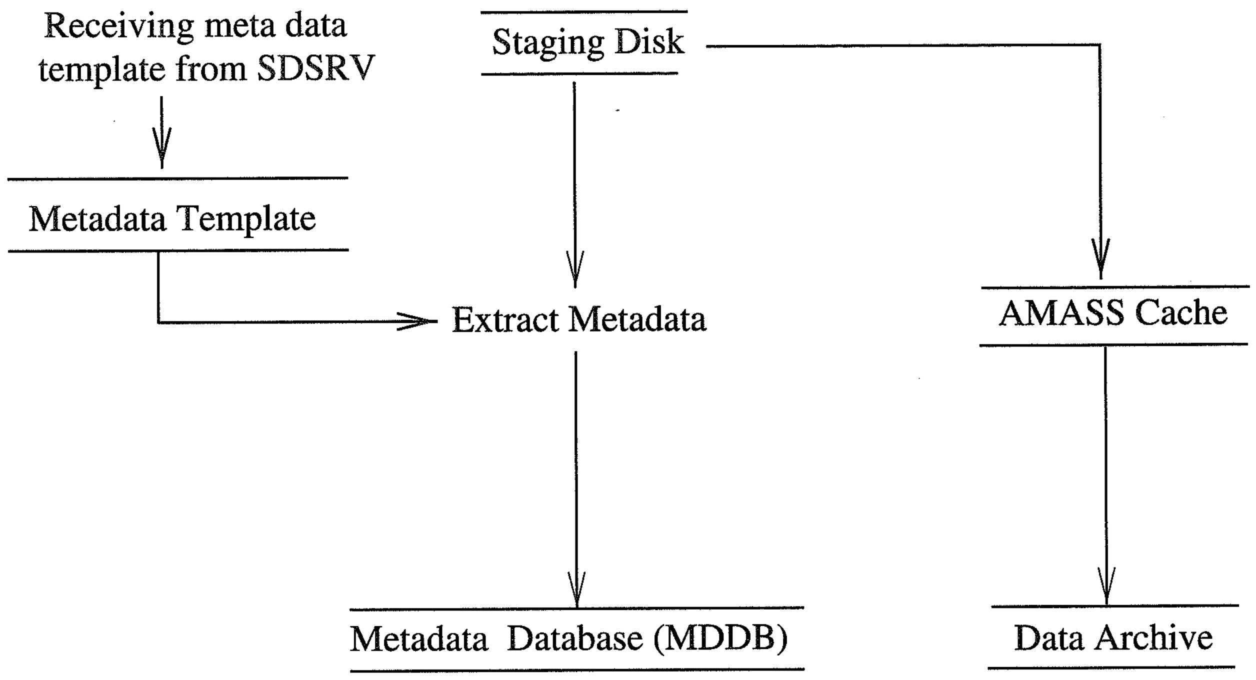


Figure 2: Data Flow Diagram for Ingest Data.

- The Subscription Server notifies PDPS PLANG of new data arrival.
- PLANG figures out (e.g., retrieves) a processing plan and based on the processing plan, passes the processing request to PRONG.
- PDPS PRONG connects to the appropriate SDSRV (may not be the SDSRV which initiated the retrieval and processing operations).

Stage 2: Retrieving data:

- The SDSRV requests that the Data Distribution Services CSCI (DDIST) retrieves the data files.
- SDSRV $\longrightarrow^{requests}$ DDIST $\longrightarrow^{requests}$ STMGT. The STMGT retrieves the files from AMASS archive into the AMASS cache if it is already not present in the cache.
- SDSRV notifies PRONG of data (identified by UR) availability.

Stage 3: Processing data and archiving, both data and metadata:

- PRONG transfers the retrieved data from the Working Storage to local PDPS disk. (If the AMASS cache and Working Storage are on different devices, then data must be first moved from the former to the latter.)
- PRONG processes the retrieved data to produce a higher level product.

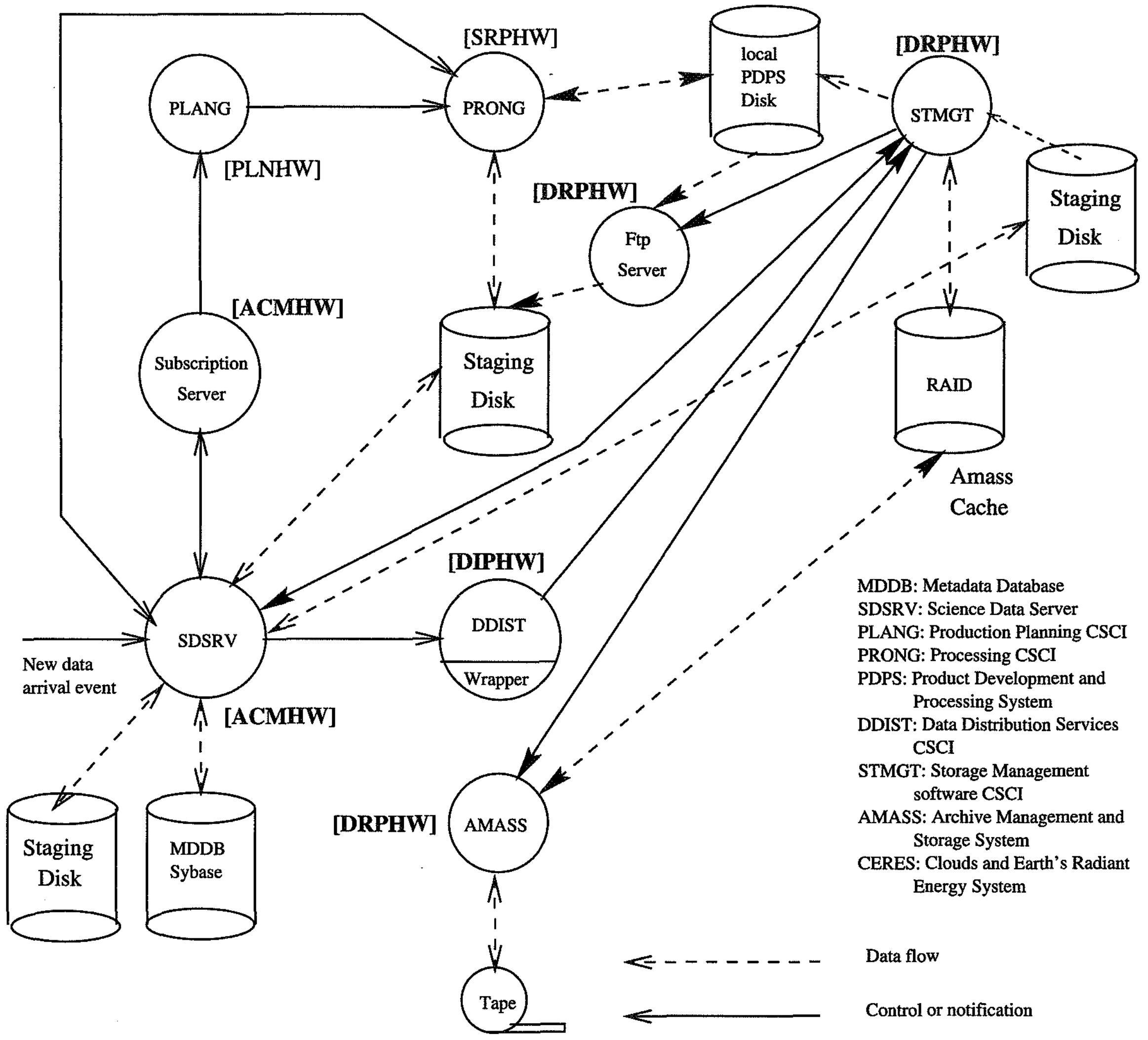


Figure 3: A Flow Diagram of Data Retrieval and Processing

- PRONG processes the data to a higher-level product and extracts metadata from the higher-level data using the Metadata Extraction Tool and populates the target metadata template and writes a metadata file (on MDDB Sybase).
- PDPS PRONG sends an insert request to SDSRV.
- SDSRV $\longrightarrow^{requests}$ STMGT $\longrightarrow^{requests}$ AMASS. The AMASS file manager archives the files. Archiving is done in two steps:
  - STMGT copies data from PDPS (local disk) to Working Storage via an ftp command.
  - data are copied from the Working Storage to AMASS cache (and then to AMASS archive).

- SDSRV inserts metadata in the Metadata Database (MDDB) and then notifies PRONG that the archival operation has been completed.

2.4 Assumptions

The various software processes shown in the previous subsection were mapped into the different hardware configuration items for the GSFC, EDC, and LaRC DAACs. The following assumptions were made when developing the scalability model.

- Processing of "Ingest data" and "Data retrieval and processing" constitute the main load on the Data Server. Thus, we modelled only these two operations.
- We did not model users' requests for data to be subsetted or subsampled nor we modelled compressed data.
- In data retrieval operations, PLANG retrieves a processing plan from a database (e.g., Sybase).
- The AMASS cache and the working storage may be implemented on the same disk.
- Servers that are not potential bottlenecks were not considered in the model. Examples include the "subscription server" and PDPS.
- We assume that mean arrival rate of both types of requests (ingest data and data retrieval) and service demands of these requests at various service stations are available or can be easily estimated.

3 A Scalability Model

We now describe our scalability model for the ECS's Data Server and our methodology for solving this model. We describe the structure of the scalability model, input parameters, algorithms for computing parameters of the scalability model solver, and algorithms for solving the scalability model. We describe our assumptions and rationale for these assumptions.

The scalability model is based on our understanding of the architecture of ECS's Data Server and the Ingest and Retrieval operations described in the previous section. The sole purpose of the model is to analyze the scalability of the Data Server, i.e., to determine whether the current architecture of the ECS Data Server can support an increase in the workload intensity.

3.1 A Framework for Scalability Analysis

Figure 4 gives the structure of the scalability model. The "Scalability Model Generator" collects information from three input files (these files define the modeling information on the ECS's data server and the workload) and processes this information to create an output file which contains inputs to the "Scalability Model Solver". This solver uses queuing network [4] techniques to obtain desired performance measures such as response times per workload, device utilizations, bottleneck indications, and queue lengths.

The first input file to the Scalability Model Generator, "Hardware Objects", defines the hardware resources (e.g., processors, disks, networks, and tape libraries) of the Data Server. The second input file to the Scalability Model Generator, "Workloads and Execution Flow", completely characterizes the workload that drives the Data Server. The third input file to the Scalability Model Generator, "Processes", defines the parameters of the software modules that will be executed on hardware servers by arriving requests for service (i.e., the workload).

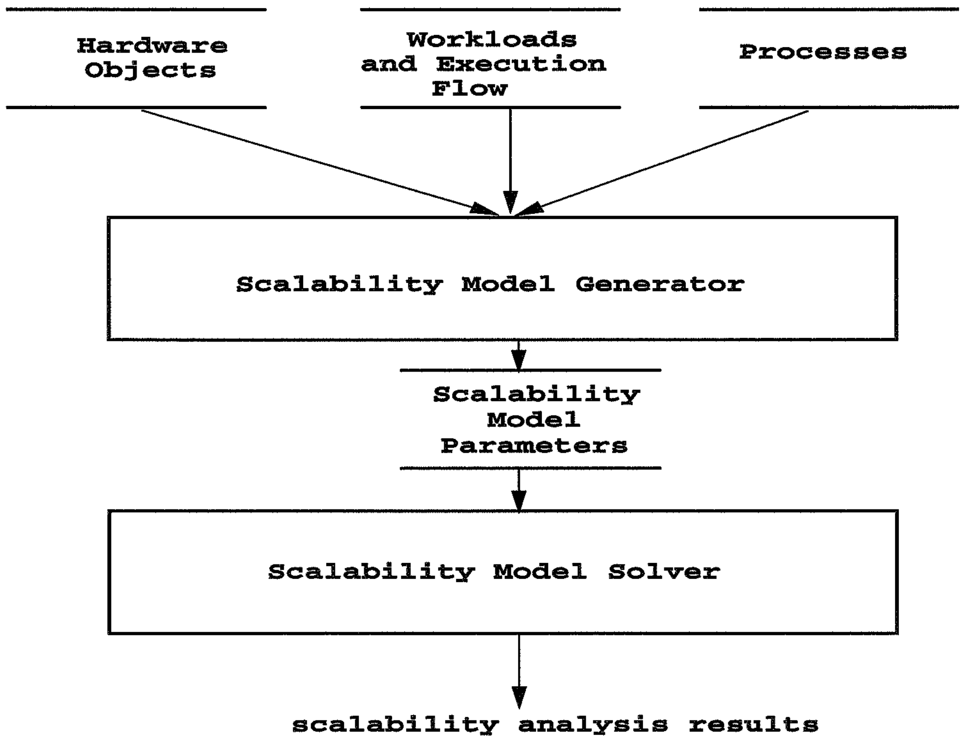

Hardware Objects
Workloads and Execution Flow
Processes
Scalability Model Generator
Scalability Model Parameters
Scalability Model Solver
scalability analysis results

Figure 4: Scalability Model Framework.

The Scalability Model Generator reads information in these three files, processes this information, and generates an output file that contains the service demands for every resource in the queuing network model of Service demand is the total service time of a request of a certain workload type at a given device. The service demand does not include any time waiting to get access to the device. Waiting times are obtained by solving the model. The equations that form the basis of computation of service demands are presented in Section 3.3. The Scalability Model Solver reads information about the service demand from this file and solves the queuing network model for desired performance measures. The underlying equations that form the basis for a solution are described in Section 3.4.

3.2 Parameters for the Scalability Model

The parameters used in the scalability model are:

- $\mathcal{P}$: set of processes
- NCPUs_s: number of processors of server s
- SPint_s: SPECint95 rating of server s
- SPfp_s: SPECfp95 rating of server s
- TypeSP_p: type (e.g., int or fp) of the SPEC rating used to specify the computation demand for process p.
- SP_p: SPEC rating of the machine used to measure the computation demand of process p.
- ComputeDemand_p: compute demand of process p measured at a machine with SPEC rating SP_p, in seconds
- $\text{PExec}_{p,w}$: probability that process p executes in workload w
- $\text{Seek}_{d,s}$: average seek time of single disk d of server s, in seconds
- $\text{Latency}_{d,s}$: average rotational latency of single disk d of server s, in seconds
- $\text{TransferRate}_{d,s}$: transfer rate of single disk d of server s, in MBytes/sec
- $\text{Hit}_{da,s}$: cache hit ratio for disk array d
- $\text{RAIDSeek}_{da,s}$: average seek time at any of the disks that compose disk array da at server s, in seconds
- $\text{RAIDLatency}_{da,s}$: average rotational latency at any of the disks that compose disk array da at server s, in seconds
- $\text{RAIDRate}_{da,s}$: transfer rate of any of the disks that compose disk array da at server s, in Mbytes/sec
- $\text{NTDrives}_{t,s}$: number of tape drives of tape library t at server s.
- $\text{NRobots}_{t,s}$: number of robots of tape library t at server s.
- $\text{Rewind}_{i,t,s}$: rewind time of tape drive i of tape library t at server s.
- $\text{MaxTSearch}_{i,t,s}$: maximum search time of tape drive i of tape library t, in seconds at server s.
- $\text{TapeRate}_{i,t,s}$: transfer rate of tape drive i of tape library t at server s, in Mbytes/sec
- $\text{Exchanges}_{t,s}$: number of tape exchanges per hour for each robot of tape library t at server s. (Each exchange involves putting the old tape in the tape library and loading the new tape into the tape drive.)

- FilesPerMount$_{p,t,s}$: average number of files accessed per mount by process p at tape library t at server s.
- FileSizePerMount$_{p,t,s}$: average size of files accessed by process, in Kbytes p per mount at tape library t at server s.
- Bandwidth$_n$: bandwith of network n, in Mbps
- NType$_n$: type of network n.
- NumBlocks$_{d,s,p}$: number of blocks accessed by process p at single disk d at server s.
- BlockSize$_{d,s,p}$: block size for each access to single disk d at server s by process p, in KBytes
- NumBlocksRead$_{da,s,p}$: number of blocks read by process p from disk array da at server s
- NumBlocksWritten$_{da,s,p}$: number of blocks written by process p to disk array da at server s
- StripeUnitSize$_{da,s}$: size of the stripe unit for disk array da at server s, in Kbytes
- Server$_p$: server in which process p is allocated
- λ_w: arrival rate of workload w, in requests/sec
- $\mathcal{P}_w$: set of processes executed by workload w
- $\pi_w = \{(p, x) \mid p \in \mathcal{P} \text{ and } x = Pr[p \text{ is executed in workload } w\}$: process flow within workload w.
- PNet$_{n,w}$: probability that network n is traversed by workload w.
- Volume$_{n,w}$: total data volume transferred through network n by workload w, in Kbytes

The input parameters for the Scalability Model Solver are:

- $D_{i,p,w}$: average service demand of process p in workload w at device i, i.e., the total time spent by the process at the device for workload w. This time does not include any queuing time.
- λ_w: average arrival rate of requests of workload w that arrive to ECS's Data Server.

3.3 Algorithms for Computing the Scalability Model Solver Parameters

In this section, we derive expressions for computing service demands for workloads at various types of devices. The service demand at a device due to a task is defined as the multiplication of the visit count of the task to the device and the service time of the task per visit to the device. The service demand represents the total average time spent by the task at the device.

3.3.1 Computation of Service Demands for Processors

The service demand that a task in workload w presents at a server s due to the execution of a process p is given by:

$$D_{s,p,w} = \frac{\text{ComputeDemand}_p \times \text{PExec}_{p,w}}{\text{ScaleFactor}(p, s)} \tag{1}$$

where

$$\text{ScaleFactor}(p, s) = \begin{cases} \text{SPint}_s / SP_p & \text{if TypeSP}_p = \text{int} \\ \text{SPfp}_s / SP_p & \text{if TypeSP}_p = \text{fp} \end{cases} \tag{2}$$

Since ComputeDemand_p is given for a processor of certain rating, $\text{ScaleFactor}(p, s)$ is used to normalize the process service time to the speed-rating of the current processor.

The service demand, $D_{s,w}$, of a workload w at the CPU of server s is then

$$D_{s,w} = \sum_{\forall p \in \mathcal{P}_w \ | \ s = \text{Server}_p} D_{s,p,w} \tag{3}$$

3.3.2 Computation of Service Demands for Single Disks

The service demand that a task in workload w presents to a disk d at a server s due to the execution of a process p is given by:

$$\begin{aligned} D_{d,s,p,w} &= \text{PExec}_{p,w} \times \text{NumBlocks}_{d,s,p} \times \\ &\left[\text{Seek}_{d,s} + \text{Latency}_{d,s} + \frac{\text{BlockSize}_{d,s,p}}{\text{TransferRate}_{d,s} \times 1000} \right] \end{aligned} \tag{4}$$

The term "$\text{Seek}_{d,s} + \text{Latency}_{d,s} + \frac{\text{BlockSize}_{d,s,p}}{\text{TransferRate}_{d,s} \times 1000}$" denotes the time the disk takes to fetch one block of data.

The service demand, $D_{d,s,w}$, of a workload w at disk d of server s is then

$$D_{d,s,w} = \sum_{\forall p \in \mathcal{P}_w \ | \ s = \text{Server}_p} D_{d,s,p,w} \tag{5}$$

3.3.3 Computation of Service Demands for Disk Arrays

The computation of service demands for disk arrays is involved and is done in several steps. The number of blocks that a process p reads at a disk (i.e., the number of stripe accesses) in disk array da at server s is given by

$$\text{NumBlocksReadPerDisk}_{da,s,p} = \left\lceil \frac{\text{NumBlocksRead}_{da,s,p} \times \text{BlockSize}_{d,s,p}}{5 \times \text{StripeUnitSize}_{da,s}} \right\rceil \tag{6}$$

where $\lceil x \rceil$ denotes the ceiling of x. The numerator denotes the total volume of information read from all five disks in the disk array and the denominator denotes the volume of information read from all five disks in a single stripe group access.

The service time to process each stripe request at each disk is given by the following equation: (The first subexpression indicates that the seek time is amortized over all stripe unit accesses.)

$$\text{ServiceTimePerDisk}_{da,s,p} = \frac{\text{RAIDSeek}_{da,s}}{\text{NumBlocksReadPerDisk}_{da,s,p}} + \text{RAIDLatency}_{da,s} + \frac{\text{StripeUnitSize}_{da,s}}{\text{RAIDRate}_{da,s}} \tag{7}$$

The service demand due to read requests at a disk in disk array da at server s due to execution of process p in workload w is given by the following equation: (Since a disk array has a data cache, term $(1 - \text{Hit}_{da,s})$ denotes the probability that data to be read is not available in the cache and a read access will have to be made.)

$$\text{ReadServiceDemandPerDisk}_{da,s,p,w} = \text{NumBlocksReadPerDisk}_{da,s,p} \times \text{ServiceTimePerDisk}_{da,s,p} \times \text{PExec}_{p,w} \times (1 - \text{Hit}_{da,s}) \tag{8}$$

Now the service demand, $D^r_{da,s,p,w}$, due to read requests at disk array da at server s due to execution of process p in workload w is given by the following equation:

$$D^r_{da,s,p,w} = \frac{H_5 \times \text{ReadServiceDemandPerDisk}_{da,s,p,w}}{1 - \text{USingleDisk}_{da,s,p,w}} \tag{9}$$

where $H_5 = \sum_{j=1}^{5} 1/j = 2.28$ and $\text{USingleDisk}_{da,s,p,w}$ is given by Eq. (14). The term H_5 shows up in the expression because a read request at the disk array is complete only after the last read at its disks is done. This approximation is based on [5].

The service demand, $D^r_{da,s,w}$, of a workload w at the disk array da of server s is then

$$D^r_{da,s,w} = \sum_{\forall p \in \mathcal{P}_w \mid s = \text{Server}_p} D^r_{da,s,p,w} \tag{10}$$

The computation of the service demand due to write requests at disk array da at server s due to the execution of process p in workload w is similar. The computation of the number of blocks that a process p writes at a disk (i.e., the number of stripes written) in the disk array da at server s is somewhat different and is given by the following equation: (The (4/5) term in the denominator is due to the fact that a parity block is generated for every four blocks written onto the disks. Thus 25% additional data is generated.)

$$\text{NumBlocksWrittenPerDisk}_{da,s,p} = \left\lceil \frac{\text{NumBlocksWritten}_{da,s,p} \times \text{BlockSize}_{d,s,p}}{(4/5) \times \text{StripeUnitSize}_{da,s}} \right\rceil \tag{11}$$

$$\begin{aligned}\text{WriteServiceDemandPerDisk}_{da,s,p,w} &\doteq \text{NumBlocksWrittenPerDisk}_{da,s,p} \times \\ &\quad \text{ServiceTimePerDisk}_{da,s,p} \times \\ &\quad \text{PExec}_{p,w} \qquad (12)\end{aligned}$$

$$D^{w}_{da,s,p,w} = \frac{H_5 \times \text{WriteServiceDemandPerDisk}_{da,s,p,w}}{1 - \text{USingleDisk}_{da,s,p,w}} \qquad (13)$$

where

$$\begin{aligned}\text{USingleDisk}_{da,s,p,w} &= \text{PExec}_{p,w} \times \lambda_w \times \\ &\quad [(\text{NumBlocksReadPerDisk}_{da,s,p} + \\ &\quad \text{NumBlocksWrittenPerDisk}_{da,s,p}) \times \\ &\quad \text{ServiceTimePerDisk}_{da,s,p}] \qquad (14)\end{aligned}$$

The service demand, $D^{w}_{da,s,w}$, of a workload w at the disk array da of server s is then

$$D^{w}_{da,s,w} = \sum_{\forall p \in \mathcal{P}_w \mid s = \text{Server}_p} D^{w}_{da,s,p,w} \qquad (15)$$

3.3.4 Computation of Service Demands for Tape Libraries

The computation of the service demands for tape drives and robots at a tape library is involved and is done in several steps.

The total average seek time that a process p experiences at tape drive i in tape library t at server s is given by the following equation: (The factor "1/2" is due to the fact that the first file access will result in searching half the tape on the average and the factor "1/3" shows up because the remaining file accesses will require searching 1/3 of the tape on the average.)

$$\text{AverageSeekTime}_{t,s} = \text{MaxTSearch}_{i,t,s} \times [1/2 + (\text{FilesPerMount}_{p,t,s} - 1)/3] \qquad (16)$$

The average tape mount time in seconds at tape drive i in tape library t at server s is given by

$$\text{MountTime}_{i,t,s} = 3,600/2 \times \text{Exchanges}_{t,s} \qquad (17)$$

The time that tape drive i in tape library t at server s takes to serve a file access request is given by

$$\begin{aligned}\text{TapeDriveServiceTime}_{i,t,s} &= \text{AverageSeekTime}_{t,s} + \\ &\quad \frac{\text{FilesPerMount}_{p,t,s} \times \text{FileSizePerMount}_{p,t,s}}{\text{TapeRate}_{i,t,s}} + \\ &\quad \text{Rewind}_{i,t,s} \qquad (18)\end{aligned}$$

The average robot service time is then

$$\text{RobotServiceTime}_{t,s} = 2 \times \text{MountTime}_{i,t,s} \tag{19}$$

So, the service demand at the tape drive i of tape library t of server s due to the execution of process p in workload w is

$$D^{tape\ drive}_{i,t,s,p,w} = \text{PExec}_{p,w} \times \text{TapeDriveServiceTime}_{i,t,s}/\text{NTDrives}_{t,s} \tag{20}$$

The service demand at the tape drive i of tape library t of server s due to workload w is

$$D^{tape\ drive}_{i,t,s,w} = \sum_{\forall p \in \mathcal{P}_w \ \mid \ s=\text{Server}_p} D^{tape\ drive}_{i,t,s,p,w} \tag{21}$$

The average service demand at any robot of tape library t of server s due to the execution of process p in workload w is

$$D^{robot}_{t,s,p,w} = \text{PExec}_{p,w} \times \text{RobotServiceTime}_{t,s}/\text{NRobots}_{t,s} \tag{22}$$

The service demand at any robot of tape drive i of tape library t of server s due to workload w is

$$D^{robot}_{t,s,w} = \sum_{\forall p \in \mathcal{P}_w \ \mid \ s=\text{Server}_p} D^{robot}_{t,s,p,w} \tag{23}$$

3.3.5 Computation of Service Demands for Networks

The service demand of workload w presents at network n is given by the following equation: (The term "$Volume_{n,w}/Bandwidth_n$" denotes the time taken by the network to transfer the data for a task in workload w.)

$$D^{network}_{n,w} = \frac{\text{PNet}_{n,w} \times \text{Volume}_{n,w} \times 8}{\text{Bandwidth}_n \times 1000} \tag{24}$$

3.4 The Scalability Model

The scalability model uses queuing network (QN) models to determine the degree of contention at each of the devices that compose ECS's Data Server. The QN model used in this case is a multiclass open QN [4] with additional approximations to handle the case of disk arrays and to handle the instances of simultaneous resource possession that appear when modeling automated tape libraries [3]. The QNs used also allow for load dependent devices. Load dependent devices are used in the model to handle the following situations:

- Symmetric multiprocessors: this case is characterized by a single queue for multiple servers. In this case, the service rate $\mu(k)$ of the CPU as a function of the number of requests k is given by $k.\mu$ for $k \leq J$ and $J.\mu$ for $k > J$ where J is the number of CPUs and μ is the service rate of each CPU.

- Collision-based LANs: in this case, the throughput of the LAN decreases as the load increases due to an increase in the number of collisions. This can be modeled by using an appropriate service rate function $\mu(k)$ as a function of the load on the network [1].

An open multiclass QN is characterized by the number R of classes, the number K of devices, by a matrix $D = [D_{i,r}]$ $i = 1, \cdots K$, $r = 1, \cdots, R$ of service demands per device per class, and by a vector $\vec{\lambda} = (\lambda_1, \cdots, \lambda_R)$ of arrival rates per class. For each device, one has to indicate its type. The following types of devices are allowed in the QN model:

- Delay devices: no queues are formed at these devices.
- Queuing Load Independent (LI) devices: queues are formed at these devices but the service rate of the device does not depend on the number of requests queued for the device.
- Load Dependent device (LD): queues are formed at these devices but the service rate of the device depends on the number of requests queued for the device. In the case of load dependent devices, one has to provide the service rate multipliers (see [4]) for each value of the number of customers. In most cases—this is true for multiprocessors and collision-based LANs—the value of the service rate multipliers saturates very quickly with the number of requests. Therefore, we only need to provide a small and finite number of service rate multipliers for each LD device.
- Disk Array: this is a special type of device used to model disk arrays (see Fig. 5 for a depiction of this type of device).

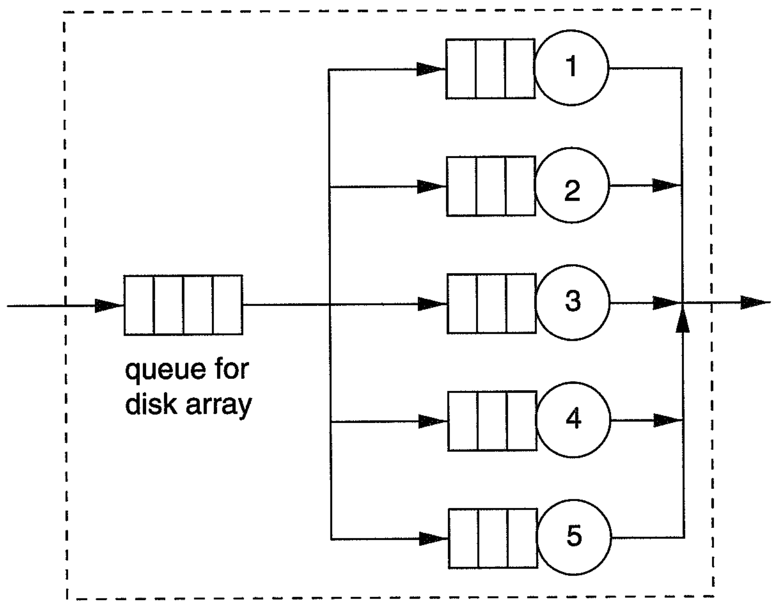


Figure 5: Disk Array.

The output results of an open multiclass QN are:

- $R'_{i,r}(\vec{\lambda})$: average residence time of class r requests at device i, i.e., the total time—including queuing and service—spent by requests of class r at device i.
- $R_r(\vec{\lambda})$: average response time of requests of class r. $R_r(\vec{\lambda}) = \sum_{i=1}^{K} R'_{i,r}(\vec{\lambda})$.

- $U_i(\vec{\lambda})$: utilization of device i.
- $n_{i,r}(\vec{\lambda})$: average number of requests of class r present at device i.
- $n_i(\vec{\lambda})$: average number of requests of at device i. $n_i(\vec{\lambda}) = \sum_{r=1}^{R} n_{i,r}(\vec{\lambda})$.

The basic equations for open multiclass QNs are (see [4]):

$$\begin{aligned}
U_{i,r}(\vec{\lambda}) &= \lambda_r D_{i,r} \\
U_i(\vec{\lambda}) &= \sum_{r=1}^{R} U_{i,r}(\vec{\lambda}) \\
\bar{n}_{i,r}(\vec{\lambda}) &= \frac{U_{i,r}(\vec{\lambda})}{1 - U_i(\vec{\lambda})} \\
R'_{i,r}(\vec{\lambda}) &= \begin{cases} D_{i,r} & \text{delay device} \\ \dfrac{D_{i,r}}{1 - U_i(\vec{\lambda})} & \text{LI device} \end{cases} \\
R_r(\vec{\lambda}) &= \sum_{i=1}^{K} R'_{i,r}(\vec{\lambda}) \\
n_i(\vec{\lambda}) &= \sum_{r=1}^{R} n_{i,r}(\vec{\lambda})
\end{aligned}$$

The extension to LD devices is given in [4].

4 Preliminary Results

The model presented here was applied to the Goddard Space Flight Center (GSFC) Data Server's architecture. Figure 6 displays the response time versus the arrival rate of data ingested in GB/day. The rate at which data is retrieved is assumed to be twice the ingest rate. It should be pointed out that at this point, these are very preliminary results and are not intended to reflect the performance of GSFC's Data Server. Data collection efforts are still in progress. The purpose of Figure 6 is to illustrate the type of results one can obtain from the type of models discussed in this paper. As it can be seen, in the curve, the ingest workload starts to saturate at approximately 160 GB/day while the retrieval workload supports a much higher data rate.

5 Concluding Remarks

In this paper, we derived the algorithms and expressions to be used to convert data describing the software and hardware architecture of ECS's Data Server into a scalability model. The model will be used to verify how well the Data Server supports an increase in workload intensity while maintaining reasonable performance. The scalability model is based on queuing network models that are automatically generated from the description of the architecture and the workload.

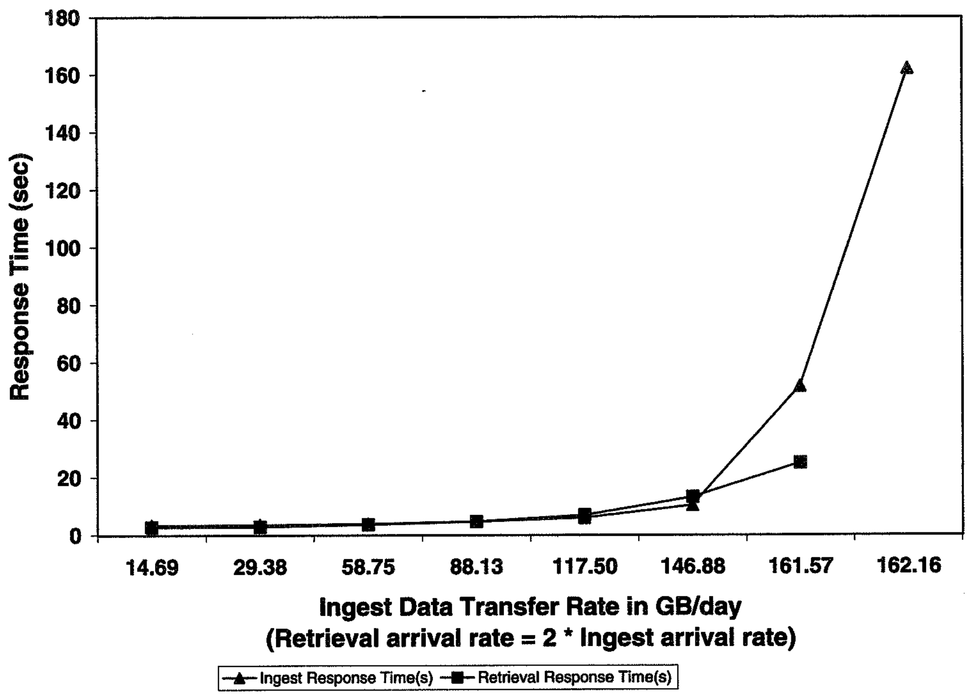


Figure 6: Response time versus data rate for GSFC.

References

[1] E. D. Lazowska, J. Zahorjan, G. S. Graham, and K. C. Sevcik, *Quantitative System Performance: Computer System Analysis Using Queuing Network Models*, Prentice Hall, Englewood Cliffs, N. J., 1984.

[2] B. Kobler, J. Berbert, P. Caulk, and P. C. Hariharan, *Architecture and Design Storage and Data Management for the NASA Earth Observing System Data and Information System (EOSDIS)*, Fourteenth IEEE Symposium on Mass Storage Systems (Second International Symposium), Monterey, CA, September 11-14, 1995.

[3] D. A. Menascé, O. I. Pentakalos, and Y. Yesha, An Analytic Model of Hierarchical Mass Storage Systems with Network-Attached Storage Devices, *Proc. of the 1996 ACM Sigmetrics Conference*, Philadelphia, PA, May 1996.

[4] D. A. Menascé, V. A. F. Almeida, and L. W. Dowdy, *Capacity Planning and Performance Modeling: from mainframes to client-server systems*, Prentice Hall, Upper Saddle River, NJ, 1994.

[5] O. I. Pentakalos, D. A. Menascé, M. Halem, and Y. Yesha, *Analytical Performance Modeling of Hierarchical Mass Storage Systems*, IEEE Transactions on Computers, Vol. 46, No. 10, pp. 1103-1118.

[6] O. I. Pentakalos, D. A. Menascé, and Y. Yesha, *Pythia and Pythia/WK: Tools for the Performance Analysis of Mass Storage Systems*, Software Practice and Experience, October 1997.

[7] O. I. Pentakalos and D. A. Menascé, *Automated Clustering Based Workload Characterization for Mass Storage Systems*, Fifth NASA Goddard Space Flight Center Conference on Mass Storage Systems and Technologies, College Park, MD, September 17-19, 1996.

[8] O. I. Pentakalos , D. A. Menascé, Y. Yesha, and M. Halem, *An Approximate Performance Model of a Unitree Mass Storage System*, Fourteenth IEEE Symposium on Mass Storage Systems (Second International Symposium), Monterey, CA, September 11-14, 1995.

A Study on the Use of Tertiary Storage in Multimedia Systems

Leana Golubchik
Department of Computer Science
A.V. Williams Building, Room 4129
University of Maryland
College Park, MD 20742
Email: leana@cs.umd.edu
Phone: (301) 405-2751; Fax: (301) 405-6707

Raj Kumar Rajendran
Department of Computer Science
500 W. 120th Street
Columbia University
New York, NY 10027
Email: kumar@cs.columbia.edu

Abstract

In spite of the dramatic improvements in the cost of secondary storage, magnetic disks alone are inadequate for meeting the storage and delivery needs of many modern applications. Novel designs of storage hierarchies are needed if we wish to take advantage of the low cost of tape media but still provide reasonable performance characteristics, in the context of modern applications. Specifically, we are interested in multimedia storage server applications, which in addition to high storage and high bandwidth requirements must support display of continuous data streams (e.g., video) and thus satisfy real time constraints. In this paper we focus on the issues and tradeoffs involved in the design of multimedia tertiary storage systems that store and retrieve *heterogeneous* multimedia objects.

1 Introduction

In spite of the dramatic improvements in the cost of secondary storage, magnetic disks alone are inadequate for meeting the storage and delivery needs of many modern applications [6]. Tape media is still two orders of magnitude less expensive than magnetic disk storage; although, tape drives exhibit access latencies two to four orders of magnitude larger than that of disk drives. Novel designs of storage hierarchies are needed if we wish to take advantage of the low cost of tape media but still provide reasonable performance characteristics, in the context of modern applications. Specifically, we are interested in multimedia storage server applications (such as video-on-demand systems), which in addition to high storage and high bandwidth requirements must support display of continuous data streams (e.g., video) and thus satisfy real time constraints — that is, once a display of an object begins, it must continue at a specified bandwidth for the entire duration of that object's display.

Thus far, relatively little work has been done on server architectures that use tape storage as an online repository of data, especially in the context of multimedia systems that store and deliver *heterogeneous* objects (the heterogeneity is reflected in the different sizes and transfer rate requirements of the multimedia objects, as defined later in the paper). The success of RAID technology, and the low bandwidths of then current tape-drive technology spurred some early work on striping of data over multiple tapes [2, 1] (due to space limitations, we will only mention a few representative existing works on this topic). At the time, the conclusions were, to a large extent, that: (1) there was limited utility to striping, as contention for the drives and the large latency reduced the throughput of the system, (2) such a strategy was useful only for sequential access and backup, and (3) tertiary storage was not a viable delivery mechanism for continuous media data (such as video), as tape-striping performed poorly under concurrent access. In a later work [4] the authors showed that, depending on the system workload, tape striping can be a viable and attractive alternative. Some of the issues encountered in multi-level multimedia server design were addressed in [3].

Given the current tape drive technology, a single drive can simultaneously support multiple continuous streams of data, and thus it is important to devise schemes to efficiently share the bandwidth among many streams while satisfying the real-time constrains. The idea of servicing multiple streams from one tape system by buffering data on secondary storage was exploited in [7], where the notion of a *cycle* was introduced in the context of tertiary storage. During a cycle, data needed to serve each active stream for the duration of one cycle is read and buffered on secondary storage and then delivered to users in the next cycle. (The "offset" between data retrieval and data delivery is needed in order to maintain the real-time constraints.) Only as many streams as can be serviced while still meeting the real-time constraints are admitted into the system. Note that, in the remainder of the paper "buffering" will refer to caching of data, read from tertiary storage, on secondary storage, since this is the level of the storage hierarchy we are studying here.

The design of such multimedia tertiary storage systems raises some interesting questions, which include: (a) how much data should be read in each cycle — the larger the amount, the better the tertiary bandwidth utilization, but the smaller the amount, the better the response time of the system (or latency to starting service of a new request) and the smaller the required buffer (or secondary storage staging) space and (b) what is the latency of the system under different workloads and what are the factors that affect latency. In this paper we focus on addressing such questions in the context of tertiary storage systems that store and retrieve *heterogeneous* multimedia objects.

In devising a scheduling technique for servicing real-time requests in this periodic manner, it is important to address the following questions. Firstly, what latency can we expect for requests being serviced when there is a certain load on the system, i.e., how early in the reservation schedule can we find a sequence of slots that can satisfy a particular request. And secondly, what is the resulting cost of the storage subsystem. More specifically, one important tradeoff is between improved bandwidth utilization (on the tape subsystem) and increased buffer space requirements (on the disk subsystem). Since it is not immediately clear how to compare savings in I/O bandwidth with savings in buffer space, one approach is to assess this tradeoff through cost considerations, e.g., in this case a meaningful performance measure is \$/stream.

In this paper, we first introduce some simple techniques for scheduling retrieval of heterogeneous multimedia objects on a tertiary storage system and show that these techniques suffer from poor utilization and unpredictable latency partly due to bandwidth fragmentation. We then present a novel strategy, termed *rounds*, which addresses the issue of bandwidth fragmentation and results in better bandwidth utilization and more predictable latency. We study the relative improvement in cost/performance of *rounds*, as compared to the simpler schemes, using simulation as well as cost and characteristics of existing commercial robotic tape storage libraries.

The remainder of the paper is organized as follows. Section 2 describes the architecture of the system, identifies the resource allocation problem that needs to be solved, and then describes several strategies that can be used to solve this problem. Section 3 presents analysis of the various strategies and their comparison using a set of performance as well as cost/performance metrics. Section 4 gives our concluding remarks.

2 System

In this section we describe the system we are considering, define some issues and problems involved in the design of such systems, and then present several schemes that address these issues.

2.1 Architecture and Terminology

The multimedia storage server considered here consists of a three level storage hierarchy, including main memory, disk storage, and tertiary storage. In this work we concentrate on the tertiary level of the system, from which data is retrieved and staged to the disk subsystem before delivery; as already mentioned, this is motivated by the need to meet real-time constraints. Briefly, the tertiary storage system under study is a robotic tape storage library, which contains a relatively large number of tapes, relatively few drives, and a robotic arm used to move tapes in and out of drives. (More than a single arm is possible, but for ease of exposition we will consider a single arm here, unless otherwise stated.) The entire database is stored on tertiary storage and data is retrieved on-demand. All data stored in the tape library is in units of constant size, termed *pages*.

All requests to the tertiary subsystem are for sets of pages which are randomly distributed among the tapes. We assume (given the large number of tapes and the relatively small number of drives) that each retrieval of a page requires a tape exchange, and more specifically, it requires a robot and drive load, a seek to the required page on the tape, the read itself, a rewind, and a drive and robot unload (refer to [1, 4] for details). The time required for all these operations to complete is termed *cycle time* (C_t).

Each request is represented by a tuple (b, p), where: (1) b is the requested transfer rate, bounded on the low side by a predetermined value, which we will refer to as *minimum bandwidth* (B_{min}), and on the high side by the combined effective capacities of all the tape drives in the system, which we will refer to as *maximum bandwidth* (B_{max}) and (2) p is the number of pages to be transferred at that rate. When a request for a certain transfer rate arrives, the following must be reserved in order to service the request and maintain real-time constraints: (a) *streaming bandwidth* on the tape subsystem for reading the data, (note that, streaming bandwidth is also needed on the disk subsystem to eventually deliver the data to the user, but, as already mentioned, we focus on the tertiary level here), (b) *staging bandwidth* on the disk subsystem for writing the data (later to be delivered from the disk subsystem), and (c) *staging space* on the disk subsystem for caching the staged data. We focus on reservation of tertiary bandwidth first. Each reservation unit is made up of two parts: the reservation of a robot for the length of time it takes to load/unload a tape and the reservation of a drive for the length of time it takes to load, seek, read, unwind, and unload. This set of reservations is termed a *slot*. Slots corresponding to one drive are staggered from another drive by the "robot latency" — if the slots corresponding to the different drives were synchronized, all drives would require the robot to exchange tapes simultaneously (which is not possible with a single robot arm).

The main notation used in this paper is summarized in Table 1, for ease of reference. We will define the various terms in this table, as the need for it arises.

Notation	Definition
D	Number of tape drives
L_t	Robot latency
C_t	Cycle time
B_{max}	Maximum allowed request bandwidth
B_{min}	Minimum allowed request bandwidth
S_t	total stagger (due to contention for the robot arm)
R_l	Round length (number of slots in a round per drive)
R_s	Round size (number of slots in a round)

Table 1: Notation.

2.2 Problem

Given real-time (or continuity) constraints of multimedia data, when a new request arrives to the system, the tertiary subsystem needs to reserve *periodic* tape drive slots for the entire length of that request. The periodicity of slot reservation corresponds to the bandwidth of the request, e.g., a request that requires a bandwidth that is a quarter of the bandwidth of a single tape drive needs to be scheduled one slot out of every four, on a single drive. In devising a scheduling technique for servicing the real-time requests in this periodic manner, it is important to address the following questions. Firstly, what latency can we expect for requests being serviced when there is a certain load on the system, i.e., how early in the reservation schedule can we find a sequence of slots that can satisfy a particular request. And secondly, what is the resulting cost of the storage subsystem. More specifically, as already mentioned, one important tradeoff is between improved bandwidth utilization (on the tape subsystem) and increased buffer space requirements (on the disk subsystem). Since it is not immediately clear how to compare savings in I/O bandwidth with savings in buffer space, one approach is to assess this tradeoff through cost considerations, e.g., in this case a meaningful performance measure is \$/stream. We elaborate on this later in the paper. Below, we first give a (fairly simple) strategy for scheduling of requests for multimedia

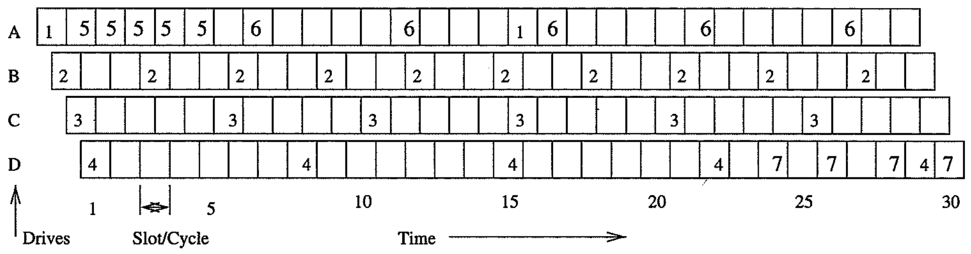


Figure 1: Schedule with *Simple* Scheme.

objects in a (strictly) periodic manner in order to illustrate the basic problem. We term this strategy *simple*.

The *simple* scheduling strategy allocates each request to one drive. The drive that can satisfy the request the earliest is chosen. (Although *simple* is a limited strategy, in a sense that it can not service requests that require more than a single drive worth of bandwidth, we use it in this paper to expose some of the issues and tradeoffs associated with delivery of heterogeneous multimedia objects from tertiary storage.) We illustrate this strategy and the associated problems through an example. Consider a system with four drives, each with an effective bandwidths of 4 MB/s. Let all requests in the system have bandwidth requirements that range from 4 MB/s to 0.25 MB/s. Thus, a request with the maximum bandwidth requirement would need consecutive slots to service it, while a request with the minimum bandwidth requirement would need one slot on one drive every sixteenth cycle. Then, given the following sequence of request arrivals:

- Req. 1: (b, p) = (0.25 MB/s, 2 Pages), i.e., 1 slot in 16
- Req. 2: (b, p) = (1.3 MB/s, 10 Pages), i.e., 1 slot in 3
- Req. 3: (b, p) = (0.8 MB/s, 6 Pages), i.e., 1 slot in 5
- Req. 4: (b, p) = (0.57 MB/s, 5 Pages), i.e., 1 slot in 7
- Req. 5: (b, p) = (4 MB/s, 5 Pages), i.e., all consecutive slots
- Req. 6: (b, p) = (0.8 MB/s, 5 Pages), i.e., 1 slot on 5
- Req. 7: (b, p) = (2 MB/s, 7 Pages), i.e., 1 slot in 2

the corresponding reservation schedule is depicted in Figure 1. Note the periodicity of the various requests and how they correspond to the request bandwidth requirements.

The state of the reservation array in Figure 1 and the resulting schedule of requests illustrate a problem: even thought the array is quite sparse, delays are already becoming considerable. For instance, Req. 7 could be satisfied no earlier than time slot 24. This is due to the fact that, even though the schedule is relatively sparse, it is already fairly fragmented and thus finding a set of empty slots *with a certain periodicity* is "difficult". This problem also makes it more difficult to predict how far into the reservation schedule we would have to look to find the slots that will satisfy a new request, i.e., it is difficult to predict the latency to service a newly arrived request.

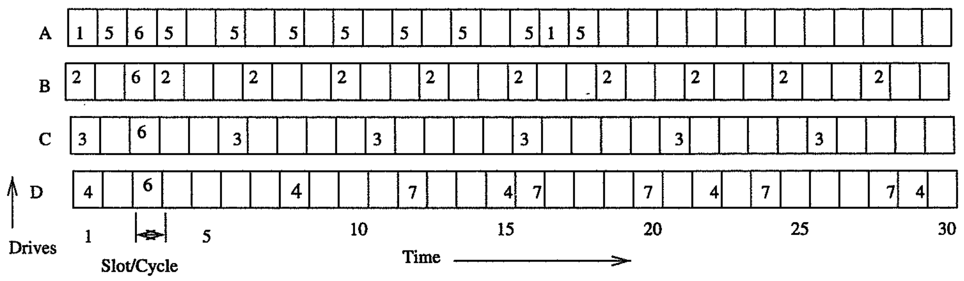


Figure 2: Schedule with *Buffered* Scheme.

One of the causes of the above problem is the restriction that each request be serviced by a single drive. This restriction exists due to the problem of *stagger* caused by robot latency — if the slots corresponding to the different drives were "synchronized", then all drives would require the robot to exchange tapes simultaneously, which is not possible. If we could find a solution to the problem of stagger, we could gain additional flexibility in scheduling requests and consequently attain a better utilization of tertiary bandwidth.

One simple solution to this problem is use of additional buffer space. That is, data from tape drives that start their reads earlier can be buffered on secondary storage until the last tape drive is ready to read. This "synchronization" of slots essentially makes tape drives "interchangeable", allowing the use of different tape drives in retrieving the various pages of a request, while ensuring constant bandwidth allocation to a request. The motivation is that in such a system, requests can be fitted into the reservation schedule earlier (than for instance in a system using *simple*), utilizing some combination of drives rather than just one to satisfy the request. We will refer to this technique as the *buffered* scheme.

More specifically, the slots that correspond to different tape drives are staggered from each other by the robot latency L_r, i.e., the second drive is staggered from the first by L_r, the third from the first by $2 \times L_r$, and so on. Thus, the *total stagger* (S_t) is equal to $\frac{L_r}{C_t} \times \frac{D\times(D-1)}{2}$ cycles (or slots), where D is the number of drives, L_r is the robot latency, and C_t is the cycle time.

We illustrate the *buffered* scheme through another example. Given the following sequence of request arrivals:

- Req. 1: (b,p) = (0.25 MB/s, 2 Pages), i.e., 1 slot in 16 on one dr ive
- Req. 2: (b,p) = (1.3 MB/s, 10 Pages), i.e., 1 slot in 3 on one drive
- Req. 3: (b,p) = (0.8 MB/s, 6 Pages), i.e., 1 slot in 5 on one drive
- Req. 4: (b,p) = (0.57 MB/s, 5 Pages), i.e., 1 slot in 7 on one drive
- Req. 5: (b,p) = (2 MB/s, 9 Pages), i.e., 1 slot in 2 on one drive
- Req. 6: (b,p) = (16 MB/s, 4 Pages), i.e., 1 slot on all four drives
- Req. 7: (b,p) = (1 MB/s, 5 Pages), i.e., 1 slot in 4 on one drive

the corresponding reservation schedule is depicted in Figure 2. Note that the "synchronization" of drives also increases the range of bandwidths supported. While the maximum bandwidth supported under the *simple* scheme was the bandwidth of a *single* tape drive, the maximum bandwidth that can be serviced under the *buffered* scheme has increased to the bandwidth of all the tape drives combined, e.g., Req. 6, which has a bandwidth requirement of 16 MB/s, is serviced by all four drives. However, the buffered scheme does not (completely) eliminate bandwidth fragmentation. Note that, even though the reservation array is fairly sparse, Req. 7 could not be serviced until slot 12.

In summary, although *simple* and *buffered* strategies will satisfy the real-time constraints of multimedia data, one problem that results from such strategies is *bandwidth fragmentation*, partly due to the *heterogeneity* of user requests. This consequently results in less efficient bandwidth utilization and latencies in servicing requests. In order to design a more cost-effective system, we need a data retrieval scheduling technique which:

1. reduces bandwidth fragmentation by "packing" slots better; for instance, if we could relax the rigid periodic allocation of slots to requests, without incurring large buffering costs, we would be able to better utilize tape drive bandwidth

2. disassociates a request from a tape drive — if different tape drives could serve different pages of the same request, we could gain additional flexibility (part of the cost here will be due to tape drive stagger; since slots corresponding to different drives are staggered by some multiple of the robot load/unload latency, requests serviced by multiple drives would require some additional buffer space to guarantee continuity in data delivery).

3. facilitates fairly simple latency prediction, i.e., given the current load, what is the expected latency for starting service of a newly arrived request; this is especially useful in systems where there is a possibility of a user reneging, where the probability of that occuring is a function of the expected latency to start of service.

2.3 The Rounds Approach

In this section, we introduce a strategy termed *rounds* that exhibits the characteristics stated above (the *buffered* technique introduced above only exhibits the second characteristic). It is based on aggregating a set of slots into a round, where each slot is homogeneous in time and space, i.e., within a round one slot is the same as any other, irrespective of where it lies chronologically or to which tape drive it corresponds. All pages read from all slots in a round are staged before streaming begins, and each reservation is for a certain number of slots from a particular round. More precisely:

- A *round* is a set of slots, whose *size* is the ratio between the maximum bandwidth, B_{max}, and the minimum bandwidth, B_{min}, i.e., the number of slots per round, $R_s = B_{max}/B_{min}$. The round *length*, R_l, is the number of slots per round per drive. In our example, where $B_{min} = 0.25$ MB/s and $B_{max} = 4 \times 4$ MB/s, $R_s = 64$ and $R_l = 16$. Intuitively, R_l is the time, in slots, it takes to stream a page at B_{min}. This is illustrated in Figure 3 where each set of highlighted slots constitutes a different round.

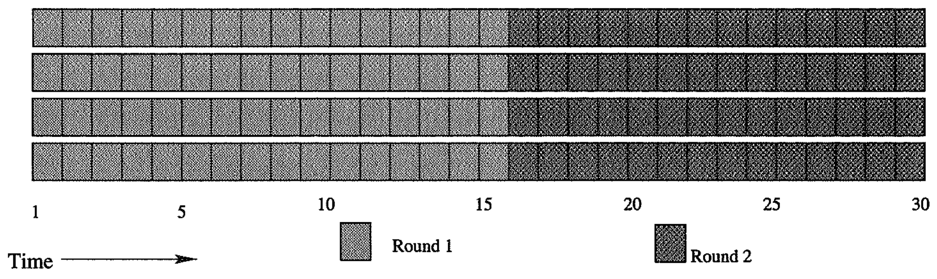


Figure 3: Rounds

- A round is the fundamental unit of reservation. All slots in a round are identical (or *homogeneous*). Reservations are done at the granularity of rounds rather than slots. If a request needs a bandwidth of b MB/s, then $\frac{b}{B_{min}}$ slots are reserved per round for the number of rounds required. Since all slots in a round are identical and scheduling is performed at the level of rounds, slots within a round can be packed without concern for periodicity or for drive association – this results in a more efficient bandwidth utilization of the tertiary subsystem.

- Every page in a round is staged to disks before streaming to the user begins. That is, streaming begins after the last page corresponding to a round has been staged. Thus, the average *latency* is half the round length, i.e., $\frac{R_l}{2}$ cycles, given that there is available capacity in the round.

- The required secondary storage staging space should be sufficient to buffer two rounds worth of data (in pages) plus the stagger (which is due to the contention for the robot arm), i.e., $(2 \times R_s) + (\frac{L_r}{C_t} \times \frac{D\times(D-1)}{2})$ pages (we elaborate on this in Section 3.1).

The *rounds* strategy exhibits the following advantages:

- *Near complete utilization.* As long as there are slots left in a round, i.e., as long as there is unused bandwidth available, and the bandwidth of a request fits (*somewhere* in a round), it will be satisfied. In the earlier strictly periodic model, the data access *pattern* had to fit the *pattern of holes* created by unoccupied slots.

- *More predictable quality of service.* As will be described later, *rounds* exhibits smaller variances in latency to starting service of a new request, which facilitates a more predictable quality of service.

- *Simplicity of reservation.* Slots in a round are completely equivalent, so there is no need to allocate blocks in certain patterns or worry about which drive a slot is associated with. Given the *heterogeneity* of applications expected to utilize a mass storage system, this is an attractive feature.

- *Opportunities for optimization.* Given the architectural differences of various tertiary storage systems, flexibility in the order of block and tape retrieval will provide opportunities for scheduling optimization *within* a round (or customization of scheduling to different architectures, e.g., as in [5]).

However, it also exhibits the following disadvantages:

- *Potentially higher average latency under a small class of workloads.* Because streaming only begins at the end of rounds, there is a potential for higher average latency as compared to a strategy without rounds where streaming can begin as soon as a page is staged. However, our experiments indicate that this occurs *only* under *light* loads, and that in those cases the latency penalty is not significant.
- *Higher secondary storage.* In the earlier strict periodic model, each page was streamed as soon as it finished staging, while in *rounds*, all pages in a round are staged before streaming begins; thus, an additional secondary storage cost is incurred.

These advantages and disadvantages are quantified in the following section. Before we proceed, we would also like to mention that other variations on the three schemes given here are possible; however, we do not discuss them here as one of our main goals is to illustrate the issues and tradeoffs involved in the retrieval of heterogeneous multimedia objects from tertiary storage, using simple strategies.

3 Analysis

To determine the usefulness of the rounds technique, we evaluate its performance and cost (which includes both tertiary storage system cost as well as secondary storage system cost needed for staging of data from tertiary storage and streaming it to users) and compare it to the two other strategies, namely *simple* and *buffered.* We study the performance of all three strategies using simulation. (In our evaluation, we consider existing robotic tape storage technologies, such as Ampex, Exabyte, etc.) Specifically, we consider their behavior using the following performance metrics: (a) *latency*, time to start service of new requests, (b) *misses*, the fraction of requests rejected (based on a maximum latency system requirement, i.e., requests that must wait for more than a prespecified maximum amount of time are rejected), and (c) *cost/stream*, which includes the cost of the robotic tape storage library and the disk subsystem needed to support a specified level of performance and quality of service. (It is simple to show that the maximum amount of secondary storage is required when all requests are for the lowest bandwidth available; thus, in our evaluation, we use this worst case possibility to compute the secondary storage cost.)

3.1 Secondary Storage Usage

We begin our analysis by examining the secondary storage requirements of the three strategies described in Section 2, since secondary storage usage constitutes one of the main differences between these techniques. More specifically, secondary storage is required by the different strategies for three purposes: staging, streaming, stagger-smoothing. All three strategies require space for staging and streaming while only *buffered* and *rounds* require space for stagger-smoothing, where the space required to compensate for the stagger is a function of the number of drives, as described earlier, and is equal to $(\frac{L_r}{C_t} \times \frac{D\times(D-1)}{2})$.

Furthermore, each tape drive requires buffer space to write the page that it is currently reading. This is the required staging space and is equal to D pages. Once a page has been staged to secondary storage, it will be resident there until it has been streamed (to the user). The length of time it is resident is a function of the required streaming rate — the slower the streaming rate, the longer it will reside on secondary storage; therefore, the peak demand for streaming buffer space occurs when all requests are for the minimum bandwidth. For instance, in our earlier example where $B_{max} = 16$ MB/s and $B_{min} = 0.25$ MB/s, the maximum demand for streaming buffer space occurs when 64 requests for 0.25 MB/s are being serviced simultaneously, where each page, which is staged at 4 MB/s, remains in secondary storage for 16 cycles. Contrast this to a situation where 4 requests, each with a 4 MB/s bandwidth requirement, are being serviced where each page is resident in secondary storage for only one cycle while being streamed. (In both cases secondary storage space can be reclaimed earlier, but for simplicity of exposition, we assume that space is not reclaimed until the whole page has been streamed.) We thus use the worst case scenario, where all requests are for the lowest bandwidth, to calculate the peak demand for streaming buffer space for each scheme. Hence, the amount of storage required varies according to the strategy used and is given below:

- **Simple**: in the worst case, each page takes a cycle to be staged and R_l cycles to be streamed. Thus, each page remains in secondary storage $R_l + 1$ cycles; furthermore, each set of D such pages are offset by one slot. Thus, at any point in time there are D pages being staged, and $D \times R_l$ pages being streamed making the peak total demand $R_s + D$ pages.

- **Buffered**:

  the worst case is the same as in the *simple* strategy; thus, $D \times (R_l + 1)$ pages are required for staging and streaming. In addition, secondary storage is required to account for the stagger smoothing, as described in Section 2.2, where the total stagger was given as $\frac{L_r}{C_t} \times \frac{D \times (D-1)}{2}$ cycles.

- **Rounds**: a page takes a cycle to be staged; it is then resident in secondary storage until the end of the round and is then streamed in R_l cycles. The peak secondary storage demand occurs during the last cycle of each round when R_s pages from the previous round are still being streamed, while the last D of the current round's pages are being staged, giving a total of $2 \times R_s$ pages. In addition, secondary storage is required to account for the stagger smoothing, as described in Section 2.2, where the total stagger was given as $\frac{L_r}{C_t} \times \frac{D \times (D-1)}{2}$ cycles.

3.2 Simulation Results

In our simulation data requests consisting of a tuple (p, r), where p is the request size in number of pages and r is the periodicity (i.e., to service a request with periodicity r, we must retrieve a page for that request every r^{th} slot of a single tape drive), are generated using a Poisson process where the inter-arrival times are exponentially distributed (the mean will be specified on per-graph basis, as needed) and p and r are uniformly distributed. (Note that, since *simple* is not able to service requests with bandwidth requirements exceeding the bandwidth capacity of a single drive, to make the comparison between the three schemes fair, we limit the experiments presented in this section to such request types. This is not

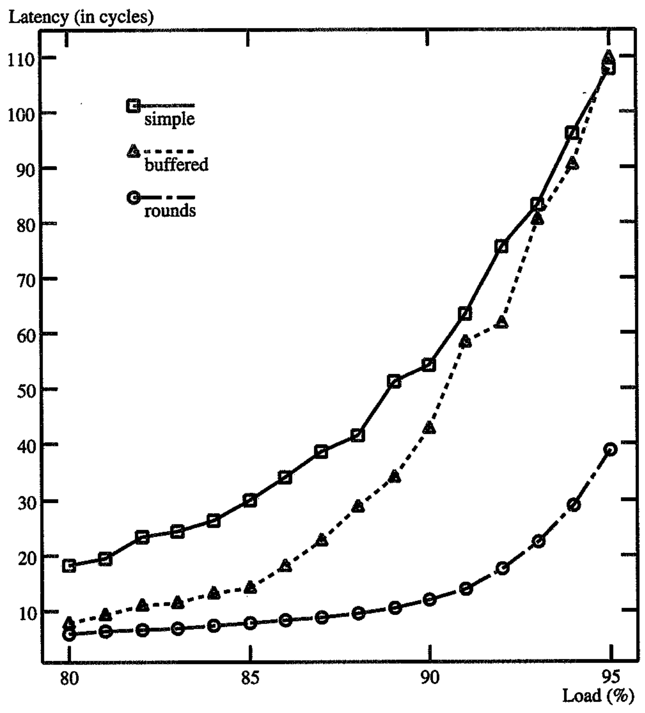


Figure 4: Latency.

limiting, in the sense that it still allows us to illustrate the basic issues and tradeoffs involved in retrieval and delivery of heterogeneous multimedia objects from tertiary storage.)

The simulation system then attempts to fit each request into the reservation schedule as early as possible using one of the strategies described in Section 2. If a fit cannot be found within a *specified maximum latency* (which is a design parameter), a "miss" is recorded. Unless stated otherwise, in the following analysis, we use a system with: (a) 4 drives, (b) request sizes that are uniformly distributed between 1 and 15, (c) periodicities that are uniformly distributed between 1 and 5, and (d) specified maximum latency of 400 cycles.

We first investigate the performance of the different schemes using the *latency* and *miss* metrics — Figures 4 and 5 depict latency and misses, respectively, as a function of system load for the three strategies described earlier. As expected, at light workloads, all three strategies exhibit similar latency and miss percentages. For moderate to high workloads, the inefficiencies of *simple* and *buffered* strategies result in much higher latencies and miss percentages, as compared to *rounds*. That is, *rounds* exhibits a more graceful performance degradation as the load increases, i.e., *rounds* can handle higher loads before system saturation occurs — this is due to lower tertiary bandwidth fragmentation which results in better tertiary bandwidth utilization. Note, of course, that the difference in miss percentages is more significant under a relatively smaller specified maximum latency (refer to Figure 5).

We next investigate the effect of *number of drives* and *request sizes* on system performance, using *latency* as the performance metric — Figures 6 and 7 depict system latency as a function of mean request size and number of tape drives, respectively, for all three strategies (that is, a mean request size of x means that the request sizes are uniformly distributed between 1 and y, where $x = 1 + \frac{y-1}{2}$).

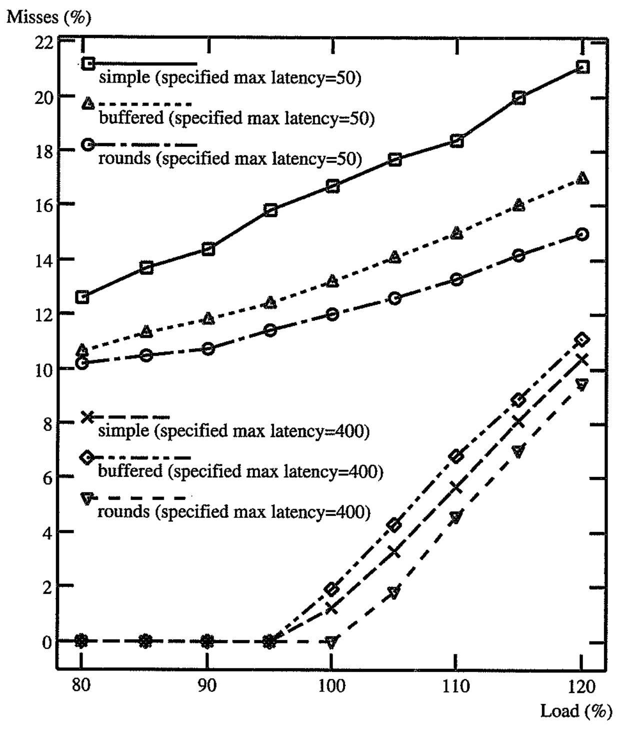


Figure 5: Misses.

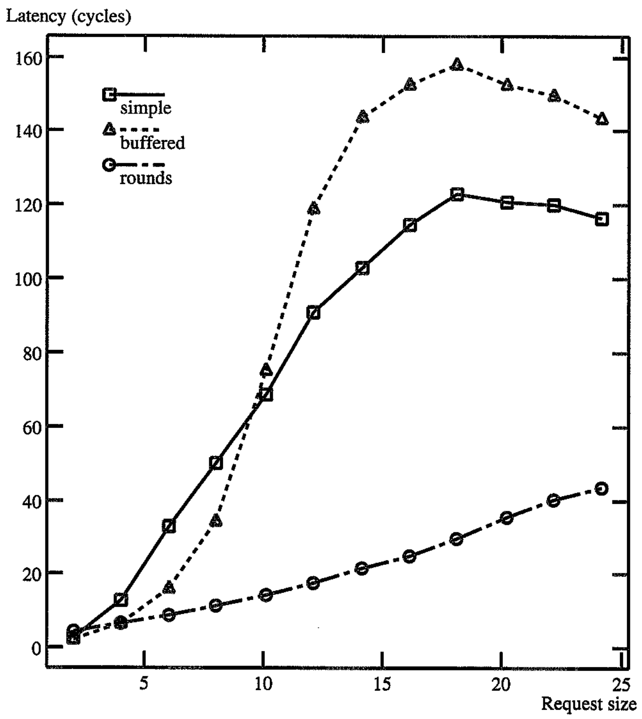


Figure 6: Effect of request sizes on latency (90% load).

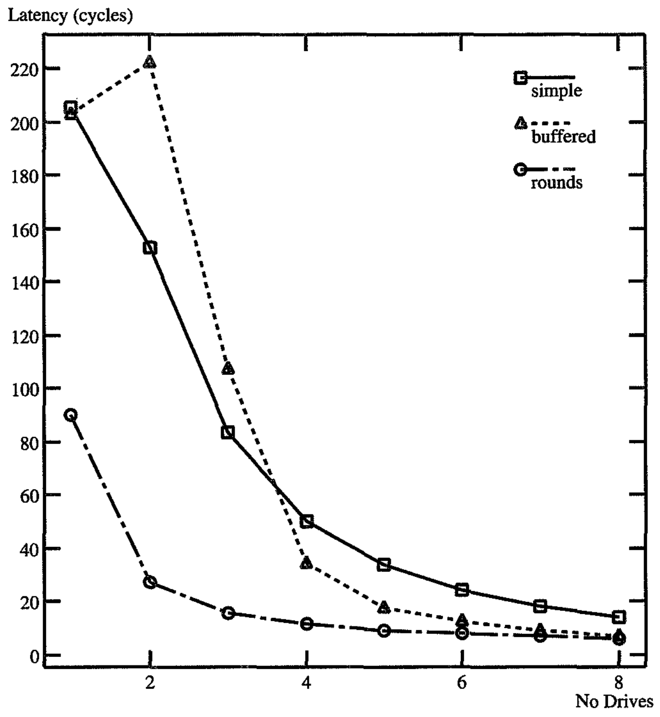


Figure 7: Effect of number of drives on latency (90% load).

As the mean request size increases, latency increases as well (partly due to the fact that it's more difficult to fit longer requests into a retrieval schedule), although again, *rounds* exhibits a more graceful degradation. Note that, beyond a certain mean request size, latency can start to decrease again (as in the *buffered* and *simple* curves here); this is partly due to the fact that, as the mean request size increases, so does the corresponding miss percentage, which effectively begins to lower the "actual" load on the system (since some requests do not get serviced due to misses). On the other hand, latency decreases as the number of drives in the system increases. This is due to the fact that with more drives, we have greater "flexibility", and thus the fragmentation of tertiary bandwidth has a lesser effect on latency. This is also evident by the fact that *rounds* performs significantly better than the other two schemes in systems with fewer drives (i.e., the better bandwidth utilization of *rounds* is of a (relatively) lesser importance in systems with many drives). In summary, these figures demonstrate that, as request sizes grow larger and the number of tape drives in the system decreases, *rounds* begins to dramatically outperform the other two strategies.

Finally, we examine the effect of *round length* on system performance; specifically we use *latency* as the performance metric — Figures 8 and 9 illustrate the effect of round length on the performance of the *rounds* strategy (in varying round length we assume that we have flexibility to vary B_{min}; otherwise, round length would be fixed, as described in Section 2).

Intuitively, the longer the round, the more efficiently we can use tertiary bandwidth, but the potentially higher the latency. Figure 9 illustrates that up to a certain point, as the request sizes increase, larger round lengths become more advantageous; however, after a certain point, it is difficult to compensate, with better bandwidth utilization, for the fraction

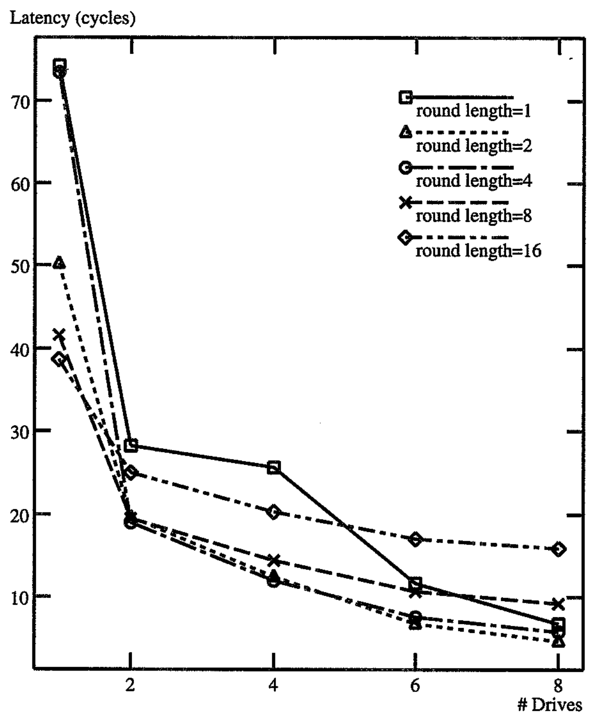


Figure 8: Effect of round length on latency (as a fnc of number of drives at 90% load).

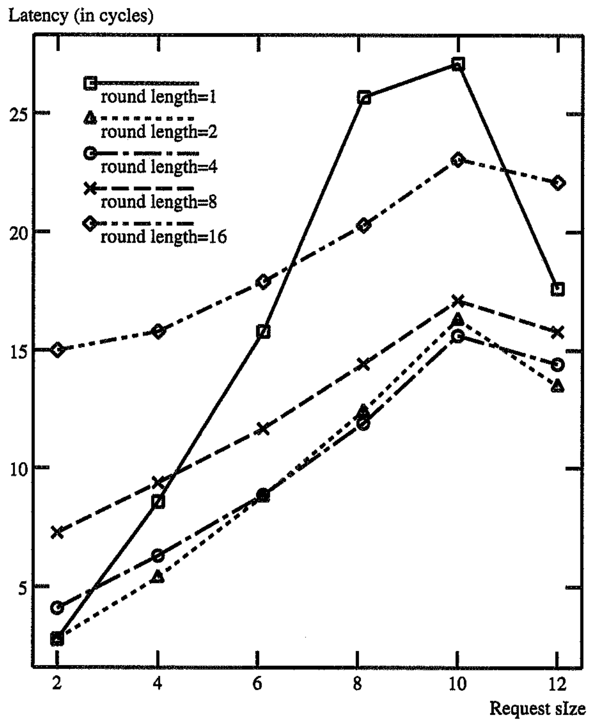


Figure 9: Effect of round length on latency (as a fnc of request size at 90% load).

of latency caused by a long round. Similarly, Figure 8 illustrates that larger round length can compensate for lack of drives, but only up to a certain point.

3.3 Cost-based Comparison of Schemes

Since rounds achieves its better performance (specifically, better utilization of tertiary bandwidth) by utilizing more secondary storage, we need a metric that will allow us to make a comparison between improved bandwidth utilization (on the tape subsystem) and increased storage space requirements (on the disk subsystem). As already mentioned, since it is not immediately clear how to compare savings in I/O bandwidth with savings in storage space, one approach is to assess this tradeoff through cost considerations — in this case a meaningful performance measure is $/stream (we will calculate this by computing the maximum bandwidth that a system can support and the corresponding cost of that system). Specifically, we will use this metric to compare the three strategies at points where they provide similar performance.

Capacity (GB)	RPM	Cost($)	Cost/GB
1	5200	600	600
2	5200	800	400
4	5200	1,400	360
9	7800	2,300	255
23	7800	6,000	255

Table 2: Secondary storage.

Characteristic	ACL 4/52	ACL 6/176	Ampex	Exabyte AME
Max. number of Drives	4	6	4	2
Number of robots	1	1	1	1
Robot Latency (s)	20	20	6	10
Drive Latency (s)	190	190	28	185
Total Latency (s)	210	210	34	195
Max. tertiary bandwidth (MB/s)	20	30	60	6
Tertiary capacity (GB)	1,820	6,020	6,400	400

Table 3: Tape libraries.

Tables 2 and 3 give characteristics and price information for some typical disk drives and robotic tape storage libraries (these are *list* prices which were compiled from quotes, web-sites, and catalogues during the Spring of 1997). For each system, related characteristics, such as cycle length, secondary bandwidth required, stagger, and so on are listed in

Characteristic	ACL 4/52	ACL 6/176	Ampex	Exabyte AME
Read time	243	243	106	310
Secondary Devices Needed	3	5	9	1
Cycle time	453	453	140	505
Cycle size (MB)	1,215	1,215	1,590	930
Round length	4	4	4	4
Round size (max)	16	24	16	8
Tertiary efficiency	0.54	0.54	0.76	0.6
Eff. tertiary BW (w/ 1 drive) (MB/s)	2.7	2.7	11.4	1.8
Eff. tertiary BW (w/ max drives) (MB/s)	10.8	16.3	45.6	3.6
Total Stagger (GB)	0.6	1.5	0.54	0.03
Total Stagger (in cycles)	0.26	0.66	0.26	0.02
Cost of Library (w/ 1 drive)	24,000	62,000	280,000	19,375
Cost of Library (w/ max drives)	57,000	117,000	610,000	26,000

Table 4: Tape library characteristics.

Table 4. (Cycle time is chosen such that the page size is "reasonably" large, i.e., results in fairly high tertiary bandwidth efficiency, while incurring "reasonably" low secondary storage costs. The number of secondary storage devices needed is chosen such that the secondary storage system bandwidth that is needed for staging matches the bandwidth of the tertiary system.)

The secondary storage required and the effective tertiary bandwidth of each system under the different strategies is given in Table 5, for the *simple*, *buffered*, and *rounds* schemes, respectively. To compute the "effective" bandwidth of each system, an "operating load" is chosen for each strategy by noting the load at which some performance metric remains below a constant threshold value. For example, using latency as the metric, 25 cycles as the threshold value, and looking up the load supported in Figure 4 (where the mean request size is 8, mean periodicity is 3, and 4 drives are used), we get operating points of 0.8 for the *simple* strategy, 0.87 for the *buffered* strategy and 0.93 for *rounds*. This is the fraction of the theoretical maximum bandwidth the system can maintain under the given constraints for the parameters chosen. (Refer to Section 3.1 for computation of secondary storage requirements.)

The resulting unit stream costs of the three strategies for the chosen operating points are listed in Tables 6 and 7, for the different systems used. The first operating point corresponds to a system with 1 drive and a mean request size of 18, where the desired performance metric (that is held constant among the three schemes) is a mean latency of 40 cycles. (The $\$/stream$ results for this operating point are given in Table 6.) The second operating point corresponds to a system with a mean request size of 8 and the maximum possible number of drives (i.e., 4, 6, 4, and 2 drives for the ATL4, ATL6, Ampex, and Exabyte AME systems, respectively). The desired performance metrics (that is held constant among the

	ACL 4/52			ACL 6/176			Ampex			Exabyte AME		
Staging (Cycles)	20	20	30	30	30	48	20	20	32	10	10	16
Stagger (Cycles)	0	0.26	0.26	0	0.66	0.66	0	0.26	0.26	0	0.0	0.0
Disk Space GB (1 drv)	24.3	24.3	26.7	36.5	36.5	55.9	31.8	31.8	49.8	9.3	9.3	15
Disk Space GB (max)	24.3	24.6	27	36.5	37.3	56.7	31.8	32.3	51.3	9.3	9.3	15
Disk Cost k$ (1 drv)	7.2	7.2	7.8	10.9	10.9	15.5	11.8	11.8	16.2	3.0	3.0	4.3
Disk Cost k$ (max)	7.2	7.3	7.9	10.9	11.0	15.6	11.8	11.9	16.3	3.0	3.0	4.3
Effective Bw. MB/s (1 drv)	3.0	3.0	6.4	4.6	4.6	9.6	12.8	12.8	26.9	1.0	1.0	2.1
Effective Bw. MB/s (max)	8.6	9.4	10	14.7	15.2	15.7	36.5	39.7	42.4	2.3	2.7	3.4

Table 5: Effective bandwidth and secondary storage requirements.

three schemes) in this case is a mean latency of 25 cycles, and the corresponding "operating load" is computed for each scheme and each tape library as described in the example above. (The $\$/stream$ results for this operating point are given in Table 7.) The costs in all cases are computed as the sum of tertiary and secondary storage costs, as described in Tables 4 and 5 as well as Section 3.1.

At the first operating point, rounds consistently shows an approximate *factor of two* improvement over the other two strategies, while at the second operating point, it varies from an $\approx 0.2\%$ degradation, as compared to *buffered* for the ACL 6/176 library, to a $\approx 40\%$ improvement, as compared to *simple* for the Exabyte AME library. Thus, depending on the required operating point, different schemes may be advantageous.

	ACL 4/52	ACL 6/176	Ampex	Exabyte AME
Simple Strategy ($/MB)	$8,000$	$13,480$	$21,870$	$19,375$
Buffered Strategy ($/MB)	$8,000$	$13,480$	$21,870$	$19,375$
Rounds ($/MB)	$3,350$	$6,460$	$10,409$	$9,226$

Table 6: Unit stream cost (operating point 1).

	ACL 4/52	ACL 6/176	Ampex	Exabyte AME
Simple Strategy ($/MB)	$7,435$	$8,699$	$17,044$	$12,590$
Buffered Strategy ($/MB)	$6,841$	$8,425$	$15,664$	$10,725$
Rounds ($/MB)	$6,460$	$8,443$	$14,768$	$9,032$

Table 7: Unit stream cost (operating point 2).

4 Conclusions

In this paper we studied retrieval of *heterogeneous* multimedia objects from a tertiary storage system. We proposed the *rounds* retrieval scheme and compared it to two simpler strategies, namely *simple* and *buffered*. Briefly, some of the main results of our study are as follows (that is, we give a summary of all experiments that we have conducted thus far — most of these results are illustrated through the figures and tables given in Section 3; however, due to lack of space, we were not able to include all of these results in that section):

- *Latency.* For light workloads, rounds exhibits a slightly higher mean latency than the buffered strategy; however, for moderate to high workloads, rounds results in a significantly lower mean latency than the other two techniques, which is due to the fact that it is able to utilize bandwidth better — this becomes more pronounced as the load on the system grows. Furthermore, the variance in latency exhibited by the rounds strategy is smaller than that of the other two strategies, which facilitates a more predictable quality of service.

- *Misses. Rounds* consistently outperforms the other two strategies under this performance metric, again due to its more efficient bandwidth utilization. The differences become more apparent as the maximum allowed latency becomes more stringent and as the number of drives in the system decreases, i.e., either as the quality of service requirements become more stringent or as the amount of resources (which experience high contention) decreases.

- *Round length.* We also experimented with "proper" round lengths (since this is one of the system design parameters). As is probably expected, very small rounds become inefficient at higher workloads, and very large rounds incur too high of a secondary storage penalty (as well as contribute to higher latencies) while only marginally improving bandwidth utilization.

- *Efficiency.* The rounds strategy is more efficient at utilizing available resources (specifically, tertiary storage bandwidth). Given the same architectural configuration, a system using *rounds* is able to maintain relatively low latencies at higher throughputs, i.e., a system using *buffered* (or *simple*) approach can become unstable and result in high latencies at significantly lower levels of workload than a system using *rounds*. Our experiments indicate more than a 100% improvement in throughput for systems with fewer drives and larger request sizes, i.e., systems with high contention for resources and high workloads. Thus, *rounds* is a more attractive scheme for systems with wider ranges of workloads.

- *Cost ($/MB/s).* In order to make this a more fair experiment, we compared the cost characteristics of the three strategies at points where they provide the same level of performance. Our experiments indicate that, depending on the architectural configurations of the tertiary libraries and the offered workloads, the cost/stream of *rounds* can vary anywhere from a $\approx$ 3% degradation to a *factor of two* improvement, as compared to *buffered* (we do not give the comparison with *simple*, as the difference is even greater). Conversely, depending on the architecrual configuration and the offered workload, for an additional cost of $\approx 3 - 5\%$, *rounds* can reduce latency by nearly a *factor of five* (because *rounds* is significantly more efficient at processing workloads, it is not always simple to make the comparison by maintaining the same level of performance). These improvements are achieved without (possible) scheduling optimizations within a round and with the worst case assumptions for the secondary storage requirements (see above).

In summary, our experiments indicate that in most cases *rounds* outperforms *buffered* and *simple*, often resulting in significant improvements. However, as is usually the case, one retrieval scheme is not absolutely better than another, but rather one must understand the issues and tradeoffs involved and choose a scheme accordingly, where the appropriate scheme is, to a large extent, a function of the architectral configuration used, the expected system workload (which is a function of the mix of applications expected to use the system), and the desired operating point.

Acknowledgements

This work was supported in part by the NSF CAREER grant CCR-96-25013. This work was partially done while Leana Golubchik was with the Department of Computer Science at Columbia University.

References

[1] A. L. Chervenak. Tertiary Storage: An Evaluation of New Applications. *Ph.D. Thesis, UC Berkeley*, 1994.

[2] A. L. Drapeau and R. Katz. Striped Tape Arrays. In *Proc. of the 12th IEEE Symposium on Mass Storage Systems*, pages 257–265, Monterey, California, April 1993.

[3] L. Golubchik and S. Marcus. On Multilevel Multimedia Storage Systems. In *Proceedings of the 2nd Intl. Workshop on Multimedia Information Systems*, September 1996.

[4] L. Golubchik, R. R. Muntz, and R. W. Watson. Analysis of Striping Techniques in Robotic Storage Libraries. *Proc. of the 14th IEEE Symposium on Mass Storage Systems*, pages 225–238, September 1995.

[5] B. Hillyer and A. Silberschatz. On the modeling and performance characteristics of a serpentine tape drive. In *Proceedings of Sigmetrics*, pages 170–179, 1996.

[6] B. Hillyer and A. Silberschatz. Storage Technology: Issues and Trends. June 1996.

[7] S. W. Lau, J. C. S. Lui, and P. C. Wong. A Cost-effective Near-line Storage Server for Multimedia System. In *Proceedings of the 11th International Conference on Data Engineering*, March 1995.

Page intentionally left blank

SLEDs: Storage Latency Estimation Descriptors

Rodney Van Meter
Information Sciences Institute
University of Southern California
Marina del Rey, CA 90292
rdv@ISI.Edu
tel: +1-310-822-1511
fax: +1-310-823-6714

Abstract

Managing the latency of storage systems is a key to creating effective very large scale information systems, such as web interfaces to satellite image databases and video-on-demand servers. Storage Latency Estimation Descriptors (SLEDs) are architecture-independent descriptions of the retrieval time of a unit of data. They describe the latency to the first byte, and the bandwidth expected. SLEDs are an important enabling technology for true end-to-end quality of service (QoS) because they can be used to predict and schedule data transfer with multimedia (guaranteed I/O rate) file systems. SLEDs provide the interface that allows database management systems (DBMS) and clients of hierarchical storage management (HSM) systems to optimize their data access patterns by choosing to read data in specific sequences or not at all. SLEDs are designed to work in intra-machine, local-area network (LAN) and wide-area network (WAN) storage systems, and to scale through twelve orders of magnitude in latency, from thousands of seconds down to nanoseconds.

1 Introduction

In this paper, we propose exposing certain aspects of storage state to allow applications to make their own determinations about storage access timing and ordering based on the current state of the data. This appears to be especially promising for use in hierarchical storage management (HSM) systems, where the cost to retrieve data can be extremely high. Our approach provides storage system state information via an abstraction known as Storage Latency Estimation Descriptors (SLEDs).

SLEDs will enable applications cooperating with storage systems to both predict their performance and improve that performance by reducing the I/O load they impose on the system. The potential reduction in I/O load comes primarily from applications electing not to perform certain I/O operations, and secondarily from reduction in thrashing due to improved cache utilization by I/O reordering.

Hints in file systems have been the topic of much recent research. Hints send information from the application to the storage system to improve prefetching and caching. SLEDs invert this process, sending information from the storage system to the application to allow applications to make intelligent, informed decisions.

A key premise of SLEDs is that CPU time is cheap, but I/O operations are expensive. During the roughly ten milliseconds required to execute a magnetic disk operation, a 100 MIPS CPU (modest by today's standards) can execute a million instructions, and current technology trends indicate that CPUs will continue to to improve faster than disk drives. Thus, spending up to a million instructions to reduce the number of I/Os by one may reduce the application running time.

In hierarchical storage management systems the access time for a given page can vary from under a microsecond for RAM-cached data to milliseconds for data on hard disk, tens of seconds for data on magneto-optical disks in an autochanger or hundreds to thousands of seconds for offline material stored in manually mounted tapes. This is a span of roughly twelve orders of magnitude. SLEDs provide methods for improving the utilization of such systems, with potential performance gains of several orders of magnitude while reducing overall system load.

SLEDs will be useful when browsing datasets kept in HSM systems – for example, large satellite image databases. Although web browsers can predict completion time as data arrives by knowing the size and transfer rate once data begins arriving, the latency to the first byte of data is usually unknown when the data is actually stored in an HSM system. Even the web server may be unable to determine how long retrieval will take. SLEDs provide the interface that allows the storage system, web server and browser to cooperate to provide the end user the information required for informed, time-efficient database browsing.

SLEDs are expected to have a broad impact on data-driven information services, including networked databases, digital libraries and video-on-demand services. Without the performance improvements and predictions SLEDs can provide, successfully deploying such services will require ad hoc solutions to these problems.

The rest of the paper is organized as follows: section 2 presents the basic technical details of SLEDs, and section 3 presents some uses. Section 4 describes the heterogeneous storage environments SLEDs support and section 5 describes implementation generations. Section 6 presents an alternative I/O paradigm built on SLEDs. Sections 7 and 8 are related and future work. Sections 9 and 10 describe ISI's qualifications to conduct such research, and a research plan should the project be funded.

2 SLEDs

SLEDs are an exposure of the file's current storage state. Using a system call, an application can retrieve information about the locations of a file's data blocks. Note that if the metadata is not memory-resident, this call may itself result in I/O being performed.

SLEDs provide:

- An open, consistent interface for storage access optimization, regardless of storage technology or location.

- Information flow about storage state from the system to applications; when combined with hints and reservations, information flows in both directions across the system/application boundary.

- The necessary "next step" enabling technology to bring predictable performance to local, network and wide-area storage systems, supporting real-time applications and quality of service (QoS).
- A "future proof" substrate for I/O programming, because they are technology independent.
- Resource utilization improvements by allowing applications to actively participate in I/O scheduling and pruning.

The metadata is returned as an extent map with two key pieces of timing information, the expected latency and throughput. In addition, SLEDs may include an indication of the reliability level of the latency and bandwidth numbers, which should be high for local disk and low for distributed file systems, and a system state change function, which will be discussed later. While it is possible to return more complex information, such as the device ID, block address, device type, bus (or network) attachment, mount status, head position, etc., the two time estimates should be adequate for many purposes, and provide a simple, device-independent approach.

The latency includes rough estimates of the time to retrieve data from tape, when necessary. This must include some estimate of the wait time for a tape drive, robot handler, tape load and seek. This information is clearly both very dynamic and difficult to estimate accurately, and the representation of this data is an open area of research.

Key difficulties in representation include characterizing the effects of a given operation on the system state. However, this must be taken into account for the system to be complete. Given complete freedom to schedule requests, finding the optimal schedule is combinatorially prohibitive, so heuristics will be used.

SLEDs describe extents, so an entire file or dataset can consist of disjoint segments stored in various places. SLEDs are independent of whether the data is stored locally in cache, on hard disk or tape, or remotely in distributed file systems. SLEDs are "future proof" in that they describe time-to-data in an abstract fashion, not tied to the concepts of sequential or rotating media (tape or disk) or networks. Thus, code written once to work within the SLED paradigm will never become burdensome legacy code.

The current three major access effect types are true random (RAM), rotating with seek (disk), and sequential (tape). Unusual, difficult-to-characterize subsystems do exist (serpentine tapes such as DLT exhibit bizarre seek-time patterns; autochangers with rotating drums and handlers may be even more complex) and other unforeseen major types may develop (two or three dimensional positioning for holographic or micromechanical storage?). Because SLEDs are technology-independent, only a new "state change function" for a given technology must be provided in order to explore the impact of different access patterns.

3 Using SLEDs

The interaction between SLEDs and applications falls into four categories: (a) applications with flexible I/O ordering that can use SLEDs to schedule I/O in arbitrary order, (b) applications that will use SLEDs to make decisions about which I/Os to perform, "pruning" the set of I/Os actually executed in the interest of completion time or cost (for those systems that charge for I/O), (c) applications that use SLEDs just to predict performance, and (d) fixed-order algorithms with no need for prediction, which will be unable to use them effectively.

SLEDs support application-controlled access patterns. They require recoding of applications in order to realize the benefits. Applications must be willing to be flexible about the order of file requests. Many database-like programs, where the order of record execution often is inconsequential, are expected to make good use of SLEDs. This will allow better use of data currently in cache, reducing the thrashing that may otherwise occur.

The Unix `find` utility is an excellent example of a utility which will benefit from being adapted to use SLEDs by being able to "prune" its I/O request tree. `find` traverses a directory tree, looking for files with specific characteristics and performing some operation on them, such as forking off a `grep`. `find` currently provides a switch which instructs it not to follow links which will lead it into NFS-mounted file systems, because doing so can very adversely affect the performance not only of the `find` but potentially all systems connected to the same network. In HSM systems providing automatic file migration, the same can be true; the negative impact of imposing load on the HSM system is significant. If SLED-aware `find` is instructed not to access any file with a latency of more than, for example, 100 milliseconds, the find will complete more quickly and with less overall system load. As the object of `find` is often to locate one or a small set of particular files, if they have been recently used, the HSM system likely has them on low-latency storage, meaning that the SLED-unaware `find`'s approach of reading all files will be not only slow and expensive but pointless; SLED-aware find would be faster and cheaper.

For applications unable to reorder or reduce their I/O requests, SLEDs will provide only a means for estimating the performance that can be achieved, a useful feature for admissions control in real-time environments and evaluation of potential execution in multimedia file systems [25, 3].

As mentioned above, applications such as web browsers may find SLEDs' performance prediction useful, in order to let the user know how long a particular request might take, possibly giving the user the option to cancel requests for which he is unwilling to wait. Servers can also utilize performance predictions to manage movement of data among levels of a hierarchy for HSM-based video-on-demand environments [20, 10]. When combined with the Internet's ReSerVation Protocol (RSVP) at the application layer, true end-to-end quality of service guarantees can be achieved.

4 SLEDs in Different Storage Environments

In this section we briefly describe the key storage environments in which SLEDs are expected to be used: local file systems, HSM systems and distributed file systems. Although all of these provide similar programming interfaces, their performance characteristics are very different. The goal of SLEDs is to effectively manage this heterogeneity.

4.1 Local File Systems

SLEDs will find their first application in local on-disk file systems. The first level of information is knowing what is in the file system cache, and what must be fetched from disk. A second-generation SLEDs implementation will understand the page replacement algorithm so that varying request sequences can be explored. Third-generation models will incorporate physical characteristics such as head position (seek) times and zoning [27, 34, 31], while in the distant future, it could ultimately become important to model the rotational position of a drive. SLEDs can ultimately support such functionality without a change in approach.

4.2 Hierarchical Storage Management

It is in hierarchical storage management (HSM) systems that SLEDs have the potential to be most effective. In HSM systems, the access time for a given data block can vary by twelve orders of magnitude. In addition, the high-latency devices such as tape drives operate on single requests, so their transaction rate is very low. Thus, reducing the I/O load on these devices has the potential to improve the performance of not only a given application, but the entire system.

HSM systems apply heuristics to improve execution schedules, but make no attempt to involve the application in such decisions. They do not offer a portable, technology-independent interface that involves applications in the management of data movement, preferring instead to hide such operations, sacrificing system performance to a simpler, more familiar API. This transparency is a strength of HSM, but many applications will want to have the enhanced interface SLEDs provide.

SLEDs initially allow applications to prune their requests, eliminating unnecessary accesses to offline data. This can be especially useful in searching applications, where a match in online data may result in termination of the program before any requests to tertiary devices have become necessary. It is here that SLEDs offer the largest potential gains, with three orders of magnitude or more improvement possible.

In HSM systems, optimizing the access patterns for tape drives is closely tied to the ability to ability to predict their performance [15, 13, 16]. SLEDs will incorporate such information, allowing applications to explore different request sequences.

When processing datasets that exceed the size of hard disk cache available, successive passes across the data are likely to find the cache in different states. By choosing to process readily-available data first, the total number of requests from tertiary storage (and, roughly, execution time) required to scan the entire data set can be reduced by $1/N$, where N is the fraction of the data set that will fit in cache.

As in local file systems, the first class of information provided is an indication of the current resident level of desired information. With just this simple information, the pruning suggested above is possible. The second class of information provided will incorporate some knowledge of cache replacement as data from tape is brought into disk caches. The third generation of SLED will incorporate mechanical knowledge of the complex devices, such as autochangers and tape drives, involved. This will allow direct estimation of the completion time for specific request sequences.

The High Performance Storage System [32] provides an interface for hints, but it is currently unimplemented; SLEDs could be incorporated as well. Systems based on the Open Storage Systems Interconnect model [17] provide *virtual stores*, and SLEDs can be used to model their characteristics.

Menon and Treiber have asserted that programmers will be unwilling to optimize their code to improve access patterns and performance in HSM systems, likening the difference between tape and disk to the difference between memory and disk [19]. However, the gap between memory and tape is larger, and therefore, more likely to produce significant performance improvements for modest programming effort.

4.3 Distributed File Systems

Applying SLEDs to distributed file systems is an interesting problem, due to the cooperation required between client and server. Ideally, both the client and the server will support SLEDs, in which case the client can request the SLED for a particular data segment, and adjust for the network performance. When the client supports SLEDs but the server does not, the SLED for data not locally cached can indicate no better than "remote", or utilize past history to generate an estimate.

SLEDs for distributed file systems will require real-time network protocols and techniques such as RSVP to create a complete end-to-end quality of service. This will create, effectively, NFS with the guaranteed I/O rates now provided by multimedia file systems [25, 3, 29].

In a distributed environment, clients have less knowledge and control of activity at the server, and unpredictable network traffic can interfere. Thus, the SLEDs in this case will indicate higher potential variance in performance.

5 Evolving SLEDs

SLEDs do not have to be perfect in order to be useful; over time, the accuracy of the SLEDs representations for specific devices and systems will gradually improve, as will the heuristics for choosing schedules. Thus, we expect the overall SLEDs concept to pass through several generations.

The details of the incorporation of change in overall system state due to requests is one of the most open areas in SLEDs. The earliest SLEDs will simply represent levels of the hierarchy, with no indication of changing state. Second-generation SLEDs will represent changes in the system state based on speculative execution of specific sequences, primarily by understanding the cache and page replacement algorithms. Third-generation SLEDs will incorporate physical knowledge of device performance, including tape motion, robotic autochanger motion and disk head seeks.

This system state change representation may have to take the form of executable code associated with a specific SLED, and make modifications to a memory-resident state diagram. The exact form for this is yet to be determined. While this may sound computationally expensive, modest CPUs, by today's standards, can already execute billions of operations in the time required to load a tape and seek, and this ratio will continue to increase.

```
frio_reset_file_record_counter(fd);
while (frio_record_remaining(fd)) {
  /* might come in in any order */
  frio_record_read(fd,buffer);
  process_record(buffer);
}
```

Figure 1: Code Fragment Using Free-Range I/O

6 Free-Range I/O

In this section we propose a new data storage paradigm, more akin to a persistent object store than the traditional Unix-style linear byte stream. We have dubbed this approach *free-range I/O*.

If a file is defined as an unordered collection of records, the system may be able to take better advantage of the current storage state in order to fulfill the application's requests. As with a database, the application may request any aribtrary record which it has not seen, giving the storage system the freedom to choose the easiest-to-access record to return.

Figure 1 shows one possible example of how free-range I/O might be used. SLEDs can provide the substrate on which it would be possible to build such an API. Such an API may simplify the process of coding applications to use SLEDs, as described in section 3. Free-range I/O may be implemented as a library over top of flat files, in typical Unix fashion, or by directly modifying the underlying storage structures. SLEDs provide the infrastructure to optimize record access ordering in free-range I/O.

7 Related Work

Multimedia file systems and video on demand storage servers [23, 6, 11, 12, 22, 3, 25] generally provide a mechanism for the application to communicate its needs to the storage system, but provide no feedback in the opposite direction, limiting the level of cooperation.

RSVP is the ReSerVation Protocol under development for supporting Integrated Services on the Internet [5]. Smooth integration of SLEDs with RSVP is paramount, but there are some important differences. RSVP is designed to support policing mechanisms and packet-based network jitter control, which are complementary to the problems SLEDs solves but perhaps not necessary in SLEDs themselves. RSVP has no "state change function" associated with accessing certain data, and deals with streams of data rather than the specific data objects of SLEDs.

Joe Touch has suggested a Resource Descriptor Interface [30], which proposes a model for representing all system resources as communications links. SLEDs are similar, but are more closely tied to the concept of data objects with storage and system state.

Hints are used by file systems to order disk accesses and make caching decisions. These hints can be provided explicitly [24, 1, 8] or determined by the system based on information such as file name or access history [14]. Hints have also been implemented across a network [26]. However, hints are typically one-way information; there is no way for the application to base its I/O patterns on the current state of the file system cache and storage devices.

Systems as far back as Tops-10 [9] have supported user-definable pagers, as does Mach. Application-controlled file caching [7] is most like SLEDs, but is tied to the concept of on-disk file systems, with no support for multiple storage hierarchy levels, and no ability to predict performance.

Tops-20, NAStore and Cray's Data Migration Facility [33] support a single-bit addition to the file system's inode to indicate that files are in archival storage rather than on disk. This aproach can be viewed as very simple SLEDs. However, many modern HSM systems support partial file caching on magnetic disk and multiple levels in the storage hierarchy, including optical disk, robotic tape and offline tape, with orders of magnitude difference in latency and bandwidth. The difference among these is important and not represented by a simple per-file bit.

SLEDs provide a way to characterize the performance of disparate devices, so that combined disk/tape operations can be written in a more device-independent fashion, providing a better growth path as device technologies continue to change. In some database systems the data is laid out explicitly on both disk and tape, and access methods must explicitly refer to one or the other [21]. Mainframe systems have explicitly made the distinction between data on tape and data on disk for years, while actively using tape to manipulate (as opposed to backup and restore) data. Sorting algorithms based on tape's sequential access method have also been used for many years [18]. SLEDs may provide the infrastructure for understanding the delay in retrieval necessary to support economic query optimization models in database systems [28].

HP's attribute-managed storage [4] incorporates performance descriptions of the storage devices to be integrated into a storage system. Their goal is to meet system-wide performance and data integrity goals through monitoring and feedback, providing quality of service by adjusting the system configuration over time. Synergistic relationships between SLEDs and attribute-managed storage are worth pursuing.

8 Future Work

Much work in defining the role of SLEDs in storage systems remains. Many details of the functionality need to be established, and SLEDs needs to be implemented and studied to make the ultimate determination of whether or not the performance improvements they appear to offer can be truly realized with modest programming effort.

The primary unresolved technical problem is representing the change in the state of the system caused by requests, as described above. However, even simple heuristic solutions offer the promise of improved performance.

In addition to managing one's own sequence of requests, the overall storage system state will of course be affected by other activity in the system. How can this be taken into account? Do SLEDs need to timeout, or provide an interface for requesting guaranteed performance through a global reservation mechanism of some sort?

The SLED model must also support replication and striping of data effectively. The data structures for this appear straightforward, but insuring that applications are aware of it and can take appropriate advantage is more complex.

An interesting question is whether or not SLEDs can be applied to memory systems, as well, where latencies can vary by two orders of magnitude or more from cache to main memory to remote memory, in distributed shared memory systems or non-uniform memory access (NUMA) architectures. SLEDs with free-range I/O could ultimately prove a useful enabling technology for data flow-like operating systems and languages.

SLEDs as presented here represent primarily a means for managing read access to data; the paradigm may be extended to writes as well by providing a reservation/hinting type of interface. The system must take into account that many devices have different read and write rates.

Since SLEDs offer a broad paradigm shift, much work in realizing them remains before a final determination of their worth can be made.

9 Implementation and Deployment Plans

In order to successfully deploy SLEDs, several technical problems must be solved, and a sufficient base of applications and knowledgable applications programmers must be built. The key problem is representing changes in system state, both those scheduled by the SLEDs-aware application and those initiated by other, unrelated clients of the storage system. In addition, for more effective use in networked environments, wider use of technologies such as RSVP is required.

SLEDs are expected to be deployed first as a performance prediction tool for use in multimedia and web servers. As experience is gained and the application base builds, SLEDs will gradually affect many data-intensive environments, such as databases.

SLEDs, although they represent a significant shift in I/O programming, can be deployed incrementally. The sophistication of applications, kernel services and device SLEDs are expected to go through several generations, all capable of coexisting.

10 Conclusion

In this paper, we have proposed Storage Latency Estimation Descriptors (SLEDs) as a means to more directly involve applications in the management of data movement in heterogeneous storage environments. SLEDs represent the latency and bandwidth to any given segment of storage, representing latency across twelve orders of magnitude. When utilized with hierarchical storage management systems, SLEDs have the potential to improve performance of applications by orders of magnitude by allowing them to intelligently control their own access patterns. SLEDs exploit the increasing imbalance between CPU and I/O device speeds by utilizing the former to improve utilization of the latter, in a device- and technology-independent fashion, so that SLEDs will remain a viable paradigm as storage technology evolves in unforeseen ways.

Acknowledgements

This research was sponsored by the Defense Advanced Research Projects Agency under Contract No. DABT63-93-C-0062. Views and conclusions contained in this report are the

authors' and should not be interpreted as representing the official opinion or policies, either expressed or implied, of ARPA, the U.S. Government, or any person or agency connected with them.

References

[1] High performance storage system design specification for client API delivery 2.0 draft 3.0, June 1994.

[2] ACM. *Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data*, June 1996.

[3] D. P. Anderson, Y. Osawa, and R. Govindan. A file system for continuous media. *Trans. on Computing Systems*, 10(4):311–337, Nov. 1992.

[4] E. Borowsky, R. Golding, A. Merchant, E. Shriver, M. Spasojevic, and J. Wilkes. Eliminating storage headaches through self-management. In *Proc. Second USENIX Symp. on Operating Systems Design and Implementation*. USENIX, Oct. 1996. work-in-progress abstract.

[5] R. Braden, L. Zhang, S. Berson, S. Herzog, and S. Jamin. Resource ReSerVation Protocol (RSVP) – version 1 functional specification. Internet draft, Nov. 1996.

[6] D. W. Brubeck and L. A. Rowe. Hierarchical storage management in a distributed VOD system. *IEEE Multimedia*, 3(3):37–47, 1996.

[7] P. Cao, E. E. Felten, A. R. Karlin, and K. Li. Implemention and performance of integrated application-controlled file caching, prefetching, and disk scheduling. *ACM Trans. Comput. Syst.*, 14(4):311–343, Nov. 1996.

[8] P. Corbett, D. Feitelson, S. Fineberg, Y. Hsu, B. Nitzberg, J.-P. Prost, M. Snir, B. Traversat, and P. Wong. Overview of the MPI-IO parallel I/O interface. In R. Jain, J. Werth, and J. C. Browne, editors, *Input/Output in Parallel and Distributed Computer Systems*, chapter 5, pages 127–146. Kluwer Academic Publishers, 1996.

[9] Digital Equipment Corporation. *decsystem10 Monitor Calls*, June 1976.

[10] C. Federighi and L. A. Rowe. A distributed hierarchical storage manager for a video-on-demand system. In *Proc. Storage and Retrieval for Image and Video Databases II, IS&T/SPIE Symp. on Elec. Imaging Sci. & Tech.*, Feb. 1994.

[11] D. J. Gemmell, H. M. Vin, D. D. Kandlur, P. V. Rangan, and L. A. Rowe. Multimedia storage servers: A tutorial. *IEEE Computer*, 28(5):40–49, May 1995.

[12] S. Ghandeharizadeh, A. Dashti, and C. Shahabi. Pipelining mechanism to minimize the latency time in hierarchical multimedia storage servers. *Computer Communications*, 18(3):170–184, Mar. 1995.

[13] J. J. Gniewek. Evolving requirements for magnetic tape data storage systems. In B. Kobler, editor, *Proc. Fifth NASA Goddard Conference on Mass Storage Systems and Technologies*, pages 477–491, Sept. 1996.

[14] J. Griffioen and R. Appleton. Performance measurements of automatic prefetching. In *ProceedingsInternational Conference on Parallel and Distributed Computing Systems*, Sept. 1995.

[15] B. Hillyer and A. Silberschatz. On the modeling and performance characteristics of a serpentine tape drive. In *Proc. ACM Sigmetrics Conference on Measurement and Modeling of Computer Systems*, pages 170–179. ACM, May 1996.

[16] B. K. Hillyer and A. Silberschatz. Random I/O scheduling in online tertiary storage systems. In *Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data* [2], pages 195–204.

[17] IEEE P1244. *Reference Model for Open Storage Systems Interconnection – Mass Storage System Reference Model Version 5*, Sept. 1994.

[18] D. E. Knuth. *The Art of Computer Programming, volume 3 / Sorting and Searching*. Addison-Wesley, 1973.

[19] J. Menon and K. Treiber. Daisy: Virtual-disk hierarchical storage manager. *Performance Evaluation Review*, 25(3):37–44, Dec. 1997.

[20] T. Mori, K. Nishimura, Y. Ishibashi, and H. Nakano. Video-on-demand system architecture using an optical mass storage system. In *Proc. Joint International Symposium on Optical Memory and Optical Data Storage*, pages 41–42, July 1993.

[21] J. Myllymaki and M. Livny. Disk-tape joins: Synchronizing disk and tape access. In *Proc. ACM SIGMETRICS Conference*, May 1995. Expanded version retrieved via ftp.

[22] W. O'Connell et al. A teradata content-based multimedia object manager for massively parallel architectures. In *Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data* [2], pages 68–78.

[23] B. Ozden, R. Rastogi, and A. Silberschatz. Architecture issues in multimedia storage systems. *ACM Performance Evaluation Review*, 25(2):3–12, Sept. 1997.

[24] R. H. Patterson, G. A. Gibson, E. Ginting, D. Stodolsky, and J. Zelenka. Informed prefetching and caching. In *Proc. 15th ACM Symposium on Operating Systems Principles*, pages 79–95. ACM, Dec. 1995.

[25] P. V. Rangan and H. M. Vin. Designing file systems for digital video and audio. In *Proc. Thirteenth ACM Symposium on Operating Systems Principles*, pages 81–94, Oct. 1991.

[26] D. Rochberg and G. Gibson. Prefetching over a network: Early experience with CTIP. *Performance Evaluation Review*, 25(3):29–36, dec 1997.

[27] C. Ruemmler and J. Wilkes. An introduction to disk drive modeling. *Computer*, 27(3):17–28, Mar. 1994.

[28] M. Stonebraker, R. Devine, M. Kornacker, W. Litwin, A. Pfeffer, A. Sah, and C. Staelin. An economic paradigm for query processing and data migration in Mariposa. In *ProceedingsParallel and Distributed Information Systems*, Austin, Texas, Sept. 1994.

[29] A. Sweeney, D. Doucette, W. Hu, C. Anderson, M. Nishimoto, and G. Peck. Scalability in the XFS file system. In *Proc. 1996 USENIX Technical Conference*, pages 1–14. USENIX, Jan. 1996.

[30] J. Touch. A view of communication middleware. presentation at Sigcomm '95 invited workshop on middleware, Aug. 1995.

[31] R. Van Meter. Observing the effects of multi-zone disks. In *Proc. USENIX '97 Technical Conference*, pages 19–30. USENIX, Jan. 1997.

[32] R. W. Watson and R. A. Coyne. The parallel I/O architecture of the high-performance storage system (HPSS). In *Proc. Fourteenth IEEE Symposium on Mass Storage Systems*, pages 27–44. IEEE, Sept. 1995.

[33] T. S. Woodrow. Hierarchical storage management system evaluation. In B. Kobler and P. Hariharan, editors, *Proc. Third NASA Goddard Conference on Mass Storage Systems and Technologies*, pages 187–216, Oct. 1993.

[34] B. L. Worthington, G. R. Ganger, and Y. N. Patt. Scheduling for modern disk drives and non-random workloads. Technical Report CSE-TR-194-94, University of Michigan, Mar. 1994.

Windows NT Clustered Storage Solutions for Enterprise Computing

Richard R. Lee
Data Storage Technologies, Inc.
Post Office Box 5073
Banner Elk, North Carolina 28604-5073
richard_lee@skybest.com
Tel: (704)-963-7773
Fax: (704)-963-7779

Abstract: The vast majority of all high-performance storage subsystems in the field today rely on the use of proprietary high-end computing platforms and OS's to act as a data server, e.g., IBM RS/6000, SGI Power Challenge, Sun Ultra Enterprise, etc.. These are characterized as Server Centric and require large server CPU's to meet end-user data demands, e.g., UniTree, FileServ, etc.

In addition to Storage Centric systems, there are a small, but growing number of sites that are beginning to utilize a distributed/high-speed switched architecture, e.g., HPSS. These are referred to as Network Attached and utilize expensive and exotic storage devices and switching fabrics to meet end-user data demands. These too rely on the use of proprietary CPU's and OS's to act as "data movers", e.g., IBM SP/2.

As data demand requirements continue to increase (some 60%+ CAGR) the need for truly distributed, low-cost, non-proprietary, high-performance storage subsystems will become more critical. In fact, many analysts, including this author believe that the solution to this challenge is key to the continued growth of computing across the enterprise.

One such solution that capitalizes on both the lessons learned from Server Centric and Network Attached storage subsystems, as well as taking advantage of low-cost, scalable, and open computing platforms is based upon the use of the emerging enterprise OS – Windows NT.

Overview

Windows NT is quickly maturing into a true enterprise OS, addressing all of the needs and requirements that such a term connotes. One of the key recent developments contributing towards the maturation of Windows NT has been the addition of clustering for both scalability and reliability increases.

Clustering will provide over time the ability to incrementally scale up any computing environment to meet end-user demand, along with supporting true fault-tolerance for maximum reliability. The cornerstone to success in clustering will be the storage subsystem centrally located within it. After all, when the final assessment is made, all that any of these architectures do is provide more reliability and higher availability of the single most important asset of the enterprise, "its data."

NT Clustered Storage Solutions

Storage options for Windows NT today run the entire gamut, from disk arrays and JBOD to tape drives and libraries to optical drives and jukeboxes. All of these are connected via

SCSI-2 FWD to host bus adapters within the server. This is a fairly straightforward architecture that is employed almost universally in the NT and UNIX spaces of the market. This architecture also has many single points of failure and performance drawbacks.

When Windows NT Enterprise Edition reaches full maturity in the late 1990's, system developers will be able to provide high-performance and fault tolerant storage systems on low-cost Standard High-Volume Servers at a fraction of the cost of UNIX and proprietary platforms in use today. These solutions will feature the following capabilities and attributes;

- n-Node (2-16+) linear scalability utilizing IA-64 "Merced" and Alpha CPU's
- Fault-tolerant computing with full application take-over
- Fibre Channel and SSA storage devices and interconnects (100 MB/s)
- Server Area Networks (ServerNet VIA) for wide-bandwidth, low-latency private network data sharing with no CPU lag
- Very Large Memory (64 bit) (1+ TB's) Support
- OS Resident Storage Management (High Ground, Seagate Software, Kodak/Avail)
- 64 bit logging file system
- TPM Performance of 20-30,000 per node

These capabilities rival, if not surpass, most of those found in standard UNIX and exotic solutions being fielded today, all without the high cost of ownership and proprietary solution limitations imposed by such products.

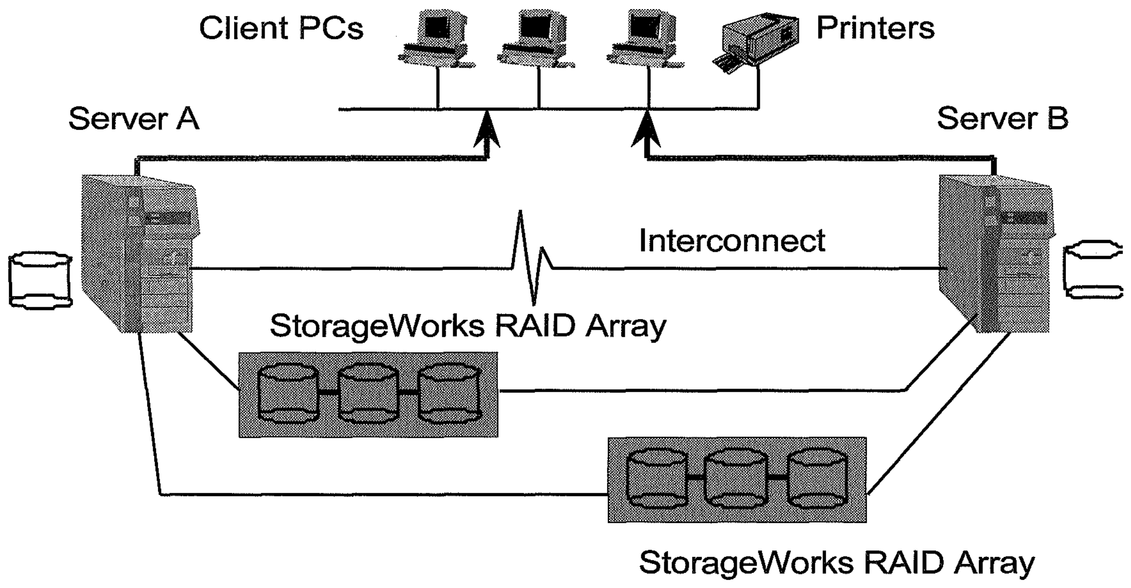


Figure 1: Microsoft Windows NT Enterprise Edition 2-node failover Cluster with Digital StorageWorks storage subsystems

Conclusions

Regardless of your feelings towards Microsoft and its technology and business practices, you cannot ignore the juggernaut that Windows NT Enterprise Edition brings to the market. With complete support of every major platform provider except one (Sun, if you did not know already!) and the dedication of every software developer to featuring their products on this platform, it will soon become the universal enterprise computing platform. The capabilities of this platform will not be limited to the low-end file and print segments, but will span all requirements from the desktop to the data center, with solutions and capabilities at all ends of this spectrum. High-performance storage will be a major focus of these developments as time passes by.

References

[1] Richard R. Lee, "A Technology Brief: Tandem Clustering Technologies", *Windows NT Magazine*, © June 1997

[2] Richard R. Lee, "Windows NT Clusters – A Storage Centric World", *Storage Management Solutions Magazine*, © DST, Inc., Sept/Oct 1997 Special Advertising Supplement

[3] Richard R. Lee, "Choosing the Right Storage Solutions for Microsoft's Windows NT Clusters", White Paper – Digital Equipment Corporation ©1997

[4] Richard R. Lee, "The Windows NT Storage Primer", Duke Press ©1997

Page intentionally left blank

An Architecture for Using Tertiary Storage in a Data Warehouse

Theodore Johnson
johnsont@research.att.com
AT&T Laboratories – Research
Florham Park, NJ 07932
tel +1-973-360-8779
fax +1-973-360-8050

Abstract

In this paper, we present an architecture for a data warehouse that provides convenient, flexible, and high-performance access to tape-resident data. Data warehouses allow an organization to store and analyze operational data. Many data warehouses (e.g., for telecommunications) create very large data sets that can be economically stored only on tertiary storage. At the same time, data analysts need to make a wide variety of decision support and data mining queries on the data. While one can (and typically does) extract compact summaries of the warehoused data, many queries can only be answered by using data from the full data set.

Using database systems to access large tape resident tables of small objects is conventionally viewed as a difficult problem, because it is easy to express an query that takes a very long time to process (e.g., the join of two tape-resident tables). Fortunately, decision support queries can usually be expressed in a way that can be implemented efficiently on tape-resident data. In addition, many data mining algorithms have been optimized to make a small number of passes over a data set. We present the features of a tape-based data warehousing system that provides efficient support for data mining and decision support queries on very large collections of small objects, a prototype implementation, and preliminary performance results.

1 Introduction

Interest in data warehousing has increased dramatically in recent years because of the vast increases in storage capacity of moderately priced on-line and near-line storage systems. Users from a wide variety of applications find that they can store and make use of information that previously was discarded or was inaccessibly stored in off-line tape racks. In a typical scenario, a data set (such as regional sales) is periodically harvested and ingested into the data warehouse. The data set is heavily indexed to accelerate decision support queries on subsets of the data. For example, a query on the sales database might be, "What is the weekly trend of shoe sales, listed by supplier and state?".

A commonly used data organization is a *star schema*. The primary table (called the *fact table* or the *detail data*) is indexed for joins against supporting tables (sometimes called "dimension tables"). The dimension tables allow selections of subsets for analysis. For example, dimension tables for the sales fact table might include a table on suppliers, on locations, and on products. The example query can be evaluated by finding all entries in the products table related to shoes, and performing a foreign key join on the sales table. Special indices have been developed for this type of operation, including join indices [20] and bitmap indices [14].

To facilitate decision support queries, summaries of the detail data are precomputed (companies such as Arbor and Red Brick specialize in databases for these type of "cube queries"). However, queries that cannot be answered from the summary tables are answered from the detail data. In addition, the detail data is made available for data mining queries.

Although the steep declines in the price of on-line storage has encouraged the increased use of data warehouses, many applications generate more data than can economically be stored on-line. One example is telecommunications data warehouses. AT&T collects very large volumes of billing and usage data for each of the services it provides. This data is used for a wide variety of applications, including identifying business trends, validation of billing procedures, detecting fraud, monitoring and optimizing network performance, and debugging network problems (other telecommunications companies have similar needs).

When such large volumes of data are collected, only the most recent detail data is stored on-line. Older detail data is migrated to tape, and only small summary tables are retained in secondary storage. When old detail data is queried, it is migrated back to secondary storage for analysis.

The current storage and structure of tape-resident detail data discourages experimentation. While HSM products can simplify the process of migrating data between levels of storage, the large-scale processing of tape-resident resident data remains difficult. Data that resides in a database must be exported to migrate it to tape, and imported to use it when migrated back to disk. The alternative to importing and exporting database tables is to use radically different methods for querying disk resident and tape resident data. While the user would like to give a declarative specification of the data set to be processed, the tape-resident files must be explicitly named and imported. If the total size of the queried data is larger than the migration cache space, the user might need to implement their own cache management algorithms (depending on the sophistication of the HSM cache management algorithms).

Data analysts would prefer to have a database-centric view of warehouse data instead of a file system-centric view, as common queries are easy to express with powerful query languages or 4GL tools. However, interfacing databases with tertiary storage is viewed as a difficult problem (see Section 7 for an discussion of related work). The problem is that databases which support standard query languages such as SQL (Structured Query Language) are designed for disk-resident data. Disk-resident file systems permit fast access to a large number of open files, fast random access, and updates in place, and database systems take advantage of these characteristics. Tape-resident data files have long access latencies, provide slow random access, and tape data is largely append-only. As a result, it is easy to express a query that is very hard to answer.

We observe that decision support data warehouses [17] have characteristics that make the use of an automated tape library for storage a reasonable option. First, the fact table is append only. Second, most queries make very long sequential scans through the data. Third, joins to tape resident data take restricted forms, being either joins to disk-resident "dimension tables", or are the equivalent of merging vertically partitioned[1] relations.

Our thesis is that a simple special purpose database can make automated tape library resident data warehouses efficient and easy to use. A small set of efficient access methods can be used as the engine for implementing a wide range of decision support queries and data mining algorithms. Because we know the schema of the data and the nature of the queries on the data, we can layout the data on tape in an optimal manner. By specifying the query in a declarative language, we can build a query plan out of optimized components.

In this paper, we present our views about how such a database should be built. Next, we discuss an architecture for such a system, along with a preliminary discussion of an API. We built a prototype database using the architecture. We discuss how we used the prototype to submit a query, and discuss performance results.

2 Motivation

This work was motivated by the need to perform data warehousing and data mining for telecommunications data analysis. In this section, we will discuss in greater detail the characteristics of typical queries in this problem domain, and show how they are similar to decision support type queries.

Queries over telecommunications data sets are made over sequences of descriptions of telephone calls, login sessions, circuit connections, data transfers, etc. A typical query selects a subset of data, joins each selected tuple to the dimension tables, and computes an aggregate over the joined tuples. The aggregates can require a very large number of tuples to compute accurately. For example, the tail of the distribution of the duration of circuit connections is of great interest. Often, only a subset of the data set is selected for aggregation. The tuples in the data set are small, on the order of 100 bytes or less each.

Telecommunications databases often contain tuples that express sequences of events. For example, one tuple indicates the setup of a connection, a sequence of tuples indicate the status of the call, and a terminating tuple indicates the end of a call. A common query is to report aggregates about such sequences. In some applications, one must be able to retrieve all records related to an individual user. For example, AT&T must store and retrieve telephone call records, in response to law enforcement requests. Often, the selectivity of

[1] *vertically partitioning* a relation means to split each tuple of a relation into two or more pieces, with each piece stored in a different file

these queries is on the order of one record retrieved per billion archived.

Decision support queries in the telecommunications domain have a great deal of similarity to generally accepted notions of what constitutes decision support queries in other domains (e.g., sales, financial, etc). Therefore, we believe that a system which supports telecommunications data warehouses will support general purpose data warehouses as well.

Modern automated tape libraries can support high data rates, whether through the prodigious performance of high end drives, or by using multiple moderately priced drives. However, access latencies and seek times are very large. Therefore, systems that make use of tape storage must access data in long sequential scans in order to achieve high performance. We observe that most data warehouse accesses to tape resident data can be made through long sequential scans, because the queries have the following characteristics.

- Most queries compute large-scale aggregates on the fact table. Often, the queries are simple trend or classification queries. Recent research [1] has shown that has shown that even more complex queries involving layers of conditions on the computed aggregates (e.g., "find all calls longer than the average call") can be made in a small number of sequential passes over the data set.
- Joins involving a tape-resident table have a limited set of forms. Most joins are to join the fact table to one or more of the dimension tables. The dimension tables are usually small, and can be pre-loaded into memory.

  Joins between tape-resident tables are often made between time windows of the respective tables. Tape resident tables might be joined because they represent vertical partitions of a tape resident table. Alternatively, the tables might represent different views of the same activity, collected by different processes (e.g., shipping records and billing records). While the table joins can become complex, joins between tape-resident tables resemble a multiway merge, instead of the cross product as has been investigated in the previous literature. Because of the temporal nature of the fact tables, merging is the natural way to join fact tables.
- The subsets selected for further processing are often fairly dense (e.g., one record selected in 100 or more). Since most disk blocks are accessed, the best query plan for performing the selection is to sequentially read the table, but to unpack and process only the selected records. Many database systems (e.g., Sybase) use *bitmap indices* on ROLAP data to support this type of query processing [14]. The sequential nature of the query plan makes it easy to implement on tape storage.

An additional requirement of telecommunications databases is to find records that occur very infrequently. We refer the interested reader to a related paper [6] for details on this type of tertiary storage indexing.

3 Design

We aim our design at the midrange systems commonly used as database servers: $O(10)$ high speed processors with $O(1 \text{ Gbyte})$ of main memory and $O(1 \text{ Tbyte})$ of on-line storage. Servers of this scale are inexpensive enough to become dedicated warehouse servers. Our goal is to extend the memory hierarchy by another one or two orders of magnitude with a moderate increment in cost by using an automated tape library holding $O(100 \text{ Tbyte})$ of data. Access to the tape resident data should be fast and efficient. At the least, querying tape resident data should be simple, at the best, transparent.

Given the nature of the data sets, the nature of the queries on the data sets, the technology used in database servers, and the performance characteristics of automated tape libraries (see [7]), we feel that efficient access to tape resident data dictates the characteristics of the data warehouse. Furthermore, we need a flexible design to support data mining and ad-hoc queries. In particular, we feel that the data warehouse should have the following characteristics:

1. **Lightweight architecture:** Conventional database systems incorporate locking, recovery, client-server computing, etc. Each of these features adds overhead to the data handling, but provides little benefit in our context. By leaving out these features (i.e., client-server) or using a lightweight implementation (i.e., locking and recovery) we can achieve a much higher throughput (other implementors [4] have taken the same approach). Similarly, implementing SQL on a tape-resident table involves immense difficulties in query planning, scheduling, data allocation, and so on. By restricting the query language, we can efficiently support the queries that a user would typically ask.

2. **Control over data layout:** Developing a data warehouse requires a great deal of planning. So we can assume that the data warehouse implementor understands what are typical queries on the data. For example, if a typical query makes a very long scan over a tape resident table, then the table can be horizontally partitioned[2], and the partitions distributed among n tapes. Other considerations might lead to an even more complex layout.

3. **Direct tape to memory transfers** Many architectures for implementing databases on tape resident data assume that the tape resident data is first loaded to disk, and then regular disk-oriented database algorithms are applied to the data. While this architecture is appropriate for some applications, we believe that for our domain processing data directly from tape is the better approach.

   Loading data from tape to disk before processing allows the use of existing database technologies, but it also creates two problems. First, transferring the files from tape

[2]*Horizontally partitioning* a relation means dividing the tuples into different tables, based on the value of the tuple.

to disk before use wastes resources – cache disk space, I/O bandwidth, etc. Second, managing the cache disk resources becomes a difficult problem itself, and the database system becomes complex, more difficult to implement, and more difficult to optimize.

Loading the tape resident data onto disk can improve throughput if the cached data is accessed repeatedly. However, we expect that this benefit to be minimal. First, typical accesses to tape resident data are likely to be long sequential scans, so repeat accesses before a cache flush are unlikely. Second, accesses to tables cached on disk are likely to also be sequential scans, and therefore can be performed efficiently from tape. Third, our architecture does not preclude loading a view of a tape-resident table onto disk for intensive analysis.

4. **Support for I/O parallelism:** Although tape drives have very slow seek times, modern tape drives have high transfer rates. Moderately priced high capacity tape drives, such as the DLT 7000, can provide a transfer rate of 7 Mbytes/sec or greater using reasonable assumptions about data compressibility. Furthermore, a collection of these drives, in parallel, can provide very high data rates. These high data rates are necessary because of the volume of data to be processed, and because in most cases selections are based on sequential scans.

5. **Support for CPU parallelism:** The complex aggregation of typical queries can be cpu intensive. Certainly, the high data rates available from the tape drives requires multiple CPUs to keep pace with the data processing. In addition, tape-resident data might need to be joined with disk resident data before processing.

6. **Support for indexing** While tapes are an inherently sequential media, indexing can significantly improve performance. Many decision support type queries process small subsets of the tape resident table. Subset membership is often determined by the value of an attribute, and therefore selections can be accelerated by using indices. Even if every tape block must be read into memory, unpacking and processing only the selected records can greatly reduce CPU requirements. We note that record unpacking might involve significant computation, because our studies have shown [7] that pre-compressing records can improve on the effective tape capacity and transfer rate.

   Existing work on indexing tape resident data uses technology developed for disk-resident data, or uses "metadata" to describe the contents of tape-resident files. These techniques are not sufficient for indexing tape-resident tables of small objects. We are researching more suitable indexing technology. For example, in [6], we describe an algorithm for indexing individual small records in a multi-terabyte tape-resident table, in which a typical key value will occur only a few times on a tape. We are refining this technology to make it applicable to cases where keys occur frequently.

7. **Support for massive aggregation and data mining:** As we have discussed, most decision support and data mining queries can be expressed as taking aggregates over a very large data set. Data mining queries and "complex" aggregation [1] might require two or more passes through the data set. In addition, the data can have the

semantics of a large number of interleaved time series. Aggregates on a time series are likely to group by the object that created the time series. This creates problems for parallel processing, because time series will cross partition boundaries.

4 Architecture

We have designed a prototype architecture to test our ideas. The architecture has a component that handles data ingest and a component that handles queries. The two components communicate through a control database as shown in Figure 1.

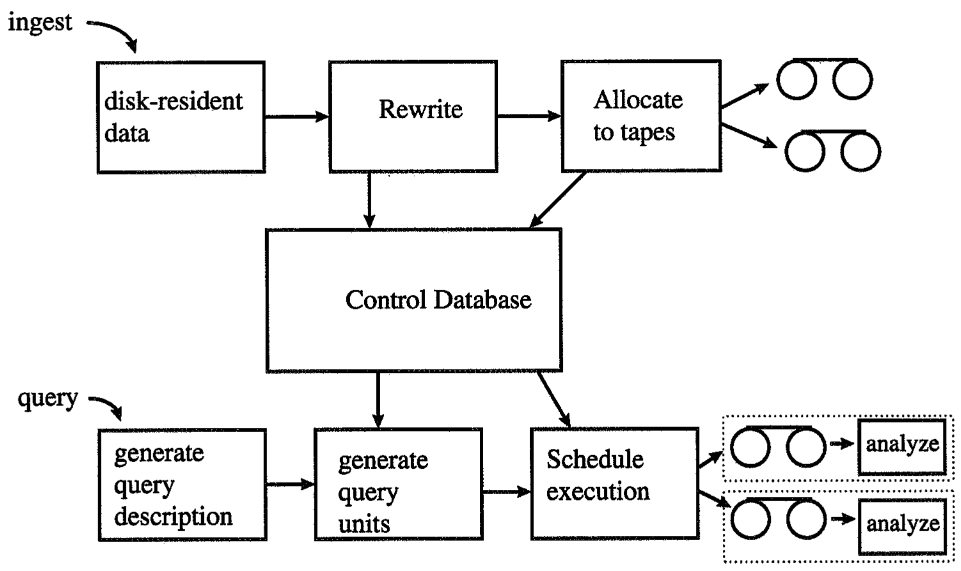


Figure 1: Data warehousing architecture.

We assume that data warehousing system has a disk-resident component and a tape-resident component. Updates to the fact table are put into a disk-resident table. Storing the newest data on disk is necessary because the newest data is usually the hottest data, and because summary tables must be built on the updates to the fact table. To make space for the newest data, older data must be migrated out. At this point a migration process is initiated that operates over the oldest portion of the disk resident fact table, transforming the data, allocating the data to tapes, and performing the migration.

Data transformation is necessary for good performance on tape-based queries, because disk and tape have very different performance characteristics [7]. In particular, queries on disk-resident data can support random access, while tape queries must primarily use sequential access. Strategies for improving tape access speed include vertical partitioning, partitioning on key value, selecting horizontal partitions for efficient seeks, replication, and resorting. Indexing tape resident data requires a different approach than indexing disk resident. The indices for disk resident data are dropped and indices for tape resident data are built.

Disk resident horizontal partitions are transformed into a set of tape resident files, which are then migrated to tape. The allocation strategy has two dimensions – the relative placement of files derived from the same disk-resident table partition, and the placement of files derived from successive partitions. The data placement strategy provides the opportunity for parallel I/O, as the data required for a query can be placed on multiple tapes, all of which can be read simultaneously.

Queries are initiated by submitting a *query description* to the system. A query description consists of a a specification of a *backend server*, of an *aggregation server* and a list of *query units*. The backend server reads tuples from tape, performs selections, projections and buffering, and distributes the results to the instances of the aggregation servers that have been started. The aggregation server performs joins on the tuples it receives from the backend server and computes the aggregate functions. A query unit is a minimal set of tape-resident data files that must be accessed together to perform a query. We assume that the invoking program generates the list of query units, although the control database is available to aid in generating the list. By this separation of responsibilities, a wide variety of programs can make use of the basic querying engine (query languages, data mining algorithms, user-written programs, etc.).

5 Query Language

In addition to a convenient programming interface, we are developing a query language for use in expressing ad-hoc queries. The language is similar to SQL, but it has several restrictions (to ensure efficiency) and several extensions (to provide the desired expressive power).

The restriction is that difficult tape joins are disallowed. Tape-resident tables can be joined to disk-resident tables. In addition, tape-resident tables can be merged. However, "cross product" joins between tape-resident tables are prohibited.

The query language is based on the extensions to SQL discussed in [1]. Instead of relying on joins to compute complex functions, "grouping variables" are used to express complex conditions. Queries expressed in this language are easy to optimize for execution on tertiary storage. We will publish details of this language at a further date.

6 Preliminary Implementation

We developed a preliminary implementation of our data warehouse architecture. The migration component will call a user-supplied program to transform a segment of data from its disk-resident data format into a tape-resident format. The partitions of tape-formatted data are migrated to tape in a controlled order. The location and defining predicates of each tape resident partition is recorded in the control database.

For an experiment, we migrated a 60 Gbyte data set to three tapes (the data set reduced to 45 Gbytes in the binary tape data format). The tape resident files were about 500 Mbytes each. The data set was "striped" across three tapes, in stripe units of about 1 Gbyte (every tuple was written on exactly one tape). The DLT 4000 achieved a 2 to 1 compression ratio on this data, and a (read) transfer rate of 1.9 Mbyte/sec.

The data set that we migrated to tape contains network operations data. Our test query collects all tuples related to a particular connection, and reports an aggregate function of them (this is a typical query). In the preliminary implementation, a query is submitted by specifying a data set and a time range (which determine the list of query units to be processed), the aggregation server executable, and a back-end executable. The aggregation server executable is identical to the one we use for queries on disk-resident data. The back-end executable is a simple single-threaded program that hashes tuples from tape to the aggregation servers.

We submitted a query using three back-end executables and five aggregation server executables. The query required 176 minutes to run, implying a processing rate of 4.4 Mbytes/sec. This processing rate is about 77% of the 5.7 Mbytes/sec that we would obtain if all three drives were transferring data at their maximum rate. An investigation showed that at most times, the back-end servers were transferring data at their maximum rate (the DLT 4000 is very forgiving of short delays in requesting the next data block [7]). Further, the time to mount all the tapes was small (a few minutes), and the seek time between files was negligible (as we accessed the files in their order on tape). However, the aggregation servers pause at the end of processing each partition to finish the processing associate with the partition and thereby free up memory. These pauses accounted for the bulk of the slowdown. We modified the aggregation server to reduce the lengths of the pauses, and obtained a processing rate of 5.2 Mbytes/sec, which is 91% of the maximum processing rate.

We note that the issues discussed in Section 3 were necessary for obtaining good performance. We did not have 45 Gbytes of temporary storage available, so we could not have processed the query by first loading the data to disk. The query made a single sequential pass through the data, so loading the data to disk would not have improved performance in any case. To obtain a moderate data rate, we needed to make use of parallel I/O and parallel processing. We expect to obtain a high data rate by using parallel I/O with the higher performance DLT 7000 drives. Finally, we note that although the interface to the data warehouse is very simple, it is easy to submit a wide range of queries. The back-end server and the aggregation server are generic, changing the query can be accomplished by changing an aggregation object and relinking.

7 Related Work

The database research community has recently investigated the integration of tertiary storage with a database management system. The drivers of this research include scientific databases [8, 18], multimedia [19, 2], and digital libraries [9, 3].

Several works have investigated methods for implementing general SQL databases with tertiary storage as the lowest layer in the memory hierarchy. Moran and Zak [10] describe an experimental integration of Oracle with the AMASS HSM managing an optical jukebox. Sarawagi [16, 15] discusses the architecture of *Sequoia 2000* for handling queries on tape-resident data. Tape-resident tables are partitioned into contiguous pieces, and a query on a table is transformed into a sequence of queries on the partitions. This architecture permits arbitrary joins, which are implemented by joining a cross product of partitions on the joined tables. Myllymaki and Livny [12, 13, 11] compare alternative algorithms for joining two tape-resident tables, again taking a cross product of the tables.

Much of the work on integrating tertiary storage with database systems is motivated by scientific database applications. Many of these (e.g., EOSDIS [8]) make extensive use of "large objects", for example tiles of satellite images. Issac [5] proposes an architecture in which a database that references file-resident large objects can be interfaced to a HSM, greatly simplifying the retrieval of data. DeWitt [22, 21] refines this idea in the Paradise DBMS.

8 Conclusions

The pervasive computerization of an organization's activities permits the large scale and fine grained collection of data related to vital operations. By loading this data into a data warehouse, operations can be analyzed and optimized. While the cost of on-line storage has dropped dramatically in recent years, the volume of data collected still exceeds on-line capacity by an order of magnitude in many application domains. In this paper, we propose an architecture by which a queries on a data warehouse can be easily and efficiently implemented. We base our architecture on the observation that most decision support type queries work well when the fact table is sequentially scanned. Thus most queries can be answered by reading data directly from tape.

We implemented an initial prototype of the tape-resident data warehouse. Though the architecture of the prototype is simple, we achieved good efficiency in reading from multiple tapes, and used a aggregation server that we developed for queries on disk-resident data. The good efficiency was because the the prototype has the characteristics listed in Section 3 (except for indexing).

Our future work on tape-based data warehousing includes:

- Research into system support issues, such as indexing, scheduling, and data layout.
- Language design for ad-hoc queries.
- Algorithm design and query optimization.
- Integration with data mining tools.

Acknowledgements

We'd like to thank the Session Chair, Rodney Van Meter, for his comments on an initial draft of this paper, and Damianos Chantziantoniou for his discussions of decision support query languages.

References

[1] D. Chatziantoniou and K. Ross. Querying multiple features of groups in relational databases. In *Proc. 22nd Very Large Data Base Conf.*, 1996.

[2] S. Christodoulakis. Multimedia databases. In *Intl. Conf. on Very Large Data Bases*, 1997. tutorial.

[3] R. Coyne and H. Hulen. Toward a digital library strategy for a national information infrastructure. In *Proc. 3rd NASA Godddard Conf. on Mass Storage Systems and Technologies*, pages 15–18, 1993.

[4] R. Grossman and X. Qin. Ptool: A low overhead, scalable object manager. In *Proc. SIGMOD 94*, 1994.

[5] D. Issac. Hierarchical storage management for relational databases. In *Proc. 12th Symp. on Mass Storage Systems*, pages 139–144, 1993.

[6] T. Johnson. Coarse indices for a tape-based data warehouse. In *Int'l Conf. on Data Engineering*, 1998.

[7] T. Johnson and E. Miller. Performance measurements of robotic storage libraries. In *Proc. IEEE Conf. on Mass Storage Systems / NASA Goddard Conf. on Mass Storage Systems and Technologies*, 1998.

[8] B. Kobler, J. Berbert, P. Caulk, and P. Hariharan. Architecture and design of storage and data management for the nasa earth observing system data and information system (eosdis). In *Proc. 14th IEEE Mass Storage Systems Symp.*, pages 65–78, 1995.

[9] A. Kraiss and G. Weikum. Vertical data migration in large near-line document archives based on markov chain predictions. In *Proc. 23rd Very Large Database Conf.*, pages 246 – 255, 1997.

[10] S. Moran and V. Zak. Incorporating Oracle on-line space management with lon-term archival technology. In *Proc. 5th NASA Goddard Conf. on Mass Storage Systems and Technologies*, pages 209–228, 1996.

[11] J. Myllymaki and M. Livny. Disk-tape joins: Synchronizing disk and tape access. In *ACM SIGMETRICS*, 1995.

[12] J. Myllymaki and M. Livny. Efficient buffering for concurrent disk and tape I/O. *Performance Evaluation*, 27:453 – 471, 1996.

[13] J. Myllymaki and M. Livny. Relational joins for data on tertiary storage. In *Proc. Intl. Conf. on Data Engineering*, 1997.

[14] P. O'Neil and D. Quass. Improved query performance with variant indices. In *Proc. ACM SIGMOD*, 1997.

[15] M. S. S. Sarawagi. Reordering query execution in tertiary memory databases. In *Proc. 22st Very Large Database Conference*, 1996.

[16] S. Sarawagi. Query processing in tertiary memory databases. In *Proc. 21st Very Large Database Conference*, pages 585 – 596, 1995.

[17] D. Schneider. The ins and outs (and everything inbetween) of data warehousing. In *Proc. 23rd Intl. Conf. on Very Large Data Bases*, pages 1–32, 1997. in Tutorials.

[18] M. Stonebraker. Sequoia 2000: A next-generation information system for the study of global change. In *Proc. 13th IEEE Symp. on Mass Storage Systems*, pages 47–53, 1994.

[19] P. Triantafillou and T. Papadakis. On-demand data elevation in a hierarchical multimedia storage server. In *Proc. 23rd Very Large Database Conf.*, pages 226–235, 1997.

[20] P. Valduriez. Join indices. *ACM Trans. on Database Systems*, 12(2):218–246, 1987.

[21] J. Yu and D. DeWitt. Processing satellite images on tertiary storage: A study of the impact of tile size on performance. In *Proc. 5th NASA Goddard Conf. on Mass Storage Systems and Technologies*, pages 460–476, 1996.

[22] J. Yu and D. DeWitt. Query pre-execution and batching in paradise: A two-pronged approach to the efficient processing of queries on tape-resident data sets. Technical report, University of Wisconson, Madison, 1996. Available at http://www.cs.wisc.edu/ jiebing/tape.ps.

On the Design and Implementation of the Multidimensional CUBEStore Storage Manager

Wolfgang Lehner, Wolfgang Sporer
Department of Database Systems
Friedrich-Alexander University of Erlangen-Nuremberg
Martensstr. 3
D-91058 Erlangen, Germany
{lehner, wgsporer}@immd6.informatik.uni-erlangen.de
Tel: +49 9131 85-7800
Fax: +49 9131 85-8854

Abstract: CUBEStore is a storage manager designed within the CUBESTAR project to fit the special needs of multidimensional applications as found in the area of Statistical and Scientific DataBases (SSDB). Some of the goals being achieved within this project were built-in support for multiple dimensions and good performance for range queries regardless of the dimensionality of the data. The general characteristics of SSDB-applications, the derived requirements with regard to a storage management system and the specific open architecture as well as the reference implementation of CUBEStore are described in this paper.

1. Introduction

With the advent of "Online Analytical Processing" (OLAP, [3]) and "Data Warehousing" as its data delivering platform, the well-known database research field of "Statistical and Scientific Databases" (SSDB, [18]) comes to a rejuvenation. Typically, SSDBs are used to store and retrieve satellite weather data or the numerical results of large physical experiments. They mostly still stick to tables for presenting results of a query. OLAP applications show the same data access and analysis characteristics, just replacing statistical tables by multidimensional data cubes ([19]).

In certain points, the characteristics of SSDBs differ remarkably from standard "Online Transaction Processing" (OLTP) database applications nowadays commonly built on top of relational database systems. Therefore, proprietary systems tailored to fit the special needs of SSDB applications are quite likely to be found in this area. Some of the points to be taken into regard are:

- efficient access to very large multidimensional data volumes
- only a limited amount of data is needed per request
- the physical storage structure should in some way reflect the logical, i.e. multidimensional structure of the data

The overall goal of the CUBESTAR project is to build a reference system matching these requirements. According to the ANSI/Sparc 3-Level Database Architecture ([21]), the CUBESTAR project works on all three levels of this architecture, thus trying to maximize usability and performance without the limitations of other projects approaching just one of these levels.

On the *conceptual level*, the *CROSS-DB data model* ([12]) is used to represent raw facts and figures as well as the basic structures supporting the data analysis phase. The description of different users' views of the data corresponds to the *external level* and is performed by queries formulated in the CUBEQueryLanguage *(CQL*, [1]). In general the distinction of several application-oriented external views on the same application-independent conceptual schema provides *logical data independence.*

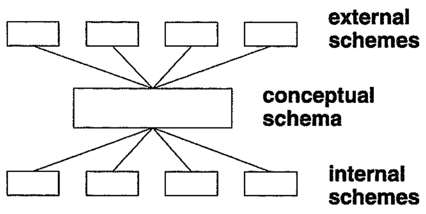


Figure 1. ANSI/Sparc 3-Level DB Architecture

CUBEStore, as to be described here, represents an important part of the *internal level*, providing CUBESTAR with adequate support for physically storing and retrieving data. The separation of a conceptual level from an internal level provides *physical data independence*, meaning that the model of the mini world is independent from its implementation. The most important means to make CUBEStore more efficient for SSDBs than standard database architectures are *distribution* and *replication.* In some way, CUBESTAR can be viewed as some kind of MiddleWare ([2]), providing efficient access to aggregated information based on large data volumes distributed over different servers and over different storage media. Thus, from an user's point of view, CUBESTAR implements a fast *'Information Everywhere'* service, with CUBEStore as the technical basis for data distribution and replication.

Referring more specifically to CUBEStore, the next section covering related work presents some other approaches of how to overcome the shortcomings of traditional database systems for multidimensional databases. The third and fourth section describe the characteristics and the requirements of multidimensional data respectively in more detail. Afterwards, the design of CUBEStore is presented, and finally, interesting features of the implementation are highlighted. The paper concludes with a short summary.

2. Related Work

In general, two streams of implementing multidimensional data models can be distinguished in scientific literature as well as in commercial products ([4]). The 'relational'-OLAP approach (ROLAP) transforms each query formulated in the multidimensional view into an SQL-statement, to be processed on a relational database engine. On the other side, the 'multidimensional'-OLAP approach (MOLAP) implements the multidimensional model directly on the physical layer. Since we (and probably the majority of the database research community) believe that there is no unique winner in this discussion, both directions should be supported by a database system providing efficient data access for OnLine Analytical Processing. Hence, CUBEStore is designed to be an *open* storage management system, capable of using relational *and* multidimensional drivers. As our outline of the basic CUBEStore architecture especially emphasizes the pure multidimensional way, we are now going to have a short look at other projects dealing with open storage management in general and multidimensional storage techniques in detail.

Supporting database systems by a sophisticated storage subsystem is a long story in database research ([20]). Furthermore, much work was done by Sarawagi, incorporating tertiary storage media *directly* into the database system ([17]). A different, *indirect* way of extending the capabilities of existing database systems is described in [14]. A virtual disk storage manager called Daisy is offered, operating similar to a virtual memory manager by transparently 'swapping' data blocks between different storage media, e.g. hard disk and tape drives. This is an interesting method to enhance existing systems, but one drawback of its transparency is the lack of integration into the database system, making it more difficult to control the activities of the storage manager.

Addressing the problem of multidimensional storage techniques, we will sketch two different ways to tackle this problem. The NASA's *CommonDataFormat* (CDF) and its extension *NetCDF* [6] are similar to CUBESTAR in the fact that they stand for a new, self-contained system adopted to the needs of multidimensional data as good as possible. But in contrast to the aim of the CUBESTAR project to build a full-featured database system as explained before, CDF mainly consists of a combination of the I/O-library and utilities to use this library. NetCDF extends the CDF data format to become more flexible and machine-independent. Thus, NetCDF might become a single module of the CUBEStore system in the future.

Relational database vendors are aware of the need to provide multidimensional data access. Oracle for example adds a new data type called *HHCODE* to their relational database system ([16]). This way offers the ability to integrate relational and multidimensional data in one system, a fact that might be outstanding to many possible users, as it allows a soft migration. On the other hand, it will be hard to adopt such a system fully to the needs of multidimensional applications. For instance, with an HHCODE-type field, it is not possible to query for exact data, only ranges can be requested, so an exact reproduction of the original values is rather hard. In general, adding multidimensional extensions to relational database systems (like the DataBlade technique of the Informix Universal Server [9]) might solve some users' needs rather comfortable, but it seems not appropriate for building a fully multidimensional application.

3. CUBEStore-Characteristics

This section tries to make the reader more familiar with the characteristics of SSDBs in general and some of the consequences for a storage management system. Afterwards, a model for redundant storage of information will be sketched.

3.1. Sample Scenario

One of the roots of the CUBESTAR project lays in a series of projects with our industrial partner, the worldwide operating retail research company "GfK". Their task can be shortly characterized by the following analysis steps: in a first phase, sales figures of single articles in single shops (mostly from Point-of-Sales systems) are gathered, classified and periodically added to a common raw database. Based on these data, typical statistical analyses like distributions, top-n calculations, etc. are performed. In general, the GfK-database is a good example for the basic properties determining the character of SSDBs [10]:

- multidimensional data structure
- data access according to dimensional structures
- very large amounts of data
- periodic bulk updates with an append-only semantic for the vast majority of data

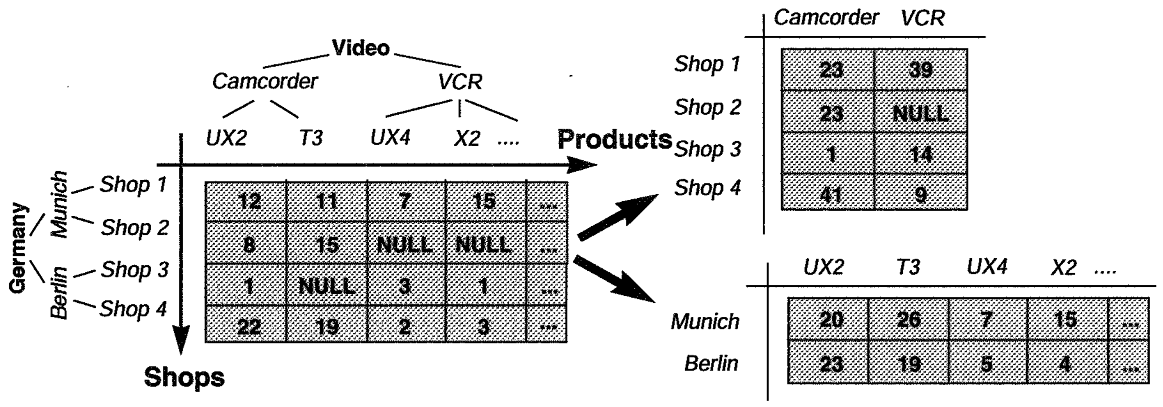


Figure 2. Sample Sales Figures of Video Equipment in Germany

For example, the left side of figure 2 illustrates some sample sales figures in a two-dimensional context. The hierarchically organized dimensional structures of the dimensions "Products" and "Shops" prescribe data access during the analysis process. Another interesting feature of multidimensional data which can also be seen in this example is the fact that data density is often very low, i.e. most of the possible multidimensional values are not existent. In our example densities between 4% and 20% were observed.

The right side of figure 2 shows two possible derivations: the upper one holds summarized values according to different product families subsumed by the notion of 'Video'; the lower one reflects the result of an aggregation process in the geographic direction. The raw database has large data volumes (10 GBytes in total for sales figures for German shops in a single gathering period for the current retail research scenario). Hence, for answering queries efficiently, data has to be *aggregated* according to different dimensions and at many different levels of detail, or *granularity,* even differing between the dimensions within one query. Typically, sales figures like Camcorder versus VCRs are requested not for single shops, but for instance summed up for the whole of Germany. On the other hand, the area manager of a retail chain running many stores might be interested in how his shops in Munich compare to those in Berlin.

Therefore, the system has to provide additional mechanisms to be able to cope with such requests. The problem becomes even worse, as all the data is kept for at least two years and used for time-oriented sequence and long-term analyses. Thus, time as a third dimension, missing in figure 2, must be added in the model and the data volumes must be multiplied by a corresponding factor (24 for two years on a monthly reporting frequency).

3.2. A Redundancy Model supporting SSDB-Applications

The append-only characteristic of SSDB-applications leads to the natural conclusion to use *redundancy* as a means to make such database systems faster. A project with GfK, published in [11], showed that an impressive speedup can be gained by using aggregated data instead of raw sales figures, and if the basis for those aggregated data stays rather stable, they can be reused quite often.

From a more general point of view, redundancy in the context of SSDB can be seen as representing one logical object x by n physical objects x_i, (i=1,...,n), with n being the number of physical representations. Each physical object x_i is generated by applying a function $x_i = f_i(x)$ to a logical object. According to the applied function f, two different kinds of redundancy can be distinguished:

- *Horizontal Redundancy (= Replication):*
  Using the identity function $x_i = f(x) = id(x)$, the physical object corresponds to the logical object. This case, with two or more physical objects[1], represents the classical way of replicating data to increase local availability and local access performance of remote data.
- *Vertical Redundancy (= Aggregation):*
  In this case, f corresponds to an aggregation function like $x_i = SUM(x)$ or $x_i = AVG(x)$. The derived physical objects can be seen as persistent results of a computation on a different object, probably of finer structure. Again, this is done mainly for performance reasons, to prevent often needed data to be recalculated from mostly unchanged raw data.

4. Requirements for an SSDB-Storage System

Consequently, three major requirements were identified for CUBEStore, influenced by the characteristics described above and the use of redundancy to improve query performance:

- being able to support different storage media
- being freely configurable, especially *what*, *where* and *how* data is stored
- reflecting the multidimensional data structure physically

The reasons for integrating different ways of storing data are now described, followed by an explanation of the need for a flexible way to configure a multidimensional storage manager, including the ability to reflect the multidimensional data structure physically.

4.1. Support for Different Storage Media

As SSDB applications are often used for several years, sometimes even more than a decade, the amount of data to be administered by CUBEStore can easily become magnitudes larger than other databases and reach several terabytes. Obviously, a limitation to magnetic hard disks as single mass storage medium is rather inappropriate here. Accordingly, the *open* concept of CUBEStore supports, by means of its *abstract, media-independent* interface, almost any storage medium, thus giving the user the possibility to decide which one fits his needs best. Organized in a hierarchic pyramid, the well-known principle of using cheaper but slower media for less often needed data can thereby be applied (figure 3). This includes, of course, the use of network drivers for horizontal redundancy.

Nevertheless, at least from within the system it should be possible to query the costs of accessing data on different storage media. This will enable the query optimizer to decide which volume to choose, if different options are available. In general, these costs can be

1. Without going in to much detail, it must be noted that at least one $x_i = id(x)$ must be present in the system

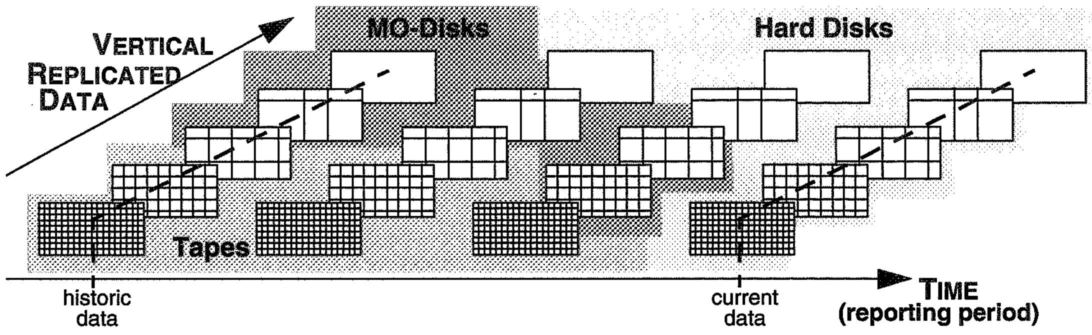


Figure 3. Data Distribution within the Storage Hierarchy

divided into *media access costs* and *transfer costs*. The former describe the costs for gaining access to a CD-ROM in a juke box, for instance, independent of how much data is needed. The latter tells the system about the actual transfer costs, normally of course dependent on the amount of data to be transferred. There are several other interesting characteristics about mass storage media that can be taken into regard. In general, the system needs to *classify* its media, and as a consequence of the former paragraph, this information has to be maintained separately from the original data in the so-called *Meta Information System*. It must be possible to decide which media to use *before* actually accessing the chosen medium.

4.2. Flexible Configuration

In addition to the claimed support for different storage media, their usage should be fully *configurable*, to adopt a system to the changing needs of its users. A global *Allocation Manager*, supported by local *Allocation Agents* might then move older or less often used data to a cheaper medium according to certain preset parameters, maybe on a different site. Their tasks include to decide what and how many physical data partitions should be used for one logical data partition. For instance, this can be determined according to access patterns, given priorities or granularities. Another part of their work is, as said before, to decide which sites should replicate frequently used data. Figure [4] shows what such a configuration could be like. It includes an example for horizontal replication, or duplication of data on a different site and for vertical replication or aggregating data.

Not only needs the system to be open to different types of storage media, equally important is its capability to support different *strategies* and *algorithms* transparently, for example with regard to the way the multidimensional data is linearized and compressed. In particular the chosen sequential ordering is a key to improve or slow down query perfor-

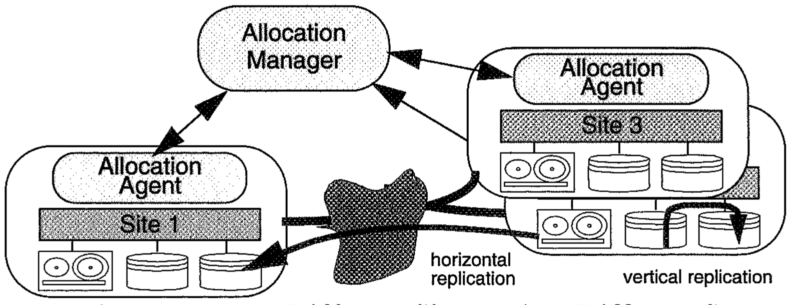


Figure 4. Allocation of Mass Data at Different Sites and on Different Storage Media

mance. Therefore, the physical representation of the multidimensional data must be chosen very carefully and it should be possible to adopt it to the needs of different applications. Furthermore, as a consequence of the low data density, intelligent compression techniques are a real necessity for efficient data storage.

To put it into a nutshell, CUBEStore provides, as a result of these requirements, an open, configurable data storage system for handling large volumes of multidimensional data with a potential high sparsity. Support for horizontal and vertical replication of data with regard to geographic distribution as well as distribution over inhomogeneous storage media, makes CUBEStore the ideal basic technique for a database management system fitting SSDB-applications' needs.

5. Design

Having explained the requirements that determined the design of CUBEStore, first of all the CUBEObjects as the core entities of CUBEStore are described. This is followed by a basic scheme for the possible division of work between the drivers within a CUBEObject. Finally, two possible configurations are presented. Some of the ideas used in this approach can be found in [5], [8], [6] and [16].

5.1. The CUBEObjects

According to the requirements described before, design and architecture of the CUBEObjects, the most important components of CUBEStore, are presented in this section. A CUBEObject is a logical representation of a data partition. Figure 5 illustrates how its main parts work together:
- CUBEDrv: driver inside a CUBEObject
- CUBEData: data objects for these drivers
- CUBEIniFiles: configuration files of the CUBEObjects

A CUBEObject is something like a black box offering a well-defined interface to its users to store and retrieve data. The data exchanged between them and a CUBEObject or its CUBEDrv is called CUBEData. The CUBEDrv form the core of a CUBEObject, they determine what and how it performs. A driver queue contains a list of drivers according to the configuration found in the CUBEIniFile belonging to that object. Getting CUBEData in the standard format described later, a CUBEObjects involves its first driver asking him to process the CUBEData. The CUBEDrv transforms the CUBEData according to the requirements of the next driver in the queue, and this goes on until the last one succeeds in storing CUBEData, or in retrieving the requested data from the storage media and handling the CUBEData back upwards the driver queue.

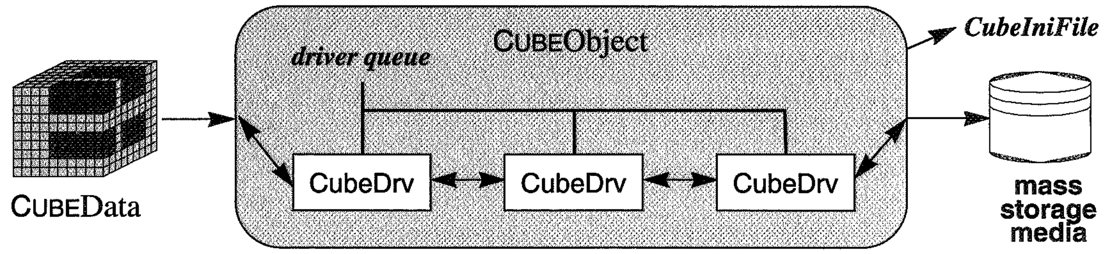


Figure 5. Architecture of a CUBEObject and its components

The division into several CUBEDrv ensures modularization and reusability. A new compression technique, for instance, can easily be added by implementing just a compressing driver and changing the CUBEIniFile as to load this driver to its place in the driver queue. Or, to use the recovery capabilities of a standard database system, only a simple driver transforming CUBEData into an SQL query is necessary. The same argument holds for network drivers in a distributed system or, e.g. for dummy drivers to overcome the limitation to one CUBEObject, thus enabling clustering of several logical data partitions data into one physical partition.

5.2. A Basic Scheme

The first step was to develop a basic scheme on how to divide work between the different drivers. According to this scheme, the task of storing multidimensional data partitions on a storage medium consists of dividing the partitions into a number of data blocks. Secondly, a format for storing (and compressing) data within these blocks has to be found. And finally, the data blocks have to be written to or read from the medium. In other words, we have got three different kinds of drivers (figure 6):

- a *block directory driver* responsible for splitting the partition into blocks
- a *block access driver* determining the storage format within these blocks
- a *block driver* reading and writing blocks to an external medium

One drawback of splitting a data partition into several blocks is the fact that additional administration has to be done by the block directory driver. On the other hand, to maintain a minimum of flexibility, in particular with respect to write performance, e.g. when adding or correcting some data, some kind of division is needed anyway.

5.3. Two Configurations

Using the basic scheme of the last section, two possible ways to interpret this scheme are described now. The first aims at using storage media of a well-known fixed block size, whereas the second one is more appropriate to devices working with sequential access. Finally, some remarks are made about using tape drives.

5.3.1. Storage Media with Fixed Block Size

This configuration is the first choice when storing data on normal hard disks. This implies a fixed block size and random access to all blocks. It is now explained using figure 7:

To start at the bottom of the driver queue, the task of the block driver is rather simple. Block number and fixed block size enable it to find the requested block quite easily. The same holds for replacing one block with another or adding new blocks.

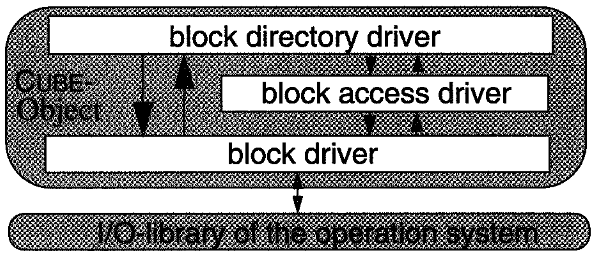


Figure 6. A Basic Scheme for Configuring a CUBEObjects

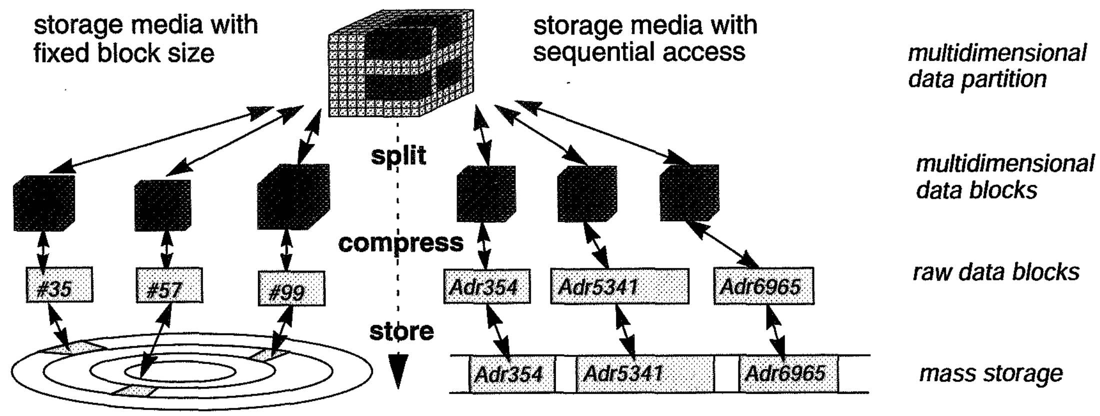

storage media with fixed block size
storage media with sequential access
multidimensional data partition
split
multidimensional data blocks
compress
#35
#57
#99
Adr354
Adr5341
Adr6965
raw data blocks
store
Adr354
Adr5341
Adr6965
mass storage

Figure 7. Two Sample Driver Configurations for Different Storage Media

The block access driver is responsible for compressing multidimensional data into a raw data block of given size. This implies that there might be some space left, or that two raw blocks might be needed.

The most complex driver in this configuration is the block directory driver, working on top of the block access driver with blocks of variable size. Therefore, further directory information is necessary to divide the data partition as good as possible and to determine the raw data block number quickly. On the other hand, removing the block access driver, thus enabling the block directory driver to use fixed size blocks as well cannot be a solution with regard to the low data density mentioned earlier.

5.3.2. Storage Media with Sequential Access

If, in contrast to the former section, the block size of the storage medium cannot be fixed in advance, this configuration might be a replacement. It uses the stream paradigm, viewing the medium as a single stream without bigger units of access. Again, it is explained using figure 7:

In this case, the data partition can be divided according to a standard mapping into fixed size blocks, resulting in a rather simple block directory driver. Now the block access driver gets blocks of a fixed size and compresses them as good as possible, resulting in variable sized blocks. Consequently, the block driver is becoming slightly more complex, as it has to maintain information about addresses and sizes of the raw data blocks stored on the storage medium. Therefore, this model is not appropriate if blocks are often moved or change their size.

As tape drives are often used in juke boxes or something similar, long access times are typically for this medium. Therefore, data on tapes are often compressed once more as a whole, as the time for doing so is rather short compared to access time. This can easily be done by adding another driver at the end of the driver queue and hence no extra configuration is needed for tape drives.

6. Implementation

Following the first of the two possible configurations for block devices with random access, the implementation of a prototype is now described. Again, the requirements are presented first. Thereafter, the implementation of a CUBEObject is explained. MemCubes, the means of exchanging data with a CUBEObject, get a section of their own, followed by a short overview over the different drivers of this implementation.

6.1. Analysis and Requirements

The standard requirements for modern software like flexibility, portability, extensibility and reusability must be inherent properties of an open system like CUBEStore, capable of adopting to different and changing needs. Nevertheless, the overall performance goal of being faster than standard database systems has to be reached, as this is the key factor for its existence. In addition to this, the traditional solution of using more or faster hardware might be slightly harder with data volumes reaching several terabytes, as said before. Hence, the C++ programming language has been chosen for the implementation. Its object-oriented features offer almost everything necessary for modern program design. Static type checking and nothing like a runtime garbage collection enable performance. Moreover, the availability on almost all modern computer architectures is the basis for portability.

The relatively clear distinction between the production and query phase in the lifetime of multidimensional SSDB data leads to the decision to look at fast read-only access as the more important feature. A lot of emphasis was put on ensuring good read performance regardless to the requested dimensions. This implies that a traditional primary key linearization could not be used. The implementation was simplified by the restriction to numerical data, leaving coding and decoding to a separate module. The units administrated by CUBEStore are data partitions. Dividing and joining these units is the task of the already mentioned Allocation Manager. As the query optimizer can be multithreaded, the CUBEStore code has to be reentrant, but not necessarily multithreaded itself.

Error handling, at least for unexpected errors, is done almost completely by throwing exceptions, hierarchically organized and named by a unique combination of major and minor error number. For handling standard tasks, the class library GNU libg++ was used.

6.2. The CUBEObjects

As said before, a CUBEObject is the logical representation of a CUBEStore data partition. Data can be accessed only if the requested CUBEObject has been opened before. This is done according to the description in the CUBEIniFile, so the name of a CUBEIniFile identifies a CUBEObject. It also serves as a simple locking mechanism. A more sophisticated name and access administration is to be implemented in future versions of CUBEStore. On the other hand, file system methods enable e.g. the transparent use of network devices. Each CUBEDrv to be loaded for a CUBEObject has its own paragraph in the *driver section* of the CUBEIniFile, with the content being dependent on the driver. The *cost section* in the CUBEIniFile describes access and transfer costs as explained at the beginning of the design section.

To prevent the CUBEObjects from having to know the type of each CUBEDrv it uses, thus making it impossible to use drivers written after the compilation of a CUBEStore application, the drivers are generated by a separate, global or local CUBEDrvFactory with a standard interface for all drivers. As a result, only this driver factory needs to be updated if new driver classes are developed. Accessing data in a CUBEObject is done by two basic methods, Read and Write. They transfer data to and from CUBEObjects in the MemCube format described later. Actually, as shown above, they just order the first driver in the queue of the CUBEDrv in a CUBEObject to start working.

```
Bool Read(MemCubeData<T>& cube_data) {
  return (drv_queue.First())->Read(cube_data);
}
```

```
Bool Write(MemCubeData<T>& cube_data) {
  return (drv_queue.First())->Write(cube_data);
}
```

Data is held in main memory using the multidimensional MemCube class. Speaking more exactly, the MemCube module contains a group of classes: Firstly, there is the class CUBEIndex describing a multidimensional point by its coordinates. Two points form a convex ScanRegion. The class MemCubeContainer is responsible for storing data in a format similar to C-arrays. The MemCube class itself offers a comfortable interface to access data in a multidimensional way, using the three helper classes just mentioned. Among others, it allows to compare, replace or resize MemCubes as a whole.

6.3. The Drivers

This section corresponds to the largest portion of the implementation effort, the CUBEDrv. Based on the configuration for random access block devices, a variety of drivers was developed. Their hierarchical structure is shown in figure 8. Corresponding to each layer of the basic scheme explained in the design section, there is a base class, containing an abstract interface definition of the service offered by this layer: BlockDirDrv, BlockAccess-Drv and BlockDrv.

6.3.1. The Block Layer

To start bottom-up, the block layer is the most low-level layer providing others drivers with the capability to read and write a raw data block of a fixed size as specified in Fixed-SizeBlockDrv. The twins SyscallBlockDrv and StreamBlockDrv perform the task of transferring data to and from disk, currently using system calls or C++ streams. Management of

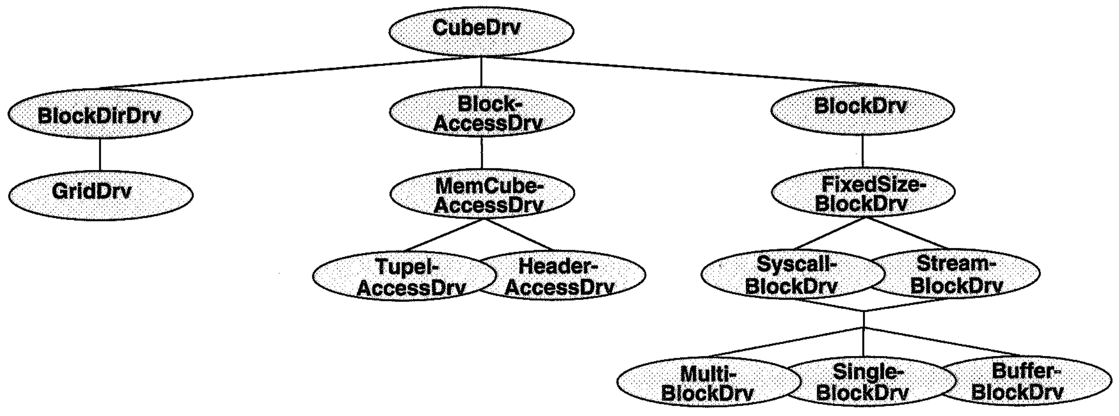


Figure 8. Hierarchy of Driver Classes

the dynamic main memory used for the blocks read is done in three different ways by MultiBlockDrv, SingleBlockDrv and BufferBlockDrv. The first one does almost no optimization, the second one tries to reuse memory as far as possible, whereas the third one serves as a front end to a global system buffer implemented in a separate module. Thus, the BufferBlockDrv demonstrates of how to overcome the borders of a CUBEObject, useful e.g. for implementing a storage format clustering several logical partitions into one physical.

Their base class is, as shown by the following short code extract, only given as a template. By using this enhanced mechanism, it became possible to unify the use of two different I/O -drivers with three different dynamic memory management drivers in quite an elegant way:

```
template <class T>
class MultiBlockDrv : public T
{
        public:
                MultiBlockDrv(CubeObject* cubeP, const int size, const String& file_name, const Mode mode=OPEN)
                        : T(cubeP, size, file_name, mode) { }
}
```

6.3.2. The BlockAccess Layer

In this configuration, it is the task of the block access layer to ensure that the multidimensional data are stored in blocks of fixed size, and as a consequence, the block access layer determines the format within these blocks. As it can be seen in figure 8, there are two implementations of block access drivers, namely HeaderAccessDrv and TupelAccessDrv. The latter stores a multidimensional value just by adding its coordinates. Even this simple format incorporates a compression of data, as long as the data density is lower than $1/(1+n)$ with n being the number of dimensions. Changing data, including adding or deleting values, is rather easy with this format.

A more sophisticated format is implemented in the HeaderAccessDrv: This driver uses the SingleCountHeader method according to [7]. With this method, one *distinct* value can be removed from an array, here the NULL standing for 'data not available'. To do this, a header noticing the distance between non-NULL values is calculated. Standard C linearization is used to create the array needed for compression out of the multidimensional data, so this format is not as stable to operations resizing the multidimensional data block. On the other hand side, since no space is needed for unavailable data, resize operations should not be necessary too often, an advantage of this format over alternative methods like a bitmap header.

6.3.3. The Directory Layer

The most sophisticated driver in this model is the block directory driver, whose task is to split the multidimensional partitions into smaller blocks. To fulfill the requirement of being rather independent to the dimension used for access, the grid file format originally proposed in [15] has been used. On average, its performance over all dimensions is quite good, although a method using the primary key for linearizing might be better with respect to this special dimension. An additional benefit of this format is its local character, with only limited changes after adding or deleting data. Furthermore, its open character makes it an ideal partner for different block access drivers.

A grid directory consists of two different parts: A grid scale for each dimension and a multidimensional grid array. The latter reveals the data block numbers where the requested data can be found, and the scales indicate the upper and lower bounds of the data contained in a block. Using the example given in figure 9, it can be seen that all values with coordinates between 13 and 21 in dimension 1 and between 66 and 81 in dimension 2 are contained in block number 27.

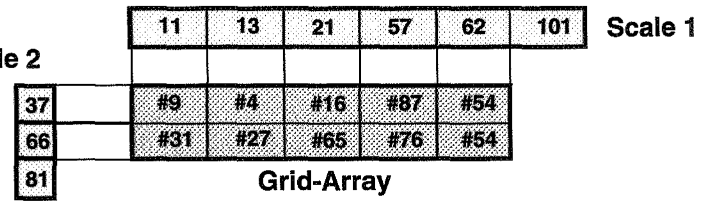


Figure 9. Sample Grid Directory

Consequently, especially *range queries* asking for data within some upper and lower bounds, quite important in multidimensional applications, can be answered straightforward.

The implementation of the grid directory is divided into two classes, a driver class and a directory administration class. Furthermore, the dimension chosen for splitting if the directory has to be extended can be determined according to different strategies like for example round-robin, fixed or toggle, therefore allowing to distinguish between more often needed dimensions with higher granularity scales and less often needed dimensions with lower granularity scales.

7. Summary

Basically, the development of the CUBEStore storage manager included two big tasks: Firstly, a design flexible enough to fit the needs of different SSDB applications had to be developed. This design was presented in section four. To prevent the description from becoming too abstract, two possible ways of realizing such a configuration were added. Nevertheless, further work in this area, especially in defining a more sophisticated partition administration than the basic CUBEIniFile mechanism might be helpful, but has to be done in close connection with the development of the CUBESTAR run time system ([13]).

The second part was to show, in the implementation section, that this design could be realized. As this implementation was only a prototype, several interesting concepts could not be realized. In this context, a completely relational database interface in addition to the special purpose design presented here might be useful.

Despite these remarks, the design and implementation of CUBEStore, a multidimensional storage manager, has been described, which was developed with respect to the needs of SSDB applications. CUBEStore offers a useful basis for further developments in this area.

References

[1] Bauer, A.; Lehner, W.: The Cube-Query-Language for Multidimensional Statistical and Scientific Database Systems, in: *5th International Conference on Database Systems For Advanced Applications* (DASFAA'97, Melbourne, Australia, April 1-4, 1997), pp. 263-272

[2] Bernstein, P.A.: Middleware, in: *Communications of the ACM, 39(1996)2*, pp. 86-98

[3] Codd, E.F.; Codd, S.B.; Salley, C.T.: *Providing OLAP (On-line Analytical Processing) to User Analysts: An IT Mandate*, White Paper, Arbor Software Corporation, 1993

[4] Colliat, G.: OLAP, Relational, and Multidimensional Database Systems, in: *SIGMOD Record, 25(1996)3*, pp. 64-69

[5] Cabrera, L.-F.; Steiner, S.; Penner, M.; Rees, R.; Hineman, W.: *ASDM: A Multi-Platform, Scalable, Backup and Archive Mass Storage System*, Research Report RJ 9936, IBM Almaden Research Center, San Jose, CA, 1995

[6] Davis, G.; Rew, R.: NetCDF: An Interface for Scientific Data Access, in: *IEEE Computer Graphics and Applications*, 1990, pp. 76-82

[7] Eggers, S. J.; Olken, F.; Shoshani, A.: A Compression Technique for Large Statistical Databases, in: *7th International Conference on Very Large Data Bases* (VLDB'81, Cannes, France, 1981), pp. 424-434

[8] Fortner, B.: *The Data Handbook*, Springer Verlag, New York, 1995

[9] N.N.: Informix Universal Server, Product Information, Informix Corp., (http://www.informix.com/), 1997

[10] Lehner, W.; Ruf, T.; Teschke, M.: Data Management in Scientific Computing: A Study in Market Research, in: *International Conference on Applications of Databases* (ADB'95, Santa Clara, California, Dec. 13-15, 1995), pp. 31-35

[11] Lehner, W.; Ruf, T.; Teschke, M.: Improving Query Response Time in Scientific Databases using Data Aggregation - A Case Study, in: *7th International Conference and Workshop on Database and Expert Systems Applications* (DEXA'96, Zürich, Switzerland, Sept. 9-13, 1996), pp. 201-206

[12] Lehner, W.; Ruf, T.; Teschke, M.: CROSS-DB: A Feature-extended multi-dimensional Data Model for Statistical and Scientific Databases, in: *5th International Conference on Information and Knowledge Management,* (CIKM'96, Rockville, Maryland, Nov. 12-16, 1996), pp. 253-260

[13] Lehner, W.; Teschke, M.: On the Architecture of a Multidimensional Database System for 'Decision Support' Applications, *Technical Report TR-97-132*, University of Erlangen-Nuremberg, Erlangen, 1997 (in German)

[14] Menon, J.; Treiber, K.: *Daisy: Virtual Disk Hierarchical Storage Manager*, Research Report RJ 10075, IBM Almaden Research Center, San Jose, CA, 1997

[15] Nievergelt, J.; Hinterberger, H.; Sevcik, K.C.: The Grid File: An Adaptable, Symmetric Multikey File Structure, in: *ACM Transactions on Database Systems 9(1984)1*, pp. 38-71

[16] N.N.: *Oracle Spatial Cadridge*, Product Information, Oracle Corp., (http://www.oracle.com/st/cartridges/spatial/), 1997

[17] Sarawagi, S.: Query Processing in Tertiary Memory Databases, in: *21th International Conference on Very Large Data Bases* (VLDB'95, Zurich, Switzerland, Sept. 11-15, 1995), pp. 585-596

[18] Shoshani, A.: Statistical Databases: Characteristics, Problems, and Some Solutions, in: *8th International Conference on Very Large Data Bases* (VLDB'82, Mexico City, Mexico, Sept. 8-10, 1982), pp. 208-222

[19] Shoshani, A.: OLAP and Statistical Databases: Similarities and Differences, in: *16th Symposium on Principles of Database Systems* (PODS'97, Tuscon, Arizona, May 12-14, 1997), pp. 185-196

[20] Stonebraker, M.: The Design of the POSTGRES Storage System, in: *13th International Conference on Very Large Data Bases* (Brighton, England, Sept. 1-4, 1987), pp. 289-300

[21] Tsichritzis, D.C.; Klug, A.: The ANSI/X3/SPARC DBMS framework report of the study group on database management systems, *Information Systems 3(1978)3*, pp. 173-191

Page intentionally left blank

Petabyte File Systems Based on Tertiary Storage

Marc J. Teller **Paul G. Rutherford**
marc_teller @emass.com paul_rutherford@emass.com
EMASS, Inc.
10949 E. Peakview Ave.
Englewood, CO 80111
Tel: +1 303 792-9700
Fax: +1 303 792-2465

Abstract: In a matter of a few short years, the computing industry has moved from terabyte storage systems to petabyte environments. While manufacturers and implementors alike are struggling with access to 1000's of gigabytes, planners are requesting bids for systems in excess of 1,000 terabytes of data. The requirements for petabyte systems are much the same as for terabyte systems. Access must be the same as disk-based file systems; consequently, it is transparent to applications. In addition, performance should be predictable and meet minimal application requirements. Finally, system management should require few resources. In addition to these operational requirements, the size of the system requires strict attention be paid to the cost of each component. As a vendor of storage solutions, EMASS was interested in validating its AMASS product against these requirements.

This paper documents a project for benchmarking a petabyte storage system residing in our Denver facility. In addition to profiling the ability of AMASS to scale to a petabyte, the study looks at the operation of a large file system, alternate tools for dealing with large file systems, and additional development and research to support petabyte file systems.

Results of the project indicate a high degree of scalability for AMASS up to the initial project-imposed limit of 16 million files and identified specific areas for enhancement of the AMASS product.

1. Introduction

We used to worry about storing and retrieving terabytes (TB) of data. Today, magnetic disk can be used for small numbers of terabytes. In fact, most operating systems can manage these large disk farms as standard file systems. Therefore, these operating systems have become the de facto standard against which higher capacity storage systems are measured.

At higher capacities, tape technology has met the challenge by providing us with high density recording. Standard half-inch cartridges now contain up to 50 gigabytes (GB). A small desk side automated library, using low-cost 30 GB tapes, can easily hold the 20 to 50 cartridges needed for a terabyte of data and fit within the budget of most small departments. Because hundreds of terabytes has become the normal mode of operation for data centers, it is now time to push toward a thousand terabytes (a petabyte) of online data.

While still costly, a petabyte (PB) system requires 20 to 50 thousand cartridges. The system can be automated using available tape libraries from one of the high storage capacity providers such as EMASS or Storage Technologies. Given that the hardware is available, the real question is management software. Is there software that can scale to this capacity? Some management software is precluded because the total number of bytes stored is inadequate; other software is excluded because the number of files managed is inadequate.

This paper presents the results obtained in creating a petabyte file system, the scalability of the metadata and operational characteristics of a large file system, and future work needed for petabyte file systems.

2. The Problem

At the 1996 NASA Goddard Storage Conference, a panel of large system users commented on the lack of storage management software that could address their needs. Many were still considering in-house development or were feeling forced into less than adequate systems. Most were using Hierarchical Storage Management [2] (HSM) products such as DMF from Silicon Graphics (formerly Cray Research), UniTree from UniTree Software Inc., or FileServ from EMASS. The panelists were concerned both with long term availability and with scalability of the products for petabyte file systems.

A list of some of the requirements, by no means intended as exhaustive or addressed by this project, for a system solution to the issues on this area can be summarized as follows:

- High capacity media.
- Large number of files (tens of millions).
- Large amount of total data (multiple petabytes).
- Stable, reliable, and easy-to-use software. For example, file system view of storage allows easy programming and portable applications.
- Data is kept online. Offline storage, except for backup copies, is unacceptable.
- Relatively predictable data access times.
- Multiple host platform support. This allows server platform selection by the end-user.
- Performance able to meet, or exceed end-user expectations.

2.1. Media Technology

Given current technology, the least number of cartridges that can reasonably be automated yet still provide petabyte storage capacity is 6,667. Table 1 provides examples of the media types applicable to this size system.

Vendor & Drive Model	Media Type	Capacity /Cartridge[†]	Cartridges /PB	Native Transfer Rate	$/GB[*]
Ampex DST312/M	DD-2	150GB	6,667	15 MB/s	0.84
StorageTek Redwood	SD-3	50GB	20,000	11 MB/s	1.80
Quantum DLT7000	CT-IV	35GB	28,572	5 MB/s	2.83
Sony SDX-300C	AIT-1	50GB	20,000	3 MB/s	2.00
magnetic disk drive	n/a	23GB	43,479	13 MB/s	150.00
magnetic disk array	n/a	176GB	5,682	18 MB/s	660.00

†This represents native capacity, so does not include any compression the drive may support.

*This cost reflects only the media for tape devices. Because a tape drive is used to service all media, there is not a one-to-one correlation of drive to capacity as with a disk drive or disk array.

Table 1. Media Possibilities

For comparison purposes, the table shows standalone disk drives as well as drive arrays. But even if we used a 1 TB-sized array, we would still need a thousand of them to store 1 PB of data. In any case, attaching the 1,000 arrays would pose a significant problem.

Cost, access profile, and transfer rate must be considered. Not all automated library vendors support the different types of drives. So while the actual library choices might be limited, solutions are available.

2.2. Number and Size of Files

System administrators are concerned about the amount of disk space needed to represent the files in a petabyte file system. Although one cannot predict the exact mix of file sizes, an estimate of the range of files a petabyte system must handle can be made. If the average file size is 1 GB, the number is a million files. For this project, 16 million files were created using a single 62.5 MB set of data blocks.

In a regular file system the actual space occupied by files must take into account the space used by the inodes and indirect blocks needed to access the file data. The size of the inodes range from 128 bytes to 512 bytes depending on the vendor. If HSM is added, database information increases to track the file location. Typically this is between 512 and 1024 bytes of data per file. In an archive file system like AMASS, the database contains the file metadata and totals 120 bytes per file. In a regular file system with an HSM, about 20 GB will be needed to hold the metadata. With AMASS, about 4GB will be needed. However, even 20 GB is not unreasonable.

Another issue in the computing industry is the size of the largest file that can be stored in a file system. Extended file systems support at least 40 bits of file address, or 1 TB. File systems are moving toward full 64-bit addressing in the next year or so. This gives us the ability to address 18.5 exabytes in a single file.

2.3. Data Access

While magnetic disk has a relatively uniform access time, tertiary storage devices (specifically tape, a sequential access device with slower positioning time) have a decidedly non-uniform access time. Access to data on tertiary storage is governed by a combination of the following:

- Time for a drive to become available.
- Time to unload media in the selected drive.
- Time for the robotic arm to fetch the appropriate media and load it into the selected drive.
- Time to position the media to where the data resides [1].
- Time associated with accessing metadata to obtain addressing information.

In order to minimize the impact of the above times, systems can employ a cache (usually based on magnetic disk) to hide the overhead by managing *readahead* and *residency* of data. The caching behavior is important due to the difference between disk and tape access characteristics. Contention for tape drives becomes a significant issue in a heavily loaded system. For reads, there is not much that can be done. The data is on tape so the requestor must wait for a drive to become available. For writes, a disk based cache can be used to hold data until tape I/O can be completed.

2.4. Issues With Large Datasets and Metadata

When a file system begins to store a very large number of files or datasets, or a very large number of directories, the time it takes to access file data becomes a growing concern. This is usually (with the exception of simultaneous user access interaction) the result of acquiring the metadata that describes the location of the file data on the physical media. It does not matter whether the file system stores metadata in inodes (as in standard UNIX® file systems) or stores metadata in a database (as in AMASS [3]). While the inode-based systems suffer from indirect block lookup and read operations (even extent-based file systems can suffer from extent metadata management), database systems can suffer from scanned lookup times depending upon the nature of database operations and how file keys are used. As a result of these concerns, the project focuses on characterizing metadata access performance as an AMASS file system grows from initialization to 16 million files referencing over a petabyte of data (see Section 4.1.).

3. Project Goals

This project is designed to provide answers to the questions raised in the previous discussion. As a software vendor, we are interested in how well our AMASS product meets the needs of a petabyte system.

AMASS – Archival Management And Storage System – stores and retrieves files using robotic libraries, jukeboxes, and standalone drives. AMASS provides transparent, direct access, to the files stored in the AMASS file system. AMASS makes the drives and media – normally considered offline storage – appear as a single, online, logical device with a single, mounted, file system. Therefore, the extensive storage provided by a library appears as one large file system. Because AMASS is implemented at the virtual file system (VFS) layer of the server's operating system, it is transparent to application software.

The primary goals of the project are to verify scalability of AMASS to 1 petabyte, to profile operation of user commands with large datasets, to identify product enhancements for petabyte size systems, and to create a base system for exploration of multi-petabyte systems.

4. Project Definition

The project being undertaken is designed to evaluate two scenarios using AMASS as the storage management environment. In phase I, the petabyte ($1x10^{15}$ bytes) storage requirement of very large systems and the performance impact, from an end-user perspective, is analyzed. In phase II (to be described in a future paper, see Section 7.), the data access characteristics of AMASS when executing random access requests to data files that range from 100 GB to 1 TB in size and span media volumes within the library, will be analyzed.

4.1. Large Dataset Storage

In an effort to characterize the performance plateaus in a file system as it approaches large numbers of files, directories, and amount of data stored, a test system has been setup that is designed to ramp to a data storage in excess of 1 PB and over 16 million files. Because the only measure of interest is the relative performance of the file system database against the scale of the file system population, a slow platform is just as useful as a fast platform. This portion of the investigation is running on a Sun Microsystems SPARCclassic with Solaris 2.5 and AMASS Version 4.7.1. The media choice is not germane as data performance is being evaluated in a later phase of the project. In addition, the amount of data being stored is not germane to the scalability of metadata in the file system because each byte of data is directly addressed in AMASS, unlike typical UNIX file systems where no indirect addressing is used. This provides a very consistent access time to any byte of data in the petabyte store except for the issues raised in Section 2.3.

A single set of data blocks is used for all 16 million files. The data descriptors for each file reference the same blocks on the media. However, links are not used so that the system appears to have actually stored a petabyte of data. This saves the $7.1 million for media (100,000 Magstar® cartridges) and 1.6 years to write the files (at 20MB/sec. sustained).

Figure 1 illustrates the namespace created. It consists of 512 directories, under the root directory for the library. Each of these directories has 128 subdirectories each of which contains 256 files. In other words, there are 512 sub-trees with 128 directories and 32,768 files (128 * 256; for a total of 512 * 128 * 256, or 16,777,216 files).

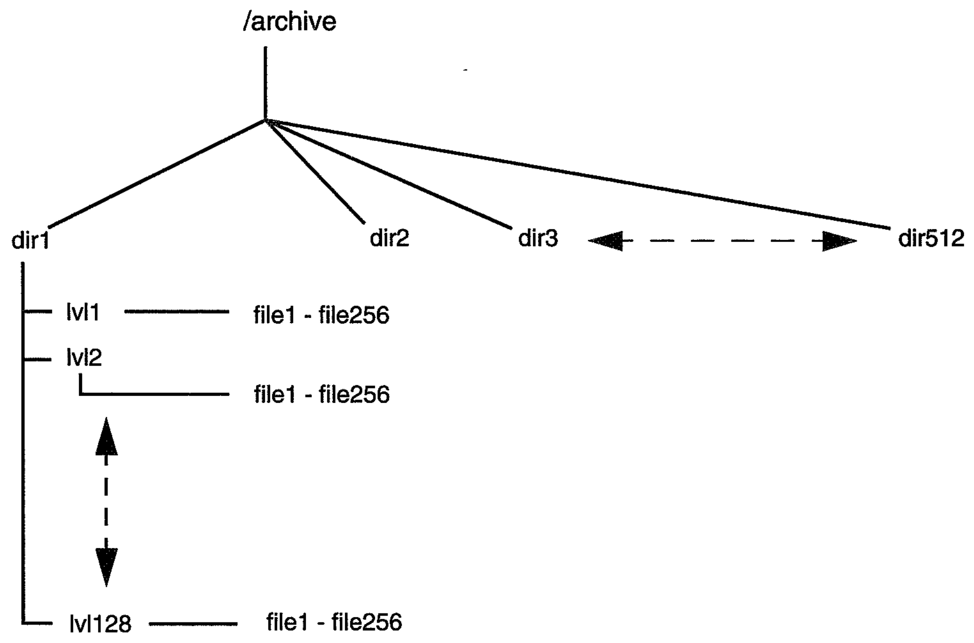


Figure 1. Namespace Organization

The structure of the namespace was chosen as a relatively easy way to increase the number of files to the goal of 16 million. Since the namespace is built using an import facility in AMASS, the ability to create 32,768 files (128 directories each containing 256 files) with each import simplifies the namespace creation.

Because the metadata for the file system is accessed by a keyed lookup, the actual namespace organization does not significantly impact performance and consequently, is not an issue. Although it might be possible to construct an unbalanced namespace layout that could affect the test results, this would not be a real-world scenario since most systems have a more balanced structure as did the test scenario.

Phase one of the project's performance evaluation is oriented towards metadata access since AMASS uses direct access for data, after the metadata has been retrieved. This has been accomplished by using standard UNIX® commands that query the namespace or the file/directory attributes without accessing the actual file data, such as `ls(1)`, `find(1)`, and `pwd(1)`. The performance data is gathered after a group of four trees are created, resulting in the addition of 131,072 files across 512 sub-directories to the namespace.

The characterization script uses a ten iteration loop with a recursive '`ls`' beginning in the 64th subdirectory of the first tree, followed by a '`find`' beginning in the 100th subdirectory of the 3rd tree, and ending with a '`pwd`' in the 127th subdirectory of the 2nd tree. The ten iteration loop is followed by a five iteration loop with the same commands, in the same subdirectories running in the last tree created. All results are averaged and use the UNIX® `time(1)` command for reporting the *real* execution time of the commands. This approach was chosen so that the end-user perspective can be examined and all of the "benchmark" tests that have been received from customers are also based on timing command level scripts.

5. Results

5.1. Metadata Performance

Figure 2 shows essentially flat performance, indicating a very scalable system. The absolute performance values can be reasonably tied to a slow, minimally configured workstation. The relative performance shows the scalable nature of the system. The graph data points include a 90% confidence interval.

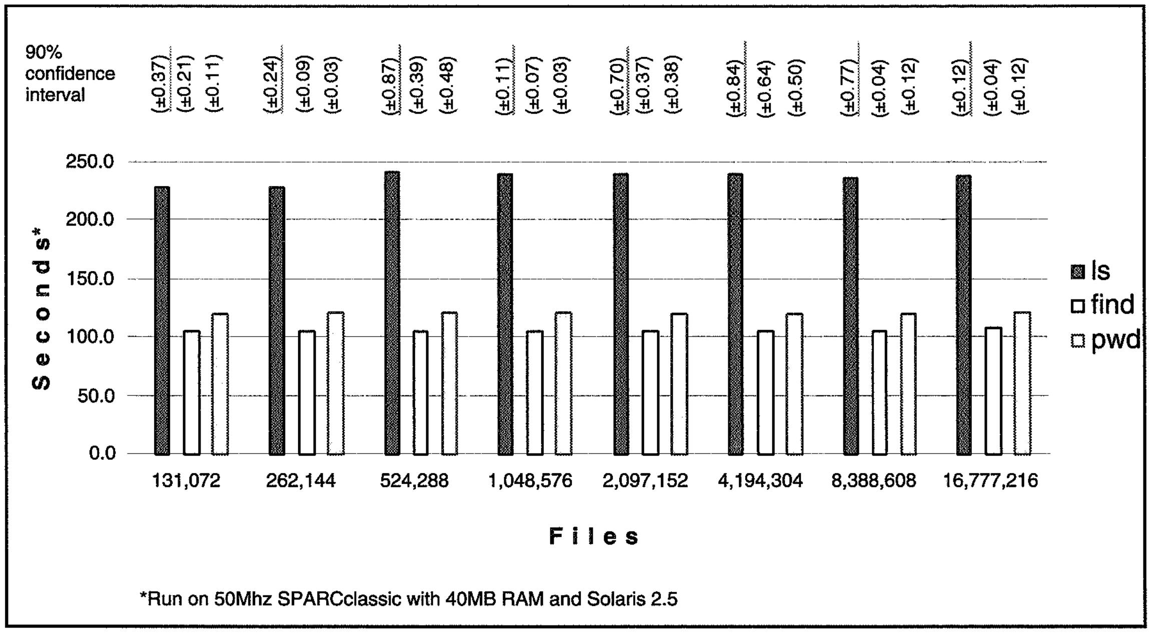


Figure 2. Metadata Test Results

As can be seen, a 'pwd' command for a location 127 levels deep in the tree takes about 120 seconds in real time. This timing was gathered by executing the standard UNIX® `pwd(1)` command that performs recursive `stat(2)` calls. In other words, the standard file system paradigm is at work. AMASS implements a file system view of the library to allow unmodified applications to run using the media stored in the library. While this is extremely beneficial in terms of a customers investment in application code, it does incur some overhead, including kernel <-> user space context switches, to allow redirection of file system requests to the AMASS code.

A future phase of the project will compare the scalability of AMASS to that of a native file system.

5.1.1. Database direct-access for commands

To see if performance could be improved using direct access to metadata, we wrote an AMASS 'print working directory' command. Figure 3 shows the results of comparing the two commands, the standard UNIX password command and the new AMASS password command. As in Figure 2, the data points include a 90 percent confidence interval.

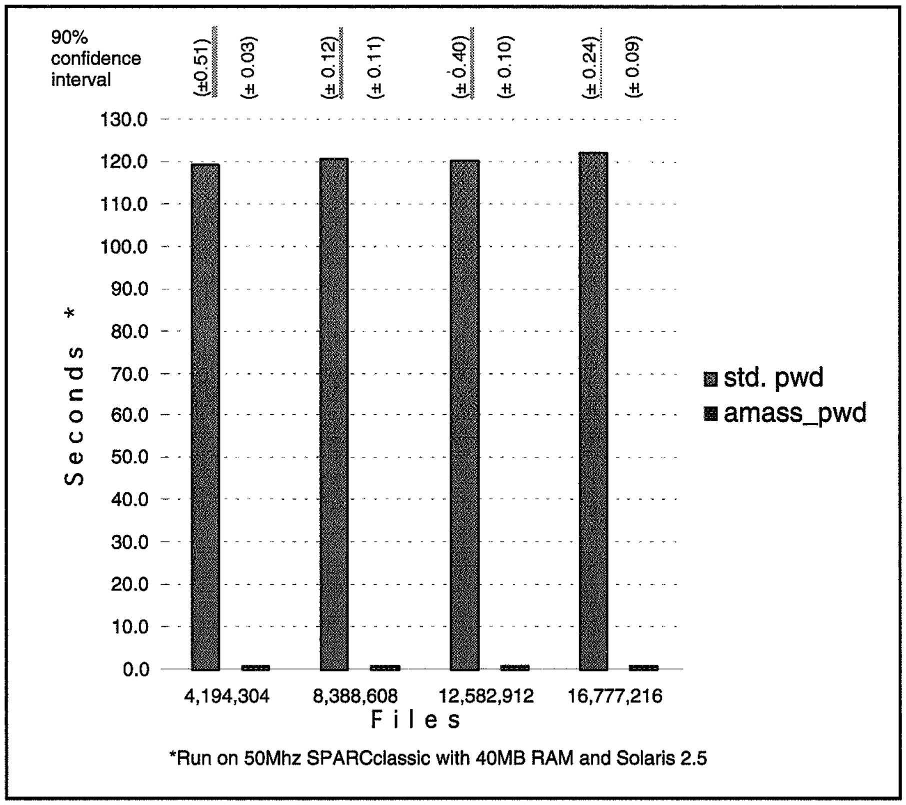


Figure 3. PWD Comparison

As the graph shows, the performance of direct metadata access significantly out-performs the performance of file system access. However, this increase sacrifices standard command transparency. For environments that require increased performance, an Application Programming Interface (API) is available for AMASS.

Analysis of the standard UNIX `pwd` command indicates that the main loop for walking the namespace tree consists of:

- open("./../../...")
- fcntl (sets 'close on exec flag' for open file)
- fstat -- 2 calls on open file descriptor
- getdents -- 2 or more, depending on the size of the directory
- close

However, the `amass_pwd` command uses a namespace walking loop consisting of:

- lseek (1 for main record, 1 for name record)
- read (1 for main record, 1 for name record)
- brk -- 2 calls of 0x2000 each

AMASS uses a Virtual File System (VFS) "driver" to re-direct file system calls to a user-level daemon that queries the database for metadata information. The data is returned to the caller after it has been transformed into a compatible format. Therefore, standard file system calls incur the overhead of this redirection and transformation, including context switching between kernel-level and user-level. While this overhead is negligible for I/O operations (due to the implementation), it can become significant in metadata-only operations due to database lock contention.

In the amass_pwd case, the AMASS database points from the directory and file record to its name record, so neither directory searching (via getdents) nor open calls are required. Bypassing the file system interface and directly accessing the database eliminates the most significant portion of the overhead.

In addition to the pwd command, we evaluated the benefits of using a variant of the find and ls commands. Ultimately, when a file system scales to many millions of files, the standard metadata access paradigm, which uses stat and getdents to access inode and directory information, has an unacceptable overhead associated with it. Therefore an alternative path to this information is required to achieve high performance goals.

While the user of such a system can fine-tune the applications used to manage the storage system, it is reasonable for the storage software vendor to make variants of the typical UNIX command set available for these sites.

6. Additional Requirements

Based on the operational experience during the project, this section outlines a set of requirements for any solution to address the petabyte storage world, and presents approaches for modifying AMASS to meet these requirements.

6.1. Data Integrity and Technology Migration

With such an enormous amount of data being managed, a method to allow protection of data from loss due to media failure must be provided. EMASS refers to this as Total Data Life Management (TDLM) since it designed to manage data integrity during the entire lifecycle of the data. As each volume of media is unmounted/unloaded, heuristics are updated to capture retry counts and ECC codes. This data is examined and suspect media are selected for data copy operations. When the data is copied, the original, suspect, media would be made inactive and the operator notified so it can be removed from the library. This process helps to ensure the integrity of data stored within the petabyte environment. As archives grow in size, many volumes remain unaccessed. Programs referred to as “tape sniffers” access each volume in the library on a rotating basis to randomly read files. This activity results in collecting tape error rates for each volume in the file system on a periodic basis. The rates determine which volumes should be copied before any data loss is incurred.

As new drive technologies become available, the question of how a site moves existing data from older media to new media becomes increasingly important. When the amount of data is only a couple of terabytes, a batch data transcription process is probably acceptable. On the other hand, when the amount of data exceeds this number or when the site cannot tolerate any data downtime, another method must be found. Although transcribing a petabyte of data is unacceptable as a foreground activity (see Section 6.3. for an estimate of the time involved), every large site will eventually have to transcribe data as older drives are either retired or no longer supported.

One approach would be to treat the older media as read-only. As this data is updated, it is copied to new media, and the older version is marked as invalid. Over time, much of the data would be transcribed in this manner. Any unmodified, and therefore non-transcribed, data is handled in a batch mode in order to complete the migration.

Another approach, and probably one more likely to be used, is to provide a set of tools to assist administrators in planning and executing the migration. These tools would provide estimates of the time required to migrate all of the data from the older media to newer media and mechanisms for this migration without requiring or causing any service interruption for users. Server platform and drive independence is a great help as the site is not constrained in its decision on which technology to adopt.

This leads into the metadata standard for describing data that is currently under development (see Section 6.7.).

6.2. Extensible Metadata

When a site is managing a petabyte of data, any user-defined keywords for searching the available data is an aid to the user community. One approach is to use well understood catalog tools on top of AMASS. Another approach, is to provide the ability for users/applications to manipulate metadata that is opaque to AMASS. A pair of AMASS API (Application Programming Interface) routines could provide a way for client applications to set and retrieve metadata that AMASS is currently unaware of. The metadata could be correlated with either a file or with a directory, providing flexibility in organization.

6.3. Backup and Recovery

After a petabyte, or more, of data has been stored, how do you back up this file system? Typically a full backup essentially duplicates the storage size. If you assume a sustained throughput, including media exchange times, of 20 megabytes/second, a 1 petabyte backup would take 1.6 years to complete! Add to this the cost of media, plus the usually overlooked impact of having a reduced number of drives available to respond to client requests during this 1.6-year long backup! The validity and scope of this petabyte backup is staggering — to say the least.

For petabyte file systems, a revision history paradigm may be more efficient than the more traditional image backup paradigm. In a revision history paradigm, only modifications to a baseline image are copied. This is essentially a versioned, incremental backup. Unlike a traditional backup, only the changes made to files since the last backup are copied, the entire file is not. The initial backup, which forms the baseline image of each file, could be created with a file replication feature run when the file is first placed in the file system or with a manual copy run in batch mode during system idle time. The net effect of this approach is to reduce the impact of backing up the file system and constrain the amount of data needed to ensure the recreation of the dataset in case of disaster. The nature of incremental backups, however, is that recover is also incremental from the last full backup.

However, from an archival storage perspective, with slow changing data, the real issue is disaster recovery. In this scenario, the use of multiple archives is optimal, along with replication of the data streams to both archives when data is ingested. This provides replication of data, on the fly, as opposed to batch mode, which is too expensive in time and resources.

Another option to consider is tape RAID where data is spread across multiple tapes, with a parity tape, so the loss of any single tape does not result in the loss of any data. The liability of using this option is the mount time needed to retrieve a file. Any option used to address disaster recovery — must pay a price in increased overhead.

6.4. Startup and Consistency Checks

Another feature required for petabyte storage management is startup and recovery times. With a database almost 3.9 gigabytes in size, checking the consistency of the file system against the database takes a long time to complete — even if the database has no errors to correct. It is critical that this process be optimized. Some of the optimizations being investigated (given that the process is CPU-bound) are:

- Multi-threading the utility as most systems with a very large dataset environment will likely be hosted on multiprocessor systems.
- Optimizing the code for larger memory machines. AMASS can execute on systems with as little as 24Mbytes of RAM; however, this results in a coding style that makes use of redundant code loops to avoid a larger memory footprint. Most multiprocessor hosts running AMASS have significantly larger memory configurations.

6.5. Long Runtimes

Because management utilities, like `df`, scan huge amounts of data, running this type of command can raise concern about the health of the system when it seemingly causes the system to hang. The solution may be for the software vendor to provide a set of utilities that duplicate the functionality of the file system-based UNIX tools, yet employ a shortcut to access the file system metadata directly, improving the performance of some commands (see Section 5.1.1. and Figure 3).

6.6. System Utilities

As with some standard UNIX utilities, some of the AMASS commands use fixed-width formats in reporting size, block number, and other information that might overflow in a petabyte storage environment ([5] describes some issues in migrating to 64-bit file system environments). These need to be identified and modified to support the larger environments. The most likely approach is the one followed by most UNIX vendors: a new set of commands, replacing the older ones, can be installed which will handle the larger data fields needed to correctly present data about the dataset store.

6.7. Standards

An aspect of the technology migration issue is being addressed by the AIMM [6] Standards committee C21.1. Using current technology, petabyte systems require many thousands of cartridges. As mentioned in Section 6.1., it is impractical to move this amount of data from one file management system to another and, in the process, force the data to be rewritten in a different format. The C21.1 committee is working to define a metadata standard that describes the data contained on media. EMASS is actively participating in this effort.

When the storage management system at the source site provides a standard metadata, the storage management system at the destination site can import the metadata and have sufficient information to access the data contained on the original media [7].

7. Future Work

To study the data access characteristics of AMASS with large (1TB) file sizes, a second phase of the project will be conducted. In this phase, an AML/S library [4] equipped with a pair of IBM Magstar 3590 drives and loaded with 120 tape cartridges will be connected to a Silicon Graphics Challenge L running IRIX 6.2 and AMASS Version 4.9.1.

A file creation task will be used to write the 1 terabyte file with drive compression disabled (thereby allowing a true data performance characterization). The task will report the amount of data written every hour resulting in a 'write rate' profile of a single writer across the entire terabyte of data. It should be noted that media exchange times are included in the profile (the terabyte file should require 100 cartridges at 10 gigabytes each).

After the file has been completely written, a pair of reader tasks will read random, non-overlapping, regions of the file. These tasks will report data access times, drive utilization, and media exchange wait times. This test scenario does not investigate resource (drive or volume) contention resolution but it does establish a reasonable profile of I/O activity. Variations from the observed performance can be affected by the nature of the cache environment, the number of drives, and the way in which the data is organized within the namespace.

The reader tasks are designed to scatter read requests across media volumes (never the same media volume by both readers). Ideally, each reader would have a drive available in which to load the media volume containing the requested data region (even if previously accessed media needs to be unloaded first). Given that no more than one reader task would be accessing a specific piece of media (due to the gap between regions read), there should be no read request queueing except for media exchange times.

8. Conclusions

As phase one of this project clearly demonstrates, there IS storage management software available that can scale file systems to petabyte capacity. As petabyte-sized file systems are typically used for long-term data storage and retrieval, it is essential that library and media types be vendor-independent to allow for technology migration during the lifecycle of the data. While some operational issues need to be improved, such as data integrity and administrative tools, these issues can be resolved as the file system grows to a petabyte in size.

9. References

[1] Sarawagi, S., "*Database Systems for Efficient Access to Tertiary Memory*", Fourteenth IEEE Symposium on Mass Storage Systems, September 1995.

[2] Woodrow, T., "*Hierarchical Storage Management System Evaluation*", Third

NASA Goddard Conference on Mass STorage Systems and Technologies, October 1993.

[3] Sarandrea, B., "*Storage System Architectures and Their Characteristics*", Third NASA Goddard Conference on Mass STorage Systems and Technologies, October 1993.

[4] Produced by EMASS, Inc. See *http://www.emass.com* for information on the library.

[5] Shaver, D., Schnoebelen, E., Bier, G., "*An implementation of Large Files for BSD Unix*", USENIX Winter Conference Proceedings, January 1992.

[6] Association for Interactive Media and Marketing, *Metadata for Interchange of Files on Sequential Storage Media Between File Storage Management Systems (FSMS).*

[7] For more information on this effort, please see the following URL:
http://ses2.ses-inc.com/~joelw/

Page intentionally left blank

Internet Protocols for Network-Attached Peripherals

Steve Hotz, Rodney Van Meter, and Gregory Finn
Information Sciences Institute
University of Southern California
Marina del Rey, CA 90292
{hotz,rdv,finn}@isi.edu
tel +1-310-822-1511
fax +1-310-823-6714

Abstract

This paper presents our thesis that the advantages of the internet protocol framework make it the best choice for communications protocols to, and between, network-attached peripherals (NAPs). Moreover, the IP suite is more appropriate than the specialized protocol stacks being developed for commercial NAPs. The benefits of using IP include support for heterogeneous network media, wide-area connectivity, and reduced research and development effort. We examine the issues for use of the internet protocols (TCP/UDP/IP) for NAPs, address commonly held concerns regarding its performance, and describe the Netstation project's prototype implementations of IP peripherals.

1 Introduction

Netstation is a research system architecture based on a switched gigabit network as a system backplane [10]; all of the major subsystems are network attached peripherals (NAPs). A network-attached display has been prototyped, and a software-emulated network-attached disk has been implemented. A camera NAP is also under development, based on the same motherboard as the display.

One of the primary decisions in designing such a system is the choice of a communication infrastructure. To provide maximal ubiquity and independence from specific media technologies, Netstation component communication is based on the TCP/IP protocol suite. This design choice is controversial, in that most commercial NAP efforts (e.g., Tandem's ServerNet [15], Fibre Channel and HiPPI disks and disk arrays) and some research projects (e.g., Minnesota's GFS [21]) use special purpose protocol stacks optimized for performance on specific media. Among research systems, the most similar effort is the High Performance Storage System NAP at Lawrence Livermore [26]. Digital's Petal [18] uses UDP/IP over ATM as part of a large distributed virtual disk system implemented in general-purpose hosts. CMU's Network Attached Secure Disk [13] does use IP for its higher level protocols which are derived from NFS, however, they provide a network "object store" rather than a block-level NAP. The two projects most closely related to Netstation are MIT's ViewStation [1] and Cambridge's Desk Area Network [14], both of which use ATM to interconnect multimedia peripherals.

The main reason cited for choosing a specialized protocol rather than the media-independent TCP/IP approach is the latter's perceived lack of performance, or the expense to achieve the required performance. We believe that the advantages of TCP/IP merit further consideration for its use in NAP applications, and that the performance concerns are based on inaccurate comparisons with specialized protocol performance. We argue this by examining protocol functionality and its overhead, and by presenting results of our implementation experience.

This paper presents the case for TCP/IP based NAPs. Section 2 discusses the current trend for NAP protocols, provides an overview of the IP protocol suite, and discusses the the use of TCP/IP for NAPs. Section 3 addresses the functionality and potential performance problems of using TCP/IP for NAPs, and Section 4 discusses implementation issues that are often perceived as protocol performance limitations. Section 5 describes the Netstation prototype implementations and their use of TCP/IP, and our conclusions are summarized in Section 6.

2 Protocols for Network Attached Peripherals

Networks such as HiPPI, SSA and Fibre Channel are becoming the access technology of choice for peripherals such as disk drives, tape drives and disk arrays. These networks scale better than traditional I/O channels, connecting more devices over greater distances and providing greater aggregate bandwidth. However, these networks require more complex protocols than are required for traditional bus-based channels such as SCSI.

The NAP community, in most cases, has chosen to use specially developed protocols similar to channel access protocols rather than existing network standards such as TCP/IP, because of perceived differences in functionality, focus, complexity and especially performance. We reason that most of these concerns either reflect misunderstanding of the IP suite or are being met as the suite evolves.

We further argue that the benefits of using IP, including wide-area connectivity, cross-media bridging and reduced research and development efforts, are substantial. Specialized protocols simply do not address inter-LAN communication, which will be important as machine rooms integrate new LAN technologies into increasingly heterogeneous computing environments, and as new uses for sharing NAPs over a wider campus area emerge.

Therefore, we believe that IP is the best choice for storage device peripherals, and should be the protocol selected by NAP system architects.

2.1 IP Framework Overview and Definitions

Throughout this paper we use the terms "IP protocol suite", "IP protocol framework", or "TCP/IP protocol suite". We use these terms interchangeably to refer to a set of protocols that provide the functionality of the network layer and transport layer of the OSI seven-layer reference model. Specifically, this set of protocols are the DARPA Internet Protocols, and they include the Internet Protocol (IP) at the network layer, and both the Transmission Control Protocol (TCP) and the User Datagram Protocol (UDP) at the transport layer. It is important to note that there is exactly one network layer protocol, while there are multiple transport layer protocols. UDP and TCP are the most common and thus will be discussed in more detail. However, the framework does allow for other transport layer protocols.

The network layer protocol, IP, provides the basic functionality of exchanging packets of data between any two entities connected to an internet, i.e., a network of networks which may include many different physical media and link layer protocols. The primary philosophy is that this protocol must be ubiquitous: IP runs on everything, and everything runs on IP. IP is intentionally simple and provides very basic functionality which allows it to be implemented on even the simplest of lower-layer protocols. It provides no guarantees on data delivery other than if a packet is delivered, it is delivered to the end host with the specified address. To this end, IP provides a common addressing scheme, a simple header checksum to guarantee that the addresses are not corrupted, and a mechanism to fragment and reassemble packets to accommodate transmission over link layers with different maximum packet sizes.

The transport protocols, TCP and UDP, build on top of IP, exchanging their own headers and data payloads using the basic facilities provided by IP. These protocols use different mechanisms to provide various levels of end-to-end functionality. UDP provides simple user-level access to the basic unreliable service provided by IP, and as such, it adds very little overhead to the exchange of data. On the other hand, TCP provides for a reliable, in order, byte stream and is by far the most complex of the three protocols discussed here.

There exists other related and supporting protocols, such as an Address Resolution Protocol (ARP) for mapping IP addresses to link-layer addresses, and various routing protocols to exchange information so that intermediate internetwork nodes can determine the correct disposition and forwarding of IP packets to their destination. However, these other protocols do not directly impact the per-packet data processing overhead at end systems, and consequently are only mentioned briefly in this paper.

2.2 IP for NAPs

Choosing IP as the communication infrastructure for NAPs alleviates the problems of committing to a protocol suite which is tied, for the most part, to the choice of physical media, e.g., Fibre Channel or ATM. This provides a growth path that is unconstrained by the future development of a particular technology. In the same vein, IP allows for cross-media bridging with minimal incremental effort. Cross-media bridging will be useful given increasingly heterogeneous computing environments, allowing transparent interoperation among different types of networks. Finally, the wide area connectivity that is IP's strength opens up new functionality for peripherals, for example, remote mirroring of disk drives and remote backup.

Using IP will allow NAP developers to exploit the large existing body of research and development in flow control, routing, congestion control, and reliability. This will reduce R&D effort, as well as allowing quick integration of emerging features such as resource reservation and real-time protocols. The effort to adapt and optimize IP implementations for the NAP environment should prove less costly than reinventing pieces of the network solution as new NAP requirements are realized.

We believe the concerns about IP performance are founded on comparisons between IP and specialized protocols, and that the performance differences are more dependent on implementation and environment than on the complexity of the protocols themselves. IP performance measurements are typically made in general purpose (and often outdated) operating systems with protocol implementations tuned for wide-area communication using relatively small packets, while NAP protocol performance benefits from a low-cost embedded operating environment, large data packets, and specialized protocol coprocessors.

These concerns about IP performance and complexity should prove to be non-issues. The issues and differences can be classified and addressed in a general way as follows:

- **Protocol functionality required by NAPs, already provided by IP.** NAPs will benefit from the functionality such as segment size negotiation, reliability, flow control, and cross-media bridging already provided by IP. These common functionality requirements are the main motivation driving the use of IP for NAPs.

- **IP protocol functionality and complexity not required by NAPs.** Concerns about the performance impact of additional "baggage" functionality can be addressed by fast-path and common-case implementations. As evidence, we note that the combined transport (TCP) and network (IP) processing of common-case packets has already been optimized to approximately 200 machine instructions, which would cost 2-8 microseconds on a typical NAP-embedded microprocessor.

- **Protocol functionality required by NAPs, not provided by IP.** While NAPs can take advantage of IP to avoid re-inventing a variety of technologies and functionality, there are clearly cases where the NAP community can (and must) extend the IP framework. Support for application-layer framing and fast application demultiplexing might be addressed by transport-layer options, or may require a new transport protocol to complement the existing UDP and TCP. Section 3.3 discusses these issues in more detail.

- **Implementation and operating system issues.** Many of TCP/IP's known performance problems come from outdated implementations or restrictions imposed from outside, such as data copies through the UNIX socket API and ethernet's small packet size limit. This overhead is considerable compared to the actual fast-path protocol processing. These problems will disappear as the protocol stack is moved into the highly-optimized embedded NAP environment. Section 4 discusses these issues in more detail.

We reason that as systems move to larger, more complex switched networks for I/O, some sacrifice of performance is the inevitable result. However, IP has no inherent performance penalty relative to other protocol choices; the loss is entirely attributable to managing the additional complexity. In this environment, IP offers significant advantages and few drawbacks. IP, therefore, should be the network protocol of choice for network-attached peripherals.

2.3 NAP Implementations of the IP Protocol Suite

We do not expect that performance-critical NAPs will run based on current IP implementations, or without additional development of the IP protocol suite. Rather, there is an open IP framework for network communication, and NAP networking development could benefit from working within this framework. Within this framework there are both design changes that can not be made, and considerable flexibility to accommodate NAP requirements.

The discussion in Section 2.1 points out those parts of the IP suite that will not change, specifically, the IP network layer protocol itself. To be an IP NAP, a device must implement the IP protocol.

There are three general approaches that NAP developers can take to adapt the IP protocol suite for their devices:

- **Use protocol options.** Protocols of the IP suite, including IP and TCP, define protocol options that allow the protocol to provide additional required features on a per-use basis, e.g. per IP packet or per TCP connection. If existing options do not provide for efficient NAP communications, new options can be introduced within the IP protocol framework.

- **Optimize implementations for expected NAP usage patterns.** IP protocol implementations can benefit from many of the same optimizations that facilitate specialized NAP protocol performance. Section 4 discusses these issues in more detail.

- **Develop an alternative IP-based transport protocol.** TCP and UDP are the most widely used transport protocols in the IP suite, however, alternate transport protocols can also be supported within this framework (section 2.1). Sections 3.2 and 3.3 discuss transport protocol issues in more detail.

We believe these methods will allow NAP developers to adapt the IP protocol suite to their high-performance requirements, and gain the benefits of IP's inter-LAN connectivity. Further, the development effort to adopt the IP suite can build on IP's years of development experience to solve NAP-specific issues, and this effort should be less complex than the effort to develop media-specific protocols for NAPs.

3 TCP/IP Protocol Stack Issues

This section examines the functionality of the TCP/IP protocol to address the issue of complexity, and determine why the performance may be suitable (or unsuitable) for NAPs.

Our general premise is "you get what you pay for"; functionality required by NAPs will have approximately the same overhead, regardless of whether it is implemented within an IP framework or in a specialized NAP protocol stack. Further, the overhead for portions of IP not required by NAPs can be minimized by fast-path, common-case implementations. The transport (TCP) and network (IP) processing of common-case incoming packets has already been optimized to approximately 200 machine instructions. This number was observed in [6], and is confirmed by analysis of our TCP/IP implementation for the Netstation display. This would allow a baseline 250,000 packets/sec on a relatively humble 50MIPS embedded NAP processor. This simple analysis indicates that the TCP/IP processing is not a performance bottleneck that would make it unsuitable for NAPs.

3.1 Network Layer Processing

Examination of the IP network-layer implementation shows that the IP end-host processing requires a relatively small amount of code. There are four primary functions that would affect an IP NAP: packet header processing, header checksum calculation, byte-swapping, and fragmentation reassembly. None of these functions represent a prohibitive processing cost.

Processing of the 20-byte IP header is straightforward, and should not represent a significant incremental overhead beyond processing for a specialized NAP protocol. The required processing on outgoing packets generally involves an efficient bulk copy of a 5-word template, and then 4 load and store operations to modify the packet identifier, length, destination address, and checksum fields. To verify proper reception, a simple implementation would have a compare and branch operation for each of the 12 header fields, although optimization can reduce this even further.

The short IP header checksum calculation involves 12 16-bit add operations, 2 shifts and 3 mask operations; the nature of the checksum would allow optimizations using larger (32-bit) add operations. Although the header checksum itself is not a prohibitive cost, data checksumming incurs considerable overhead; we address data checksumming in Section 3.2.

Byte-swapping is necessary only for little-endian machines, where byte order does not match network standard byte order [7]. However, it is an operation that may affect all network communications, thus this discussion applies to transport layer processing as well. Moreover, this implies that even in a non-IP NAP, this function is required to support heterogeneous hosts.

The most complex, and potentially expensive, functionality we must deal with is IP fragmentation reassembly. Fortunately, we can virtually eliminate this processing by sending appropriately sized transport layer segments. Sending such segments should be straightforward for common-case NAP applications on a single local area network (LAN) where the maximum frame size is known. Mechanisms such as MTU discovery [19] can provide the analogous information across a WAN. For both cases, higher layer communication protocols can avoid packet fragmentation by negotiating the segment size; this is specifically supported by TCP's Maximum Segment Size option.

3.2 Transport Layer

Transport layer functionality is more complex than that of the network layer. In the case of TCP, it provides for in-order reliable delivery, with a number of window and congestion control mechanisms. However, we also do not believe transport layer functionality will prove prohibitive, rather this is the area where NAP developers would focus their efforts.

The general reasons that we believe transport functionality in an IP stack is not prohibitive are as follows:

- **UDP is available as a lower-overhead alternative.** Assuming that NAPs will be commonly accessed within a local environment, the general-purpose functionality provided by TCP may not be needed. UDP provides a very simple protocol that provides for connection demultiplexing and checksumming. Both our Netstation VISA protocol (Section 5.2) and the common NFS protocol [22] implement their own simple reliability for LAN communications on top of UDP.

- **Fast-path TCP is already reasonably efficient.** As noted earlier in the paper, the common-case TCP overhead allows for many thousands of packets per second, provided system issues such as data copies, context switching, and checksumming are addressed. These issues must be dealt with in either the case of an IP or a non-IP NAP.

- **IP can support other transport protocols.** This paper argues primarily for an IP NAP, and that there would be considerable advantage if NAP development could exploit either of the existing transport protocols (UDP and TCP) by providing implementations optimized for the NAP environment. However, IP already supports multiple transport protocols, and recasting current NAP-specific transport protocols into the IP framework seems a viable alternative. New general-purpose transport protocols face a practical hurdle on account of the large installed base of TCP and UDP, and need for widespread deployment and backward compatibility. However, a NAP transport protocol could be deployed incrementally as NAPs are integrated into the computing environment, which would mitigate this issue of ubiquitous deployment.

These general observations allow for the feasibility of an IP NAP with appropriate transport functionality. Due to the number of transport protocols and different levels of service, we cannot address the spectrum of transport protocol functionality. However, the most important transport functionality that a NAP would have to address will be common to any protocol, specifically data checksumming and retransmission and reliability. The issues of connection demultiplexing and segmentation and reassembly are addressed in the following section.

Computing a checksum over the entire data packet is one of the biggest obstacles to low-overhead transport performance. In the case of a NAP (or any network node), either reliability guarantees are required from the transport layer or they are not. If this functionality is to be provided, the checksum overhead is incurred regardless of the particular protocol framework. If a checksum is not required or is not used by a NAP-specific protocol, then an IP-based stack can also eliminate this overhead. Currently, the UDP specification allows for non-checksummed payloads, and in the future, NAP developers could incorporate this functionality into their preferred transport layer protocol. Efficient checksum implementation is discussed in Section 4.

Our reasoning about reassembly and reliability mechanisms at the transport layer is much the same as for checksumming; either this functionality is needed, or the media layer provides sufficient guarantees to allow a simpler mechanism. In the former case, TCP has years of development lead time, in the latter case UDP provides a simple low-cost framework, and there is the option of developing another transport protocol to run over IP.

3.3 Open Issues

We classify the efforts required to achieve IP NAPs as either addressing implementation and system issues (discussed in Section 4), or as protocol development efforts. The latter includes (1) the selection, design, and development of a transport protocol and transport protocol options, and (2) providing support for application layer functionality.

The transport layer issue is of practical as well as technical importance. The perceived lack of a suitable standard transport protocol is perhaps the most significant barrier to the development of IP NAPs. We have also run into this issue within the Netstation project, and are not 100% decided after having experimented with UDP, TCP, and our own specialized transport protocol designed to run over IP. The main action item is a study to decide whether (1) an enhanced implementation and operating system for TCP will provide sufficient performance, (2) a standardized simple reliability mechanism can be effectively provided on UDP, or (3) whether one of the current media-specific protocols should be brought into the IP framework.

Support for high-performance application layer functions is also an open issue. In particular, application layer framing and fast application demultiplexing may be critical functionality. Currently, support for these issues is minimal.

Application layer framing is concerned with providing the efficient communication of application-sized data chunks (e.g. a disk block or track) using packet sizes provided by the lower layers. A simple case of this would be using a path MTU (maximum transmission unit) discovery mechanism to find that the physical network supports 20Kbyte packets, using TCP's Maximum Segment Size option to coordinate the transport layers, and providing this information to the application so it can send an integral number of its data blocks. However, this issue becomes more complex as application data units become larger than a single network packet. Application layer framing not only deals with protocol issues such as packet sizes and transport-layer fragmentation, but also must be integrated with other sources of overhead in the system such as interrupts and context switches (per network packet or per application data unit) and data copies often required for packet reassembly.

Fast application-level demultiplexing and delivery is another issue, and one of the areas media-specific protocol stacks are addressing (e.g., Fibre Channel allows data blocks to be passed directly into pre-defined memory locations at a receiver, achieved by processing all network protocols in the host adapter). The first component of this issue is simply an efficient mechanism to associate an incoming packet with the expecting receiver. If there are thousands of open connections, a poor implementation of TCP, for example, can require considerable overhead to find the appropriate match on the tuple:

<source address, destination address, source port, destination port>

Implementations can use hashing or caching of recently active session to improve performance, but another option to examine is providing a short header "handle" that applications may pass back and forth. Providing for application-specific delivery or disposition of data is also an important area of research [9, 16, 2, 27].

4 Implementation and System Issues

Many of TCP/IP's known performance problems come from outdated implementations or restrictions imposed from outside. In this section, we address some of those concerns in the context of implementing TCP/IP inside a network attached peripheral.

Changing to larger data units can dramatically reduce the CPU load at high data rates. On media that support large packet sizes, the first step in reducing CPU utilization and increasing throughput is to increase the packet size, which IP supports. For disk drives, data payloads which conveniently map to file system pages, such as 4KB, may be particularly efficient.

Much of the CPU cost of networking in general-purpose hosts comes from context switches and virtual memory management. With a lightweight, real memory embedded operating system in the NAPs, context switches are inexpensive. Without virtual memory, maintenance of page tables, with their associated locking, mapping and protection, is unnecessary.

Reduction of data-touching operations is required to achieve high performance with minimal CPU cost. Data touching takes two forms, data copies and checksum calculation.

Data copies are reduced through integrated layer processing and possibly the use of zero-copy APIs, without the overhead of the Berkeley sockets layer. Data may be shared directly between the "application" (the SCSI command processor/disk drive track buffer manager, in the case of a disk drive) and the networking stack [8]. Fast demultiplexing of received data can also be used.

The overhead of the checksum can be eliminated in several ways. The simplest but least desirable option is to eliminate it and depend on the LAN checksum to protect the data; this is common today for UDP, but does not protect data end-to-end or across network boundaries. Alternatively, the CPU can calculate the checksum in conjunction with a data copy [20]. This can generally be done for zero cost on RISC processors by putting the adds in the delay slots of loads and stores. The checksum can also be calculated on any data movement, such as the reception from the network or from the disk platter into buffer memory, with simple hardware modifications [11, 23].

It is possible to store the TCP checksum on disk with the data by reformatting the disk with a larger sector size. Because the checksum is additive, storing it with each sector allows quick calculation regardless of the packet boundaries used for transmission.

The remaining per-packet CPU costs can be controlled by careful use of a "fast path" through the system, in which the common case is optimized at the expense of unusual cases. For example, TCP header prediction assumes that most packets are either an in-order data packet or the next expected ACK, and simple optimized tests for these cases appear very early in the code. This improves performance for the majority of packets, but is additional (and redundant) processing for packets arriving out of order.

5 Netstation Implementation Experience

The Netstation Project has implemented prototype IP NAPs as part of its research into a LAN-based system architecture. We have built a display NAP based on a custom hardware design, and are currently completing a camera NAP based on the same hardware components as the display. Other NAPs, including the IP disk and keyboard, are emulated using SUN workstations. Netstation uses Sparc 20/71 workstations running SunOS 4.1.3 as the "CPU nodes" that control and access the IP NAPs.

The remainder of this section provides a summary of the Netstation Project, and describes our IP NAP implementations with a focus on the IP stack performance and overhead.

5.1 Netstation Overview

Netstation [10, 12] is a heterogeneous distributed system composed of processor nodes and peripherals (NAPs) attached to a high-speed LAN which provides high aggregate bandwidth; currently we use both a 640 Mbps Myrinet network and a 100 Mbps ethernet.

Netstation is predicated on the observation that shared I/O buses provide poor scalability, and are falling behind the technology curve of high-speed LANs. By connecting peripherals directly to a switched high-speed LAN, a Netstation system can avoid the shared-bus bottleneck. This allows Netstation components to communicate directly with each other, without intervention of the main system processor or server processor, e.g. a user can initiate a disk backup to tape, and have negligible impact on a resource-intensive multimedia conference. This sharing of resources also creates flexibility in system configuration.

The ultimate Netstation goal is a ubiquitous network-based architecture, but the results can also be applied individually to produce NAPs. The Netstation project concentrates on operating systems mechanisms and network protocols for NAPs. Netstation provides a single system image, allowing the CPU to access the NAPs in a manner as similar as possible to directly attached devices. Because the devices are attached to an open network with both trusted and untrusted nodes on the net, safe shared access and security at the NAPs are critical.

Netstation has developed an abstraction we refer to as the *derived virtual device*, or DVD [25], that provides the mechanisms for safe shared device access. DVDs provide a protected execution context at the device, allowing direct use of the devices by untrusted clients, such as user applications. The owner of a device, or its managing process, specifies the DVD resources, DVD interface, and the security policy, then initializes this configuration at the NAP; in turn, the NAP's DVD provides the specified interface to these resources and enforces the access policy. We believe that DVDs can accommodate all of the issues regarding shared access to NAPs, including third-party I/O transfers. Thus, a camera can be granted write access to a restricted region of a frame buffer, or a user application can be given read-only access to a DVD which represents a disk-based file or disk partition.

Netstation has a number of IP NAP prototypes in various stages of development. A custom motherboard provides the basis for our Netstation display NAP, which provides a network-attached frame buffer interface. The display is controlled by an adapted version of the MIT X11R5 server, which drives its frame buffer remotely across the network. A camera NAP based on the same motherboard is under development. Support for third-party I/O, in which video will be transferred directly from the camera NAP to the display NAP, is a near-term goal. Netstation has also implemented an *IPdisk*, an emulated network-attached disk drive. Access to the IPdisk is made via the Virtual Internet SCSI Adapter (VISA), which provides a SCSI-to-DVD translation to support access to storage peripherals via the network. A simple keyboard NAP was also prototyped earlier in the project.

Described below is our experience with two different IP stack implementations; a summary of the standard SunOS IP stack performance for the IPdisk, and a more thorough discussion of our own TCP/IP implementation for the custom Netstation display system.

5.2 IPdisk Performance and Issues

IPdisk is our emulated network-attached SCSI disk drive. It currently runs as a user process on a Sun workstation. It provides all the mandatory commands for a SCSI direct access device, including TEST UNIT READY, RESERVE, RELEASE, READ, WRITE, FORMAT, and others. Commands and data can be sent via either TCP connections or UDP datagrams with a simple reliability mechanism. Support for third party copy between IPdisks in cooperation with the file system is currently under development. This will move data directly from disk to disk via the network, without consuming memory or bus bandwidth at the controlling host.

VISA, our Virtual Internet SCSI Adapter, is an operating system module added to SunOS. It implements a `scsi_transport` layer in the layered device driver system, accepting commands from the higher-level SCSI disk and tape drivers and packaging and transmitting them to the appropriate devices. The SCSI disk driver and all of the file system components are completely standard; a normal fast file system is built on top of the IPdisk. The VISA module is roughly 2,000 lines of C code added to the SunOS kernel, less than the amount for the standard SCSI host bus adapter.

Preliminary performance measurements indicate that VISA is capable of running at a write rate of 72 Mbps (megabits per second) on a 75MHz SPARCstation 20/71 through the file system to IPdisk via UDP over Myrinet. The same CPU is predicted to reach 110 Mbps through the file system over an infinitely fast SCSI bus to an infinitely fast disk, limited primarily by the OS's ability to manage the virtual memory system. Known optimizations and reductions in the number of data copies could be expected to raise the data rate to 90-95 Mbps, or above 80% of that achievable with a SCSI bus, without the addition of special network coprocessors. We feel this performance is acceptable, and that this supports our premise that the advantages of IP outweigh the performance concerns.

A more detailed description of VISA and its performance can be found in [24].

5.3 Netstation Display

The Netstation display hardware is a custom motherboard centered around TI's 40MHz TMS-320c80 MVP chip, selected primarily for its high performance video and data transfer controllers. A PCI bridge chip (V3 Semiconductor's V960PBC) provides a path from the main C80 bus to an off-the-shelf Intel EtherExpress PR0/100 interface card and shared packet buffer memory. The display hardware also includes an audio subsystem and provisions for two JPEG decompression cards that are under development.

The display operating system is based on the software in the TI development kit, specifically the C8x Multitasking Executive, release 2.0, and the accompanying runtime libraries. The C8x executive provides basic synchronization, communication, and task primitives to support cooperative multitasking in a single shared memory space. This executive is fairly lightweight and efficient; the profiled baseline kernel performance takes 115 clock cycles (2.875 microsecond at 40 MHz) to preempt, switch context, and transfer a message between two tasks.

The TCP/IP stack, including device drivers for the PCI bridge and Intel Ethernet board, is built on top of the OS primitives. Initially, we facilitated development by instantiating a simple task to handle each layer and each side (send and receive) of the communication stack. Over time, we have revised the structure of the system to include two primary tasks: (1) a task to handle the low-level driver, and to process the receive-side of ethernet and IP, and (2) a TCP task to provide the application's network API, which also processes IP and ethernet send-side functionality before passing packets to the low-level task for transmission.

Data copies in the system are kept to a minimum by exploiting the shared-memory model of the OS. Currently, a sending application copies data into buffers to be sent, and after being passed between tasks by reference, the data is copied out of the system to the ethernet board for transmission; the receive side incurs two analogous copies. We believe the current structure would allow us to eliminate one copy in each direction with minimal effort, using mechanisms such as scatter-gather vectors and requiring application cooperation to manage buffer memory.

5.3.1 TCP/IP for the Netstation Display

The network and transport layer functionality currently implemented for the Netstation display includes: IP, TCP, Address Resolution Protocol (ARP), and the echo function of the Internet Control Message Protocol (ICMP). We have not yet implemented UDP, however, it is straightforward and would be much simpler than TCP to implement. The ICMP function was included for testing and debugging of the low-level drivers. We believe these five components represent the minimal stack needed to communicate in a TCP/IP environment, and that this is a reasonable amount of functionality to include in a NAP.

The IP, ARP, and ICMP protocols were implemented from scratch. Our TCP implementation is based on INRIA's user-level TCP [5], which in turn is based on BSD TCP. Note that our TCP implementation does not run as part of the user task, rather we take advantage of the low-overhead kernel primitives and manage TCP as a separate task. We also manage data buffers and the user API differently than either the INRIA or Berkeley implementation.

The amount of memory required to implement this functionality at a NAP is also reasonable, and should not be a burden on potentially limited NAP resources. Further, assuming the amount of code is indicative of the effort required for the development of an IP NAP, this too seems reasonable. Table 1 enumerates the amount of C code and the memory footprint of the Netstation networking and transport layer stack, with the OS primitive information provided as a point of comparison.

Component	C-Code (lines *.c)	Memory Footprint (bytes of static code)
ICMP	90	184
ARP	570	2804
IP	1210	6444
TCP	2700	12636
shared code	390	1248
user lib	630	3908
TCP/IP Total	5590	27224
Reference pts:		
OS Primitives	3400	7708
OS Libraries	2300	3860

Table 1: TCP/IP Code Size

Other functionality (e.g. DNS, RARP, SNMP, etc) would be useful to facilitate monitoring, configuration, and other administrative functions, and the existence of these networking standards might further simplify the development of commodity NAPs. However, these additional components are not necessary for NAP communication, and should not be considered part of the "TCP/IP baggage" that might prevent the adoption of IP for NAPs. An actual IP host will have additional functionality, per the Host Requirements RFCs [4, 3], however, we believe IP NAPs would have a similar, but reduced, set of requirements.

The performance of the display TCP/IP implementation is currently being evaluated, concurrently with final hardware and software debugging and tuning.

6 Summary

This paper suggests that the TCP/UDP/IP framework offers considerable advantages for network-attached peripherals, in particular cross-media bridging, ubiquity, and the existence of considerable prior work. We have argued that using IP for network attached peripherals is preferable to the development of specific networking protocols optimized for each new physical media.

We have addressed the primary concern regarding IP for NAPs, pointing out that the performance of IP is strongly dependent on the operating system environment and the particular implementation. We suggest that IP performance would be sufficient, if the efforts to develop NAP specific protocols were redirected toward optimizing IP for the NAP environment.

Finally, we have described the Netstation project's use of the TCP/IP network protocol suite as the means to access several different types of peripherals, including disk drives and displays. We have shown how access to our IPdisk using VISA can perform at over 80% of a native SCSI disk performance using simple known optimizations to SunOS UDP. There is reason to believe that additional performance improvements would be achieved in a typical light-weight embedded NAP implementation.

Our conclusion is that IP is the appropriate choice for interconnecting subsystems and that the desired performance is feasible, although additional work is required before the performance reaches sufficient levels.

Acknowledgements

This research was sponsored by the Defense Advanced Research Projects Agency under Contract No. DABT63-93-C-0062. Views and conclusions contained in this report are the authors' and should not be interpreted as representing the official opinion or policies, either expressed or implied, of DARPA, the U.S. Government, or any person or agency connected with them.

References

[1] J. F. Adam, H. H. Houh, M. Ismert, and D. L. Tennenhouse. Media-intensive data communications in a "desk-area" network. *IEEE Communications*, pages 60–67, Aug. 1994.

[2] M. L. Bailey, M. A. Pagels, and L. L. Peterson. The x-chip: An experiment in hardware demultiplexing. In *Proceedings of the IEEE Workshop on High Performance Communications Subsystems*, Feb. 1991.

[3] R. Braden. Requirements for internet hosts - application and support. Internet Draft RFC 1123, USC/ISI, Oct. 1989.

[4] R. Braden. Requirements for internet hosts - communication layers. Internet Draft RFC 1122, USC/ISI, Oct. 1989.

[5] T. Braun, C. Diot, A. Höglander, and V. Roca. An experimental user level implementation of TCP. Technical Report RR-2650, INRIA, Sept. 1995.

[6] D. D. Clark, V. Jacobson, J. Romkey, and H. Salwen. An analysis of TCP processing overhead. *IEEE Communications*, 27(6):23–29, June 1989.

[7] D. Cohen. On holy wars and a plea for peace. *IEEE Computer*, Oct. 1981.

[8] P. Druschel and L. L. Peterson. Fbufs: A high-bandwidth cross-domain transfer facility. In *Proc. Fourteenth ACM Symposium on Operating Systems Principles.* ACM, Dec. 1993.

[9] D. R. Engler and M. F. Kaashoek. DPF: Fast, flexible message demultiplexing using dynamic code generation. In *Proc. SIGCOMM '96*, volume 26, pages 53–59. ACM, Oct. 1996.

[10] G. Finn. An integration of network communication with workstation architecture. *ACM Computer Communication Review*, Oct. 1991. Available online at http://www.isi.edu/netstation/.

[11] G. Finn, S. Hotz, and R. Van Meter. The impact of a zero-scan internet checksumming mechanism. *ACM Computer Communication Review*, 26(5):27–39, Oct. 1996.

[12] G. G. Finn and P. Mockapetris. Netstation architecture: Multi-gigabit workstation network fabric. In *Proc. NetWorld+InterOp Engineer Conference*, 1994.

[13] G. Gibson et al. A case for network-attached secure disks. Technical Report CMU-CS-96-142, CMU, June 1996.

[14] M. Hayter and D. McAuley. The desk area network. *ACM Operating Systems Review*, 25(4):14–21, Oct. 1991.

[15] R. W. Horst and D. Garcia. ServerNet SAN I/O architecture. In R. Rettberg and W. Dally, editors, *Hot Interconnects Symposium V.* IEEE Computer Society, 1997.

[16] J. S. Kay. *Path IDs: A Mechanism for Reducing Network Software Latency.* PhD thesis, UCSD, 1995.

[17] B. Kobler, editor. *Fifth NASA Goddard Conference on Mass Storage Systems and Technologies*, Sept. 1996.

[18] E. K. Lee and C. A. Thekkath. Petal: Distributed virtual disks. In *Proc. ACM Seventh International Conference on Architectural Support for Programming Languages and Operating Systems.* ACM, Oct. 1996.

[19] J. Mogul and S. Deering. Path MTU discovery. RFC 1191, Internet Request For Comments, 1990.

[20] C. Partridge and S. Pink. A faster UDP. *IEEE/ACM Trans. on Networking*, 1(4):429–440, Aug. 1993.

[21] S. R. Soltis, T. M. Ruwart, and M. T. O'Keefe. The global file system. In Kobler [17], pages 319–342.

[22] Sun Microsystems Inc. NFS: Network file system protocol specification. Technical Report RFC 1094, Internet Request For Comments, 1989.

[23] J. Touch and B. Parham. Implementing the internet checksum in hardware. Technical Report Internet RFC 1936, ISI, Apr. 1996.

[24] R. Van Meter, G. Finn, and S. Hotz. VISA: Netstation's virtual internet SCSI adapter. in preparation.

[25] R. Van Meter, S. Hotz, and G. Finn. Derived virtual devices: A secure distributed file system mechanism. In Kobler [17].

[26] D. Wiltzius and K. Minuzzo. Network-attached peripherals (NAP) for HPSS/SIOF. web page, Oct. 1995. http://www.llnl.gov/liv_comp/siof/siof_nap.html.

[27] M. Yuhara, B. N. Bershad, C. Maeda, and J. E. B. Moss. Efficient packet demultiplexing for multiple endpoints and large messages. In *Proc. USENIX Winter 1994 Technical Conference*, Jan. 1994.

Experiences With OpenVault, A Prototype Implementation of a Standards Effort

Curtis Anderson
Silicon Graphics, Inc.
2011 N. Shoreline Blvd, M/S 802
Mountain View, CA 94043
curtis@sgi.com
Voice: +1 650 933-1193
FAX: +1 650 933-3542

1.0 Introduction

The Storage Systems Standards Working Group (http://www.ssswg.org) is developing new removable media management standards—defining a set of testable standards that can be used by end users and industry to build multi-vendor storage environments. These standards will allow end users to combine the most appropriate pieces from each vendor to best meet their needs.

Silicon Graphics has provided engineering resources to implement a prototype of the proposed design to validate the architectural approach. That prototype is called OpenVault. This paper describes that effort and what we have learned so far.

2.0 Architectural Requirements

The architecture for OpenVault was chosen to meet requirements for library devices and for drives. A drive can be viewed as a device that applications know about directly and interact with. A library device, on the other hand, can be completely hidden from the application by a storage management system. The storage management system's only job is to move named cartridges into compatible drives so that applications can use them. In the future, a Mover will probably be added to this architecture to hide drive-specific issues from the application as well. The major requirements for a media management system are discussed in the following sections.

2.1 Separation of Device-Dependent and Device-Independent Software

Traditional removable storage management systems build library-device control directly application software. This linkage severely limits the user's ability to choose library hardware that is not supported by the applications. It also imposes a large burden on the application vendor in doing regression testing of each supported library with each new release of the application (or the underlying OS).

Traditional systems also build drive dependencies into the application. The current IEEE project is not tackling that issue yet, as we believe that it requires a Mover to isolate the application from drive idiosyncrasies.

2.2 Sharing of Hardware Resources

To effectively manage a library device, it is easiest to dedicate that device to one storage management application, the approach taken in almost all current management applications.

For low-usage or occasional-use storage applications, dedicating a library device to the application is generally expensive. Obtaining larger library devices, shared between applications, is both more flexible and more cost effective.

Some of today's libraries can be "partitioned" in a fixed manner between several storage management applications. Unfortunately, this usually requires that the drives be partitioned as well. Dedicating drives to applications is not only expensive, it is inefficient in terms of available system bandwidth.

Thus, the ability to use all of the available (compatible) drives for currently high priority tasks would be of great benefit to the end-user.

2.3 Cross-Platform Portability

Application vendors desire standard interfaces to storage management services on all the platforms that they support. This allows them to lower the cost of supporting each platform. To meet this desire, a storage management system should be portable to multiple platforms while providing the same interface on all platforms.

2.4 Centralized Storage System Management

System administrators desire one central place to manage their storage environment; if they have several large servers with tape drives, they would like to manage all the cartridge pools on all of their servers at the same time according to consistent policies.

Sites must be able to strike their own balance between local autonomy and central control, which implies that both local autonomy and central control must be possible.

2.5 Run Time Plug-And-Play

The separation of device-dependent code from device-independent code implies a standard interface between the two portions. Since standard interfaces allow components to be built by different parties, the system must be able to support run-time plug-and-play of components with components coming from multiple sources.

In a large and diverse market, a central registration authority for components is not practical, so a non-centrally-controlled system is required.

2.6 Network Security

To operate in distributed systems, systems must be able to protect themselves from attacks via the network. To allow export from the United States, pure encryption of the channel is not an option.

3.0 Architectural Overview

At its most abstract, the media management environment is composed of five major components, each of which is most likely a separately running process or set of processes:

- Client applications,
- Library Managers (LMs),
- Drive Managers (DMs),
- The Media Manager (MM), and
- A Catalog (part of the MM).

We will describe each of these pieces and then present a rationale for each design. We will define these five pieces indirectly by defining the communications protocol between them.

3.1 ASCII Languages and TCP/IP Sockets

The interprocess communication mechanism from clients, LMs, and DMs to the MM is plain ASCII text passed through TCP/IP sockets. Well-defined languages are parsed to recover the meaning of the messages being passed. The syntax and semantics of each language is designed for the pair of components that it is connecting.

The currently defined media-management languages are asynchronous: Many commands can be outstanding at any time and responses are not required to come back in the order that the commands were submitted. The languages are also fully bi-directional in that each end of the socket has the same level of asynchrony available to it, allowing each end to have commands outstanding with the other end at the same time.

3.2 The HELLO Protocol and Version Negotiation

All the network connections in this architecture form a star topology with the MM at the center of the star. Each of those connections is initiated by the piece outside the MM—the MM never initiates connections.

When an application, LM, or DM first contacts the MM they participate in a protocol that announces to the MM details about the connecting process. Those details include which language and which versions of that language the process is able to speak. From the language name the MM can determine what type of process is connecting, e.g., client, LM, or DM.

If there are no problems, the MM responds with a positive acknowledgment and the language version it has chosen to use during this session. Otherwise, the MM will respond with an error message and will close the connection. Examples of problems include an unknown language name or a non-supported language version.

3.3 Security and Message Integrity Codes

The initial contact from the client includes the name of the client and a name for this instance of that client. The MM uses that information to look up a shared, secret password for that client and instance.

If a non-null key exists, then a per-session unique encryption key is generated. The key is used to generate a digital signature for each command passed through the TCP/IP socket. The digital signature follows the clear text of the command and allows the receiver to determine if the message has been modified in transit.

If no key can be found then only the clear text of the command passes through the socket.

3.4 The MM and the Catalog

The Media Manager is the central coordinator for the environment. It makes use of the services offered by the attached LMs and DMs to accomplish the tasks requested by applications.

An internal component of the MM, called the Catalog, persistently stores the characteristics of the objects being managed and processes arbitrarily complex user-specified queries against that set of data. The results of those queries are used to identify the objects that the application wants to operate on. The catalog objects being operated on include cartridges, drives, libraries, applications, cartridge groups, and drive groups.

3.5 Applications and MMP

Client applications may be full hierarchical storage management systems or simple Perl scripts accessing removable media cartridges. They communicate with the MM via a language called the Media Management Protocol (MMP). MMP has multiple privilege levels to support both privileged and unprivileged clients.

The major capabilities that an unprivileged client can make use of in MMP are:

- Obtaining ownership of a cartridge from a scratch pool and releasing it again,
- Mounting and unmounting cartridges that the application owns,
- Storing and retrieving application defined metadata on owned cartridges, and
- Viewing information about the current composition and status of the system.

A privileged client application adds at least the following abilities:

- modifying the configuration and composition of the system,
- managing queued tasks, and
- providing operator interaction support for LMs and DMs if required.

MMP is composed of many separate commands, but there are also clauses in the language that are common to most of those commands. The most important of these is the "match" clause which determines the objects on which the command should operate.

The match clause is an arbitrarily complex algebraic expression that defines the criteria controlling inclusion into the set of catalog objects operated on by the command. It is a declaration of constraints that all solutions must satisfy. In theory, the expression is evaluated once for each possible combination of objects in the catalog. For each such evaluation that returns "true", that combination of objects will be included in the set of objects that will be operated on by the command. In practice, techniques more efficient than brute force can be used.

For example, if a client application wants to mount a cartridge such that it will have greater than 9 MB/sec of nominal read bandwidth from the drive, the application can express that constraint in almost exactly those terms. Given the known cartridges, the currently connected Library and Drive Managers, and the characteristics of the devices they control, the MM must find a solution that satisfies the application's constraint.

3.6 The LM and LMP

A Library Manager is a specialized interface process that implements high-level commands by understanding the particular library device's operational characteristics and the available device-specific commands. LMs communicate with the MM through the Library Management Protocol (LMP) which defines an ASCII interface to an abstract Library.

The LMP is not just a command set, it defines a relationship between an LM and the MM where they each have responsibilities to the other. The LM's primary responsibilities are to implement the commands that come down from the MM and to keep the MM updated on any changes in the state or configuration of the library device. The LM is the authoritative source for all information about the library and is required to tell the MM whenever anything changes that would invalidate the MM's cached copy of that information.

The LM describes the library to the MM in a stylized structure incorporating three basic elements: slots, bays, and drives. A slot and a drive are just what you would expect. A bay is best described by visualizing a group of Storage Technology silos connected by pass-through ports. Each silo in that picture is a "bay".. Thus, a bay is a collection of slots and drives that form a locality group in terms of access time. When the LM reports the slots to the MM, it includes the external label (a bar code, for example) along with the type of cartridge (DLT, 3480, etc.).

3.7 The DM and DMP

A Drive Manager is a specialized interface process that implements high-level commands by understanding the particular drive's operational characteristics and the available device-specific commands. DMs are similar to LMs except that they control drives instead of libraries. Their language is the Drive Manager Protocol (DMP) which defines an ASCII interface to an abstract drive; the DMP defines a relationship between a DM and the MM where each one has responsibilities to the other.

The DMs primary responsibilities are to implement the commands from the MM and to keep the MM updated on changes in the state or configuration of the drive. The DM is the authoritative source for all information about the drive and tells the MM whenever changes invalidate the MM's cached copy of that information.

Drives usually support multiple modes (e.g., compression or not), and varying levels of backward compatibility. For example, a DLT7000 drive can read but not write a DLT2000 cartridge. The DM describes the drive to the MM in a stylized structure that lists the supported combinations of various characteristics. Those characteristics include the type of cartridge, the format of the bits on the media, and a set of "capability strings".

Examples of cartridge types are "DLT" and "3480", while examples of bit recording formats are "3480" and "3490". A "capability string" is an uninterpreted token that the DM provides to the MM. Applications also provide such tokens to the MM during a mount operation. This allows the MM to rendezvous the application's desires with the abilities of the drive. Examples of capability strings include "rewind" and "compression".

The description that the DM sends to the MM contains an unordered set of named combinations. Each combination is defined by the values of "cartridge type", "bit format", and a particular set of "capability string" tokens. The MM treats each combination as a separate "mode" that the drive supports.

When evaluating a "match" clause for an MMP command, if the MM finds a match between the known characteristics of a candidate cartridge (cartridge type and bit format) and a match between the "capability strings" required by the application with those offered by the drive, then the MM will conclude that that drive can be used to access that cartridge in the way that the application desires.

4.0 Architectural Rationale

Telling customers that they must buy an additional tape robot when they add a new robot-aware application to their system is not popular, yet that is where the SCSI-attached robot market is today. If the value of that application is not enough to justify an entire robot or a high-performance drive, then the site has to make do with a smaller robot or with a less capable drive. To grow their market, the storage industry must break that linkage and share robots and drives between multiple applications.

The requirement to separate device-dependent from device-independent software led to the concept of Library Managers and Drive Managers as separate entities. This allows device-specific code to deal with the idiosyncrasies of each device while isolating that code from the core mechanisms of the system. Device-specific code also allows implementors to maximize the performance and reliability of each device.

Defining the relationship between a DM or an LM and the MM was a delicate balancing act which we will describe in this section. In designing the LMP and the DMP, we chose to migrate complexity from the LM and the DM to the MM to reduce the implementation effort for all of the LM and DM writers.

4.1 ASCII Languages and TCP/IP Sockets

The requirement for portable, run-time plug-and-play support led us to choose simple ASCII languages through TCP/IP sockets as the interprocess communication mechanism. This is the most portable solution since all major operating systems support TCP/IP and ASCII strings. Binary RPC mechanisms are less portable and make asynchronous commands more difficult to implement.

Defining the linkages between components rather than defining the components themselves allows implementors to create new components and to plug them into running systems.

4.2 The HELLO Protocol and Version Negotiation

The "hello" protocol supports system evolution; new versions of existing languages and entirely new languages can be defined, with the MM negotiating which language and which version to use at connection setup time. This allows both gradual upgrades of components and mixed-version environments.

The syntax of the protocol flowing through the socket after the successful completion of the hello protocol is dependent on the language and language version that were negotiated during the hello protocol. Thus, a binary language, rather than an ASCII one, can be defined if desired.

4.3 Security and Message Integrity Codes

Message integrity codes (MICs) were chosen over pure encryption of the command stream so that products can be exported from the United States.

Since the text of the command is in the clear just preceding the integrity code, an eavesdropper can see the command traffic flowing back and forth but cannot change it. Since no end-user data flows through the command channels, the MIC technique is a good compromise.

4.4 The MM and the Catalog

It is likely that many LMs and DMs will be written to handle the variety of current and future devices. On the other hand, we expect only a few MMs to be written from scratch. It is possible that a single, standard MM will be developed, ported, and enhanced over time. Thus, we decided to move complexity from the LMs and DMs (and storage-accessing applications) into the MM. Part of that complexity was providing a persistent storage mechanism that the LMs, DMs, and applications could use to manage system information, simplifying the task of developing LMs and DMs.

Another area of complexity was in the Catalog's support of the MMP's "match" clause. Defining the match semantics and designing efficient algorithms was difficult. There are many implicit relationships between the objects stored by the Catalog, and yet the results from a mount command and commands that report the contents of the Catalog must be consistent. For example, a "volume" is an application-accessible object that exists on a "cartridge". If a client sends a "match" clause saying that the volume color must be blue and the cartridge color must be red, the result should include only blue volumes that exist on red cartridges, not all red volumes plus all blue cartridges. The rationale is that the only cartridge that is relevant to a volume is the cartridge containing the volume. Although complex, the "match" clause provides a general mechanism that is able to handle current and future requests through one interface.

4.5 The Application and MMP

To gain initial acceptance of this architecture, existing storage management applications must begin to adopt it. This implies that we need to reduce the amount of application rework needed to begin using the MM architecture. The design of the MMP allows an application to start with basic functionality, like mounting a named cartridge into a named drive, and grow into more advanced operations such as using the "match" clause over application-specific metadata stored on objects in the MM system.

4.6 The LM and LMP

The protocol between the LM and the MM must provide a common look and feel for all robots and yet be able to take advantage of the unique capabilities of each robot design. We chose to make the LM an active partner of the MM, rather than a slave, to give the LM the ability to react to robot changes asynchronously and to initiate state changes in the MM as a result.

To be general, the MM must not be dependent on any particular hardware model. The slots and bays that the LM describes to the MM, for example, are defined solely by the LM and are named with strings. The MM only compares those strings for equality with other strings. Thus, the MM can handle any library that is defined in terms of slots and bays, and the LM is free to map those constructs onto reality in any convenient way.

The MM must also present a view of the physical hardware to administrative applications and humans. Thus, the strings that the LM gives to the MM to name slots and bays must be human readable, and the human must be able to map them onto physical features of the library device.

4.7 The DM and DMP

The DM needs to support the abstract drive that the DMP language defines while making use of the unique capabilities of each physical drive. Like the LM, we chose to make the DM an active partner to the MM rather than a slave.

To allow the MM to decide if a drive can be used to access a given cartridge, the DM must present the capabilities of the drive to the MM in a standard way. For example, the DMP must be able to represent all of the possible combinations of cartridge types and bit formats. This will require, at least, a standard set of token definitions.

4.8 The Lack of a Mover (So Far)

The current architecture supports local access to drives via the native operating system access methods (e.g., /dev nodes on UNIX) and leaves a hole for later addition of a Mover component. We believe this is an acceptable limitation at this time but acknowledge that a true Mover is required in the long term. Without a Mover, the MM will not be able to protect internal labels from malicious applications on platforms that do not provide that protection through the operating system. The MM will be able to control access to drives, however, by carefully managing either the permissions on the drive access handle (e.g., the /dev/mt node) or by creating and deleting the drive handle on demand. The particular technique used to control access to the drive is DM implementation specific.

5.0 Future Directions

There are several capabilities that we would like to add to the system. The architecture described here has the power and flexibility required to include these capabilities without a major redesign.

5.1 Addition of a Mover

There are two major advantages to having a Mover plus a few disadvantages. A Mover in the data path can intercept any repositioning commands and protect an internal label from modification by unauthorized clients, even if the operating system provide no protection for the label.

A Mover will also provide access to drives that are not physically connected to the system on which the application is running. This will allow sites to centralize removable media resources and to share devices that would otherwise be too expensive for local applications to justify.

The major challenge with a Mover component is that it must have very low overhead during data transfers. The Mover interface must be able to faithfully reproduce the semantics of the drive it is connected to, or applications must be modified to use the Mover semantics. Neither of these options is trivial.

A Mover in the MMS architecture, like the DM, will be device- and OS-specific much as a DM is device- and OS-specific. This allows the Mover to accommodate drive idiosyncrasies and allows device-specific optimizations, but requires an interface that will be common to all Movers.

5.2 Network Attached Storage

The rapid evolution of network-attached storage devices in the market today makes it difficult to predict future directions. The basic concepts of removable media will remain the same, but the details of managing such environments will change. The goal is to manage network-attached devices through the same interfaces used to manage host-attached devices.

5.3 Multiple-Simultaneous-Access Media Types

The popular removable media devices today are inherently exclusive-access devices. For example, tape drives do not support multiple applications accessing different portions of the tape at the same time. Newer types of removable media will allow simultaneous accesses.

DVDROM and the new magneto-optical disks are examples. Truly simultaneous access to a single volume might be limited to read-only access, but simultaneous read/write access to different partitions on a given cartridge should be possible.

5.4 Time Based Scheduling

With the convergence of computers and television, the concept of time-based scheduling will be needed in storage management systems. With TV stations using robotic storage to manage commercials, a natural MMS request will be to "mount this tape for access at 5:28pm".. The storage management system is the only software in a position to know all the demands being placed on the removable media system, and therefore must control the scheduling of drive access.

6.0 Lessons Learned So Far

The concept of using LMs and DMs and abstract languages to connect them to a central management engine has worked out very well. The idea of providing a declarative, constraint-based, arbitrary query mechanism has proved to be powerful and general. It is not yet clear what the correct balance should be between the most intuitive relationships between the objects represented in the MM and the most convenient relationships.

Page intentionally left blank

An Emerging Network Storage Management Standard

Media Error Monitoring and Reporting Information (MEMRI) - To Determine Optical Tape Data Integrity

Fernando Podio
Information Technology Laboratory
National Institute of Standards and Technology
Bldg. 225, Room B255
Gaithersburg, MD 20899
fernando.podio@nist.gov
Tel: +1 301-975-2947
Fax: +1 301-869-7429

William Vollrath
LOTS Technology
1274 Geneva Drive
Sunnyvale, CA 94089-1122
WilliamVollrath@msn.com
Tel: +1 408-747-1357
Fax: +1 408-747-0245

Joel Williams
Systems Engineering and Security
7474 Greenway Center Drive. Suite 700
Greenbelt, MD 20770
joel.williams@ses-inc.com
Tel: +1 301-441-3694
Fax: +1 301-441-3697

Ben Kobler
National Aeronautics and Space Administration
Goddard Space Flight Center, Code 505
Greenbelt, MD 20771
ben.kobler@gsfc.nasa.gov
Tel: +1 301-614-5231
Fax: +1 301-614-5267

Don Crouse
LSC Inc.
4211 Lexington Ave No.
Arden Hills, MN 55126
Don@lsci.com
Tel: +1 612-482-2348
Fax: +1 612-482-4595

Abstract: Sophisticated network storage management applications are rapidly evolving to satisfy a market demand for highly reliable data storage systems with large data storage capacities and performance requirements. To preserve a high degree of data integrity, these applications must rely on intelligent data storage devices that can provide reliable indicators of data degradation. Error correction activity generally occurs within storage devices without notification to the host. Early indicators of degradation and media error monitoring and reporting (MEMR) techniques implemented in data storage devices allow network storage management applications to notify system administrators of these events and to take appropriate corrective actions before catastrophic errors occur. Although MEMR techniques have been implemented in data storage devices for many years, until 1996 no MEMR standards existed. In 1996 the American National Standards Institute (ANSI) approved the only known (world-wide) industry standard specifying MEMR techniques to verify stored data on optical disks. This industry standard was developed under the auspices of the Association for Information and Image Management (AIIM). A recently formed AIIM Optical Tape Subcommittee initiated the development of another data integrity standard specifying a set of media error monitoring tools and media error monitoring information (MEMRI) to verify stored data on optical tape media. This paper discusses the need for intelligent storage devices that can provide data integrity metadata, the content of the existing data integrity standard for optical disks, and the content of the MEMRI standard being developed by the AIIM Optical Tape Subcommittee.

1. Introduction

Network storage management applications provide for high data accessibility by means of backup and archiving, remote vaulting, file mirroring, hierarchical storage management, and storage device and server mirroring (Peterson [1]). The amount of stored digital data is increasing for certain applications to Terabytes (10^{12} bytes) and even Petabytes (10^{15} bytes). The amount of stored data per media unit is also increasing. It is currently possible to utilize magnetic tapes that store tens to hundreds of Gigabytes per tape cartridge and optical disks with capacities of tens of Gigabytes. It is anticipated that when optical tape products become available they will reach Terabyte capacities per media unit.

The demand for network storage management applications that provide high data integrity is also increasing. To preserve a high degree of data integrity, these network storage management systems can make use of intelligent data storage devices that incorporate sophisticated MEMR techniques. These techniques can verify the integrity of stored data and are able to provide the user early warnings of data degradation through the network storage application. MEMR techniques have been implemented in data storage devices for many years. MEMRI information is defined to be data integrity metadata derived from MEMR techniques.

In 1995, organizations involved in the developing of optical tape technology expressed interest in addressing data integrity issues related to this emerging data storage technology. In August 1995 AIIM formed the Optical Tape Study Group (OTSG) to promote discussions among users and industry on media characteristics, testing methods, data integrity and future standards for optical tape technology. The Study Group attracted broad industry and user participation (*e.g*, Eastman Kodak Company, the Library of Congress, LOTS Technology, the National Archives and Records Administration, National Media Lab/Imation, National Storage Industry Consortium, NIST, Polaroid Corp., StorageTek, Systems Engineering & Security, Inc., Terabank Systems, and the Univ. of Arizona, Optical Sciences Center). This user and industry group expressed interest in developing a standard specifying MEMR techniques and the associated MEMRI metadata for optical tape. Data integrity issues, MEMR techniques and proposed MEMRI metadata have been discussed and analyzed in a number of papers submitted to AIIM OTSG (Podio [2]), (Silberstein [3]), (Podio [4]), (Manavi [5]), (Thibodeau [6]), (Vollrath [7] [8]).

The AIIM OTSG completed its work during 1997 by submitting to AIIM two project proposals for the development of ANSI standards: (a) a proposal for the development of an ANSI media interchange standard for digital data interchange in write-once read many times (WORM) optical tape cartridges (AIIM OTSG [9]) and (b) a project proposal for the development of an ANSI standard specifying media error monitoring and reporting information (MEMRI) to verify the integrity of stored data on optical tape media (AIIM OTSG [10]).

In August 1997, AIIM formed the Optical Tape Subcommittee C21.3 working under the Storage Devices and Applications Committee C21. This Subcommittee started work in November 1997 on the two proposed standards mentioned above. MEMRI's project scope is for optical tape drives. However it is the current thinking of C21.3 to extend the scope of the proposed standard to *any type* of sequential storage media if the magnetic tape drive industry wishes to join in the development of this standard.

The paper is organized as follows. Section 2 focus on the need for media error monitoring and reporting techniques and discusses the power of error correcting codes currently implemented in data storage devices. Section 3 describes existing network storage management data integrity standards. Section 4 describes an emerging standard that specifies metadata to determine optical tape data integrity (MEMRI) and the means of transporting the media error information in a technology-and-interface-independent manner.

2. Media Error Monitoring and Reporting and Error Correcting Codes

2.1 Need for Media Error Monitoring

The need for data integrity metadata is apparent. Error Correcting Code (ECC) systems implemented in data storage devices are very powerful. They are designed to correct burst errors and can also easily correct random errors. The user is undisturbed by (but also usually unaware of) the level of corrections taking place. Depending upon the number, length and location of burst errors; however, some of these errors may be uncorrectable. When errors become uncorrectable data loss will occur.

In the absence of early warning indicators of data degradation and other media error monitoring capabilities, data storage devices cannot provide network storage management applications with the information required to maintain data storage subsystems with a high degree of data integrity.

2.2 Error Correcting Codes for Data Storage Devices

One ECC commonly used on rewritable magneto-optical disks is an interleaved ECC (ISO [11]). In this type of optical disk, each sector contains 512 bytes which are 5-way interleaved. (If the sector contains 1024 bytes the format is similar but a 10-way interleave is used.) This example of a 5-way interleaved data field format where each of the five columns represents a codeword has been described in detail in (Podio, Vollrath, Kobler [12]).

Figure 1 depicts how bytes are interleaved within a 512 byte sector. As shown, other bytes, in addition to the 512 user-data bytes, are stored on the media. The bytes are recorded onto the disk from left to right and from top to bottom (SB_1, SB_2, SB_3, D_1, D_2, ..., $E_{4,16}$, $E_{5,16}$). Each of the five columns shown in Figure 1 represents a codeword. In this implementation, a Reed-Solomon code is used to calculate the ECC bytes, $E_{1,1}$, $E_{2,1}$, ..., $E_{5,}16$ for the first codeword and so on. Interleaving the bytes in different codewords reduces the probability of a burst error on the media (*n* consecutive bytes in error) to produce uncorrectable data.

The sector bytes in this interleaved configuration represent the following:

- D_1 to D_{512} are the user data bytes.
- $P_{1,1}$, $P_{1,2}$, …, $P_{3,4}$ are the data management pointers (DMP) bytes. These bytes are information for sector reallocation (the link between defective and replacement sectors).
- FF bytes. Two filler bytes.
- C_1 to C_4 bytes. These are cyclic redundancy check (CRC) bytes. CRC bytes are computed from the user, DMP and FF bytes.
- SB_1 to SB_3 bytes. These are bytes that synchronize the data signal and the drive clock.
- RS_1 to RS_{40} bytes. These bytes preserve synchronization within the sector.
- $E_{1,1}$, $E_{2,1}$, …, $E_{5,16}$∑ The ECC bytes.

		CODEWORD NO. ⇒	1	2	3	4	5	ROW NO.
SB_1	SB_2	SB_3	D_1	D_2	D_3	D_4	D_5	105
			D_6	D_7	D_8	D_9	D_{10}	104
			D_{11}	D_{12}	D_{13}	D_{14}	D_{15}	103
		RS_1	D_{16}	D_{17}	D_{18}	D_{19}	D_{20}	102
			D_{21}	D_{22}	D_{23}	D_{24}	D_{25}	101
			D_{26}	D_{27}	D_{28}	D_{29}	D_{30}	100
		RS_2	D_{31}	D_{32}	D_{33}	D_{34}	D_{35}	99
			...	...	...	...	...	...
		RS_{33}	D_{496}	D_{497}	D_{498}	D_{499}	D_{500}	6
			D_{501}	D_{502}	D_{503}	D_{504}	D_{505}	5
			D_{506}	D_{507}	D_{508}	D_{509}	D_{510}	4
		RS_{34}	D_{511}	D_{512}	$P_{1,1}$	$P_{1,2}$	$P_{1,3}$	3
			$P_{1,4}$	$P_{2,1}$	$P_{2,2}$	$P_{2,3}$	$P_{2,4}$	2
			$P_{3,1}$	$P_{3,2}$	$P_{3,3}$	$P_{3,4}$	(FF)	1
		RS_{35}	(FF)	C_1	C_2	C_3	C_4	0
			$E_{1,1}$	$E_{2,1}$	$E_{3,1}$	$E_{4,1}$	$E_{5,1}$	-1
			$E_{1,2}$	$E_{2,2}$	$E_{3,2}$	$E_{4,2}$	$E_{5,2}$	-2
		RS_{36}	$E_{1,3}$	$E_{2,3}$	$E_{3,3}$	$E_{4,3}$	$E_{5,3}$	-3
			$E_{1,4}$	$E_{2,4}$	$E_{3,4}$	$E_{4,4}$	$E_{5,4}$	-4
			...	...	...	...	...	...
			$E_{1,14}$	$E_{2,14}$	$E_{3,14}$	$E_{4,14}$	$E_{5,14}$	-14
		RS_{40}	$E_{1,15}$	$E_{2,15}$	$E_{3,15}$	$E_{4,15}$	$E_{5,15}$	-15
			$E_{1,16}$	$E_{2,16}$	$E_{3,16}$	$E_{4,16}$	$E_{5,16}$	-16

Figure 1. An example of a 5-way interleaved data field format where each of the five columns represent a codeword

The power of this interleave scheme and ECC is shown in the following example. If a burst error of length 16 occurs on the media (for example, from bytes D_{16} to D_{31}), each codeword (column) has no more than four bytes in error (caused by this burst error). The ECC can easily correct this burst error. This ECC implementation has the capability of directly correcting 8 bytes in error per codeword.

Different data storage devices utilize different ECC implementations. One standard (ANSI [13]) describes the use of two Reed-Solomon codes which operates between the drive data buffer and the read/write heads. The outer ECC appends 8 check bytes to each 128 byte data block and is defined as an RS (136, 128) code. The inner ECC encoder appends 8 check bytes to each 85 byte data block with two ID bytes. This is known as an RS (95, 87) code. Examples of other ECC implementations can be found in (ISO [14] [15]).

3. Network Storage Management Data Integrity Standards

3.1 MEMR – A Standard to Determine Optical Disks Data Integrity

In 1996 ANSI approved the only known (world-wide) voluntary industry standard specifying media error monitoring and reporting techniques to verify data stored on data storage media (ANSI [16]). This standard, developed under AIIM C21 Committee, "Storage Devices and Applications" and known as "ANSI/AIIM MS59" specifies a set of MEMR tools and the associated media error metadata to verify stored data on optical digital data disks. An accompanying Technical Report, known as "ANSI/AIIM TR39" (ANSI [17]), provides guidelines for the use of these MEMR tools and the associated metadata. An equivalent international standard based on ANSI/AIIM MS59 is also being developed. Users of optical digital data storage drives that implement the techniques specified in ANSI/AIIM MS59 can gather statistical information on media errors. This metadata allows them to highlight data integrity trends on particular selected disks or across their entire data sets. These techniques provide data recovery and media error monitoring tools with different levels of sophistication. ANSI/AIIM MS59 specifies two MEMR levels:

High level:

- A set of functional commands that are operating system (e.g., Unix, Windows™) and interface (e.g. SCSI-2, IPI) independent. In addition, this high level interface is optical disk media type (e.g, Write-Once Read Many Times or rewritable media) and size independent.

Interface level:

- An implementation of SCSI-2 (Small Computer Systems Interface – version 2) commands. These commands allow network storage management applications to include data verification tools at the SCSI level which are drive type and size independent, through the use of MEMR techniques.

Table 1 lists the types of data integrity metadata provided by MEMR techniques specified in the standard (ANSI/AIIM MS59). Table 2 describes the content of the technical report (ANSI/AIIM TR39). Standard metadata related to media errors allows network storage management applications to retrieve the same information even if storage subsystems consist of different optical disk drives. Decisions on the frequency of use of these tools are not specified in the standard.

Table 1 – Metadata Specified in ANSI/AIIM MS59 Standard

- Early warnings of an increasing number of media errors
- List of reallocated sectors
- Corrections above a defined media error level
- Total number of bytes in error, number of bytes in error per sector and maximum number of bytes in error in any sector codeword
- The uncorrected or corrected sector content
- Errors encountered reading header information such as the sector address, sector marks and synchronization signals
- Maximum length of contiguous defective bytes

Table 2 – Scope of ANSI/AIIM TR39

- Interpretation of the metadata provided by ANSI/AIIM MS59
- Statistical sampling methods
- Automation of media testing
- Graphical methods for media error representation
- Use of error distributions and statistical models to evaluate data integrity
- A summary of the ANSI/AIIM MS59 command set

3.2 Associated Industry Efforts

Two additional industry efforts are identified: (a) a proposed standard for tape drive management developed by Hewlett-Packard (Hewlett-Packard TapeAlert™ [18]); and (b) Imation's Media Performance Manager (Imation [19]).

TapeAlert™ is a tape drive status monitoring and messaging utility that makes it easy to detect problems, which could have an impact on backup quality. To reach agreement on an enhanced specification (Gold [20]), Hewlett-Packard has formed an industry Working Group, incorporating representatives from major tape drive and backup software companies. Imation's Media Manager is a storage management software product that allows system administrators to perform data storage management from the user's desktop in real-time. Media Manager was co-developed by Imation and Sterling Software.

4. MEMRI – An Emerging Standard to Determine Optical Tape Data Integrity

4.1 Scope and Objectives of the Standard

This standard is currently under development in AIIM Optical Tape Subcommittee C21.3. The current thinking of C21.3 is to establish an ANSI/AIIM standard specifying a set of MEMRI metadata to verify stored data on optical tape media and the means of transporting the media error information in a technology-and-interface-independent manner. The high-level interface approach proposed in this new standard is independent of the host operating system and the interface between the storage device and the host. It is also media type and size independent and it can be applied to systems that use any optical tape media size and type.

The draft standard (AIIM Optical Tape Subcommittee [21]) being developed specifies:

- A high-level interface that will allow a drive to indicate to a network storage management application whether a given media should continue to be used or should be replaced by fresh media. This information is an early warning of data degradation. The supplier of MEMRI compliance (*server*) can choose to offer this recommendation either with or without defining the underlying chain of reasoning that generated the conclusion.

- A standardized means of communicating the information that led to the drive to recommend media replacement. If a supplier chooses to define the underlying chain of reasoning that generated the early warning of data degradation, the MEMRI Standard supplies a standardized means of communicating the analyses that were used to generate the recommendation.

- The standard means of describing and accessing through a high-level functional interface, standardized or vendor-specific drive accumulators and registers that store media error information.

- A high-level standard set of metadata elements that represent the content of drive accumulators that store statistics about media errors.

Currently, the draft standard specifies transport functions (query, response, attention and clear attention) and the query layer (data types and volume-related queries). The transport functions and the query layer are briefly described below.

4.2 Transport Functions

The transport functions are the means by which the *client* communicates with the *server*. Standard communications needs are transacted across a given interface. The three proposed transport functions are:

- **Query.** Provide for communication of *queries* from the *client* to the *server*. These queries will consist of a string of characters taken from the *printable ASCII characters*. The client may send a query at any time.

- **Response.** Provide for communication of the *responses* to the *queries* from the *server* to the *client.* These responses will consist of strings of characters taken from the *printable ASCII characters*. The server must generate exactly one single response to each query sent to it by the client, and cannot otherwise generate responses.

- **Attention.** Provide for the *server* to indicate "attention" to the *client.* This attention will consist of a boolean indicator raised by the server. The server may assert attention at any time.

- **Clear Attention**. Provide for the client to recognize and clear the attention condition.

4.3 Query Layer

As currently specified, any query can have the string ".DATA_TYPE" appended to it. If the query is valid the server will respond with "INTEGER", "STRING", or "CONTAINER" to indicate the data type returned by that query. Table 3 below shows some examples using fictitious queries.

- **Integer Data Type**

- Values of integers (a 32-bit signed integer) are set by appending '=' and a value to the integer's query. Values can be set either in decimal or in hex, using the "C" notation. When an integer value is queried, the server normally responds with its decimal value. If the query has .AS_HEX appended to it, the value will be returned in hex.

- **String Data Type**

- It is a string of printable ASCII characters. Values can be set by appending '=' and a value to the string's query. Strings can be set to any sequence of byte values using the *printable characters of the ASCII character set* and *escape sequences.*

Table 3 – Fictitious Query Examples

Client	/MEMRI/X.DATA_TYPE	The client requests the data type of X.
Server	INTEGER	The server responds with the data type.
Client	/MEMRI/VOLUMES.DATA_TYPE	The client requests the data type of VOLUMES
Server	CONTAINER	The server responds with the data type.

- **Container Data Type**
- An object capable of holding and organizing several sub-objects. The sub-objects are typically related to each other. Container values cannot be set in a single step. Container values can be polled.

4.4 Volume-Related Queries (/MEMRI/VOLUMES)

The /MEMRI/VOLUMES object is a *container*. The members of this container will be a list of all volumes for which this *server* has MEMRI information available. Table 4 shows examples of volume–related queries. The volume-related queries include Volume Parameters ***(/MEMRI/VOLUMES/nn/PARAMETERS)*** and Volume Media Parameters ***(/MEMRI/VOLUMES/nn/PARAMETERS/MEDIA)***:

- Volume Parameters ***(/MEMRI/VOLUMES/nn/PARAMETERS):***
- This container object contains all *parameters* for the specified volume. Compliant devices must provide this container object and it must contain the CASETYPE, MANUFACTURER and MEDIA sub-objects. Additional sub-objects can be present.
- Volume Media Parameters ***(/MEMRI/VOLUMES/nn/PARAMETERS/MEDIA)***:
- This container object contains all media-related *parameters* for the specified volume. Compliant devices must provide this container object and it must contain the TYPE and MEMRI_STORAGE_LOCATION sub-objects. Additional sub-objects can be present.

Table 4 - Examples of Volume–Rlated Queries

Client	/MEMRI/VOLUMES	*The client requests a list of all volumes*
Server	{AA23 THX-1138 “REFERENCE TAPE”}	*The server returns the list of volumes*
Client	/MEMRI/VOLUMES/THX-1138	*The client accesses information about volume “THX-1138”*
Server	{PARAMETERS EVENTS READINGS ACCUMULATORS RECOMMENDATIONS}	*The server responds by listing all of the areas for which information is available concerning the volume*

5. Conclusions

To provide for high accessibility and high data integrity of stored data, network storage management applications require reliable indicators of data integrity from data storage devices. To satisfy this need, data storage devices are becoming more intelligent. Reliable indicators of data degradation include early warnings, media error monitoring and reporting techniques and data integrity metadata. Error correction activity generally occurs without notification to the host. With data integrity metadata available at the network storage management level, these applications can monitor the degree of data degradation and error correction activity taking place in data storage devices. Highly reliable storage systems can use these network storage management applications to significantly decrease the possibility of data loss. Network storage management industry standards that specify MEMR techniques and the associated data integrity metadata are emerging. An industry standard to verify the data integrity of stored data on optical digital data disks exists and others standards are being developed.

References

[1] M. Peterson. *Opportunities in Network Storage Management*, A document produced for Avail Systems, Nov. 1995, http://www.sresearch.com/search/105275.html.

[2] F. Podio, *Optical Storage Media Data Integrity Issues*, Association for Information and Image Management International, Optical Tape Study Group, AIIM OTSG[1]/95-007, Aug. 1995.

[3] S. Silberstein. *Analysis of ANSI/AIIM MS59, Media Error Monitoring and Reporting Techniques to Verify Stored Data on Optical Digital Data Disks,* AIIM OTSG/95-008, Aug. 1995.

[4] F. Podio. *Monitoring and Reporting Techniques for Error Rate and Error Distribution on Optical Disks Systems*, AIIM OTSG/96-013, Feb. 1996

[5] M. Manavi. *Tape Defect Size Characterization*, AIIM OTSG/96-015, Feb. 1996

[6] K. Thibodeau, *Position Letter in Support of the Development of Media Error Monitoring and Reporting Standards for Optical Tape*, AIIM OTSG/96-029, Apr. 1996.

[7] W. Vollrath. *MEMRI (TM), Media Error Monitoring, Reporting and Information*, OTSG/96-030, Apr. 1996.

[8] W. Vollrath. *MEMRI (TM) Information Structure Proposal,* OTSG/96-050, Oct. 1996.

[9] AIIM Optical Tape Study Group. *Project Proposal for the development of an American National Standard, Digital Data Interchange in 12.65 mm (0.5 in) Write-Once Read Many Times (WORM) Optical Tape Cartridges*, April 15, 1997

[10] AIIM Optical Tape Study Group. *Project Proposal for the Development of an American National Standard, Media Error Monitoring and Reporting Information*

[1] Association for Information and Image Management International, Optical Tape Study Group documents can be obtained from Fernando Podio, NIST.

(MEMRI) to Verify the Integrity of Stored Data on Optical Tape Media, AIIM OTSG/97-022 Rev. 1 (Revised August 7, 1997).

[11] International Standards Organization (ISO). *Information Technology - 130 mm Rewritable Optical Disk Cartridge for Information Interchange*, ISO/IEC 10089-1991.

[12] F. Podio, W. Vollrath, B. Kobler, *Media Error Monitoring and Reporting Information (MEMRI) - Metadata for Intelligent Digital Data Storage Devices*, Proceedings of the Second IEEE Metadata Conference, September 1997, http://computer.org/conferen/proceed/meta97/papers/fpodio/fpodio.html.

[13] American National Standard for Information Systems. *Information Technology - Helical-Scan Digital Computer Tape Cartridge, 12.65 mm (0.498 in), for Information Interchange*, ANSI X3.267-1996

[14] International Standards Organization (ISO). *Information technology -- Data interchange on 90 mm optical disk cartridges -- Capacity: 640 Mbytes per cartridge*, ISO/IEC 15041:1997.

[15] International Standards Organization (ISO). *Information technology -- 3,81 mm wide magnetic tape cartridge for information interchange -- Helical scan recording -- DDS format using 60 m and 90 m length tapes*, ISO/IEC 12247:1993.

[16] American National Standard for Information Systems. *Media Error Monitoring and Reporting Techniques for Verification of Stored Data on Optical Digital Data Disks*, ANSI/AIIM MS59, March 1996.

[17] American National Standard for Information Systems. *Guidelines for the Use of Media Error Monitoring and Reporting Techniques*, ANSI/AIIM TR39, January 1996.

[18] Hewlett Packard Co. TapeAlertTM - A New Approach to Tape Drive Management, http://www.hp.com/tape/tapealert/ta.html.

[19] Imation Corp. *The Imation Media Performance Manager*, Imation's Press Release, April 1997.

[20] S. Gold. *TapeAlert Specification Version 1.2*, September 1997. http://www.hp.com/tape/tapealert/taspec12.html

[21] AIIM Optical Tape Subcommittee. *Preliminary MEMI Standard*, AIIM C21.3/97-013.

Page intentionally left blank

Evaluating RAID in the Real World

Ian Bird, Rita Chambers, Mark Davis,
Andy Kowalski, Bob Lukens, Sandy Philpott,
Roy Whitney
SURA/Jefferson Lab
12000 Jefferson Avenue,
Newport News, VA 23606
{igb,chambers,davis,kowalski,rlukens,
philpott,whitney}@jlab.org
Tel +1-757-269-7514
Fax +1-757-269-7053

Abstract: High energy nuclear physics experiments at the Thomas Jefferson National Accelerator Facility ("Jefferson Lab") will have a data collection rate of 10 MB/second, generating 1 Terabyte (TB) of raw data per day of accelerator running, and a similar amount after processing. The requirement for on-line disk storage for raw and reduced data sets will exceed 1 TB during 1998. This paper discusses the on-line storage strategy that provides both high performance as well as high capacity, and focuses on the in-house evaluation of RAID (Redundant Arrays of Independent Disks) systems to fulfill the needs of both data acquisition and analysis.

1 Introduction

The Thomas Jefferson National Accelerator Facility ("Jefferson Lab") operates a 4 GeV continuous wave electron beam accelerator for nuclear physics research for the U.S. Department of Energy. The largest experiments will generate close to 1 TB of data per day, for some 120-150 days of accelerator running per year. In addition, the reconstruction of this data will yield a similar amount of processed data. Both the raw and analyzed data sets will be stored in an STK silo, using RedWood tape cartridges and drives. The data is collected on data acquisition computers close to the experiments and copied to the Computer Center and into the tape silo over a 1-km long dedicated fibre-channel connection. Once the data is in the silo, it may be copied back out for processing on a "farm" of Unix workstations. Results of this analysis are also stored in the silo. During the last year the Computer Center has gone through 2 major RAID procurements for storage to support different parts of the data path – areas for fast staging of files between tape and the farm, and for large analysis work areas. As part of those procurements we asked the competing vendors to bring to the lab examples of their systems for demonstration and testing. In this paper we present the process we went through to develop the tests that allowed us to determine the behavior of those systems in our environment.

1.1 Why RAID?

The sizes of the data sets that we deal with are largely driven by the data rate. A single hour of running of the largest experiment generates around 50 GB of data. Since there are several things that impose a 2 GB file limit (32 bit OS, tape storage manager software), the current processing is restricted to files of this size. The design goals, however, anticipate the requirement to handle significantly larger file sizes. The RedWood tape cartridges have a capacity of 50 GB and the drives can transfer data at a rate in excess of 10 MB/s. In order to optimize the use of the drives and to allow the tapes to stream, the ideal access method is to stage data from tape to a staging disk and then to transfer from that staging area to the destination. The data processing farm will eventually approach 50-100 CPUs running in coarse parallelism accessing the data and storing the results via the staging areas. Thus the staging device needs to be capable of reading or writing at 10 MB/s with simultaneous remote accesses (which are at transfer rates limited by the network), so that we require not only a high data stream performance but also a high aggregate performance. In addition we require ideally some 150 GB of staging area per tape drive.

The other main use for high performance redundant storage is in the need of the experiments to have available on-line large data samples for analysis, visualization and algorithm development. Typically a single analysis (of which there may be many simultaneously) will demand some 50-100 GB of data to be on-line and randomly accessible from both the batch processing farms as well as central analytical systems.

In both cases, the need is for both performance and large data set sizes. Only RAID systems provide these capabilities.

1.2 Procurement Process

During the last year, we have undertaken two major procurements. The first was for host attached RAID for use primarily as the fast tape staging space, but also to provide an initial implementation of some of the work areas. Here, performance - both throughput and aggregate rate, was the critical factor. A certain level of configurability was also desirable.

The second procurement was specifically for network attached (NFS) RAID for use as work areas for large data sets. In this case management and configurability of the space was the key. These areas need multiple, reliable, network accesses as well as the ability for the space to be managed with group and directory quotas. Good performance was also an issue.

In addition to standard product research both before, and as part of, the formal procurements, the Jefferson Lab Computer Center chose to include on-site benchmarking as a requirement of the solicitation process. Vendors choosing to participate were required to bring to the laboratory a system closely configured to the system(s) offered in their proposals. Vendors were given advance knowledge of the tests to be performed and the local system and network configuration. Encouraged to pre-configure their RAID systems and to arrive early for system uncrating and installation, all vendors were able to complete the locally administered test suite in well under 4 hours, and generally used the remainder of their scheduled half-day time slot to demonstrate other product capabilities to site staff.

The tests were run in advance by the Computer Center staff on existing hardware in order to understand the infrastructure limitations as well as to determine the time constraints of the tests, calculate baseline results, and ensure that the test results were meaningful.

1.3 Importance of Testing

A relatively large number of manufacturers and integrators now offer RAID products that support storage solutions from the low-end (20 GB) up to multi-Terabyte range, provide a variety of combinations of hardware and software-based RAID architectures, and offer a wide spectrum of native redundancies. The challenge to the consumer in selecting the products best suited in terms of performance, capability, and cost for their specific application is substantial. The market is characterized by product announcements with the next generation always on the horizon, pricing models on a steady downward slope, and vendor capabilities that can in fact vary significantly. A complete evaluation will include information from vendor-based and trade journal product comparisons as well as recommendations of professional counterparts. Each of these sources has limitations that could be critical particularly given the significant dollar investment (*i.e.*, even at 50 cents/MB, a TB costs $500 K!) and vital role the equipment plays in the business of the institution. Vendor product specifications and technical white papers provide only the vendor's eye view of their product line. Standardized benchmark suites (SPEC LADDIS, etc.) report what in fact are vendor-generated results of "standardized" tests, and allow a test environment and configuration which can vary dramatically from the consumer's intended application. Neither consultation with industry counterparts nor examination of non-biased commercial product evaluations is a guarantee that the equipment will in fact perform suitably in your specific environment and application. On-site testing, with real data and an environment customized to mirror the actual production application, whenever feasible is the ideal culmination of a product evaluation process.

The challenge for the RAID procurements described in this paper, beyond the obvious logistics of performing in-house testing of multiple vendor boxes, was to design tests that could be run in reasonable time frames on existing non-production equipment (in a relatively small computing center), and that in fact provided a realistic picture of how each box would perform in the lab's environment as well as demonstrate the relative capabilities of multiple vendor offerings. While it was important to a fair evaluation process to provide a determinant testing environment (isolated from network broadcasts, for instance), it was equally important to determine how the box would perform in the real world, specifically "our" real world. The tests needed to test for the "right" thing -- would the test in fact be limited not by the vendor's equipment but by the network itself or by the receiving batch node? Would the tests in fact show no measurable differences between the vendor systems or would the differences determined be invalid indicators of the viability of the proposed solution?

2 Analyzing the Data Path and Developing the Tests

The tests that were used in the evaluation were, in the end, relatively straightforward. However, there are a variety of factors that need to be considered in order to arrive at a series of tests that not only demonstrate that the system has the desired capabilities, but are also feasible in a limited time. In the following we give an outline of what those considerations are.

The first step in the analysis is to really understand the data path. This may not be so obvious, especially if the system is new and this analysis is actually part of the initial design process. For example, at first glance the cost/performance ratio for a farm of "pizza box" processors seems to be far better than that of a few large SMP systems. However, taking into consideration the actual data rates and consequent I/O requirements, and then taking into account networking costs, *i.e.*, considering the system as a whole, is that solution still the most cost effective? Consider also, any hidden assumptions. For example, at Jefferson Lab it had long been assumed that ATM would be the only way to deal with the high bandwidth and throughput requirements. However, it became clear that it was much simpler and cheaper to use switched Fast Ethernet, and that the necessary aggregate performance could be easily achieved.

A high level schematic of our actual data path is shown in Figure 1.

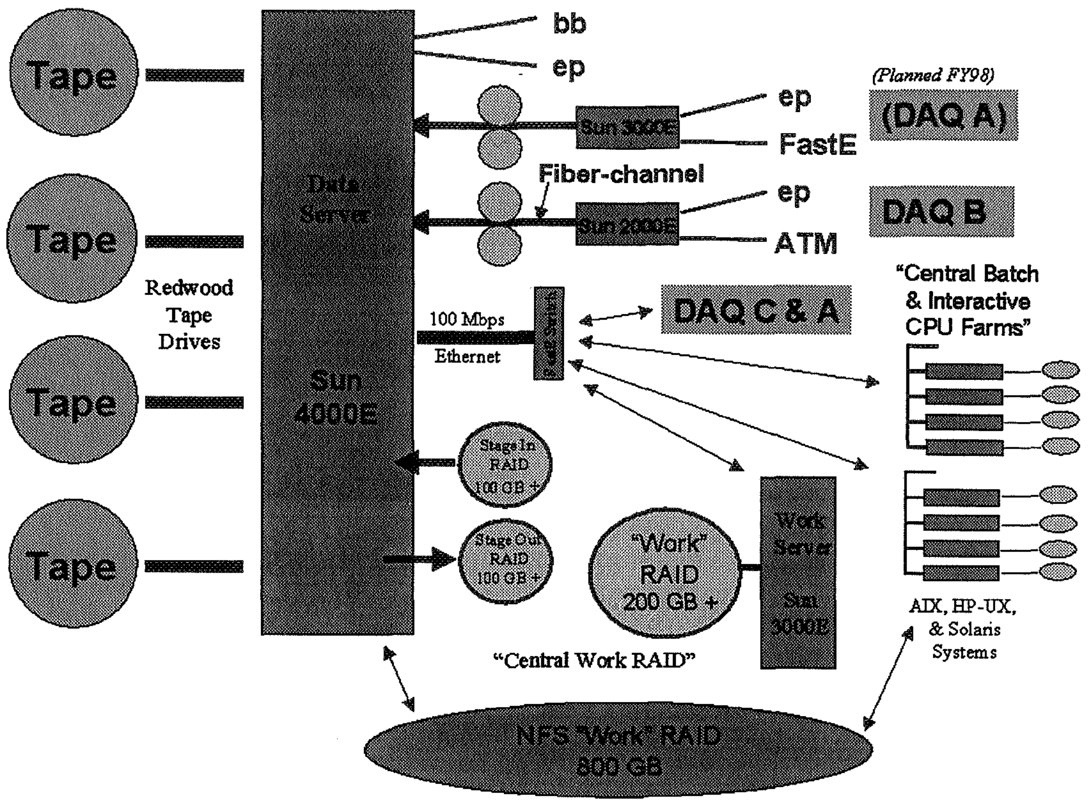


Figure 1. Data Path

Data arrives from the data acquisition systems of the experiments either directly from staging disks connected over 100MB/s Ethernet, or staged over Fiber-Channel and copied into the tape silo via a Sun E4000 data server. The batch processing nodes are dual processor UNIX workstations, with locally attached disks for local processing. This was determined to be the most cost-effective solution. The RAID staging disks attached to the E4000 are used to stage data both into and out of the tape silo. It is these staging areas that were the primary focus of the first procurement. For these areas the requirement on the RAID system was that it should accept data at 10MB/s – matching the capability of the RedWood drives, whilst simultaneously allowing access to and from the farm processing machines at network speeds. Note here, that all these accesses, both to the silo and over the network, are all going through the data server machine. This was a potential bottleneck, although that machine has multiple network and I/O interfaces.

The second procurement was for an NFS RAID server to provide data areas that will be used to hold intermediate processing results from the farm nodes allowing interactive analysis of ongoing processing, as well as for areas to hold large data samples for later analysis by multiple researchers. The requirements for these areas were somewhat different from the staging space. Rather than total throughput, the main demand was that the system be able to provide many simultaneous network accesses with a performance that should be limited by the networks rather than by the RAID system itself.

In order that the tests measure the performance of the RAID systems and not simply measure a bottleneck like network performance, some care must be taken with the design. Some baseline measurements are essential. Such measurements determine exactly where the bottlenecks are – are the I/O adapters, the remote disk, or the network itself the limiting factors? They also give a base measurement with which to compare the test performance of the new systems. In our case these preliminary tests were made using groups of single disks running software RAID (Solaris DiskSuite) to ensure that the disk performance was really the limiting factor rather than one of the other limits. In that case there would be no reason to use expensive hardware RAID. Furthermore, some tests serve simply to qualify the RAID system – does the equipment pass the test or not? In our case, could the RAID match the 10 MB/s transfer rate of the RedWood tape transport? Would the equipment match or improve on the throughput achieved by the software RAID at each point in the data handling - from DAQ to RAID staging (via Fibre-Channel as well as network), to/from the RedWood tape transports, and collection to/from a network-accessed batch processor? Other tests can be used to discriminate between systems. Tests of transfer rates between memory and RAID or tape, and simultaneous transfers to multiple remote machines will generally provide such data.

In our setup, the parts of the data path that provide real tests were between the RedWoods and the RAID, from the RAID to a remote networked host, and from the RAID to several remote hosts. It is important also to test the data transfer in both directions as the reading and writing performances of the RAID systems are potentially very different. The tests should also be under controlled conditions. This is particularly important in terms of the load on the data server host, the remote hosts, and the network traffic. All the tests that were performed with the RAID systems were with unloaded hosts and quiet networks. However, as part of the test design we also made comparisons between loaded and quiet systems and with real network traffic.

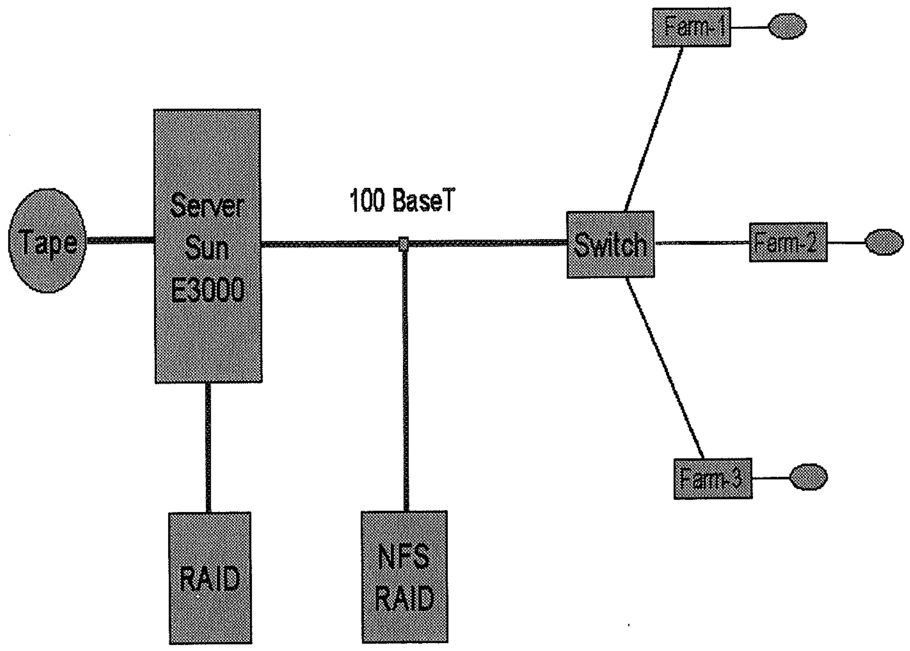


Figure 2. Test setup

For the tests themselves it is essential to limit them to testing what should be tested and not try to do too much. For example, even though in our production environment we will always be handling 2 GB files, we soon realized that for most tests 1 GB files were sufficient, and for other tests involving network transfers 500 MB was enough. Similarly, tests involving several remote hosts give as much information with 3 simultaneous transfers as with 10, since at some point something other than the RAID device becomes saturated. Most of these factors can be estimated ahead of time by running the proposed tests on existing equipment. Once the system has reached an equilibrium state no more information can be extracted – those are the data rates and there is little reason to run the test for a further 20 minutes.

The test programs were Perl scripts which facilitated changing the source on the fly and allowed its execution on multiple platforms without recompiling. For the read tests, the data was transferred to a 256 KB memory buffer and then discarded. For write tests, the same 256 KB buffer was written repeatedly out until the desired file size was reached. The data used was a sample of real physics data. It was realized early on that system generated pseudo-data could not be used as that data was easily compressible, gave very optimistic test results and did not represent the real situation at all. The same script was used for the tape to memory, memory to tape, and tape to disk tests. The network transfers were done with a version of rcp, NFS being far too slow for our needs for the fast staging space. The test system setup (See Figure 2) used a Sun E3000 as the test server for the direct attached RAID, with connections to both the STK RedWood drive as well as a dedicated 100BaseT Ethernet network. Three IBM RS6000 systems were used as the batch farm nodes. A similar environment was used with the Network Attached RAID tests (See Figure 2). You can view a copy of the Perl script we used and descriptions of the specific tests used in the two procurements at the following URL: http://www.jlab.org/ccc/gsfc/raidtest.html.

The test results were all logged automatically to create a permanent record of the test. The individual test results together with an evaluation sheet were returned to the vendor at the conclusion of the test. As part of the evaluation, the staff also considered demonstrated administrative functionality, as well as how easy it was for the vendors to install their systems in our environment, and how long it took to format and build the file systems.

There were some practical logistical considerations. We provided the vendors with the tests ahead of time, so that they could consider how best to configure their systems. However, we were very careful to ensure that the tests were fair, and that comparisons were only made of similar capabilities. The vendors were given a setup time half a day in advance of the testing, and a total test period of 4 hours. In general the tests were run well within that time and they were able to demonstrate other features and enhancements of the systems.

3 Results

The performance test results were highly effective discriminators in the solicitation process we describe for RAID systems and in fact were useful to both the vendors as well as Jefferson Lab. The testing process in the first procurement was completed in advance of the final bid submission date, and based on their results, several vendors opted not to continue in the proposal process based on results that clearly did not meet the minimum specifications of the procurement. On our part, the computing staff who assisted with the setup and configuration of the systems as well as the actual running of the tests, gained an in-depth knowledge of the architecture and capabilities of the systems being proposed. This perspective proved invaluable in the later evaluation of the written proposals.

The results of the tests are shown in Figures 3 and 4. The host-attached RAID tests (Figure 3) were: 1) copying a 1 GB file from memory to the RAID; 2) copying 1 GB from RAID to memory; 3) three simultaneous copies from memory to disk – the results shown are the average rate per process; 4) three simultaneous copies from disk to memory; 5) copy a 1 GB file from disk to tape; 6) copy from tape to disk; 7) copy a 1 GB file from memory to disk and simultaneously copy a file from disk to tape; 8) simultaneous copies from disk to memory and from tape to disk; 9) copy a file from RAID to the local disk on a remote machine and simultaneously copy a file from tape to RAID.

Figure 4 shows the NFS RAID tests: 1) writing to RAID simultaneously from 3 machines; 2) three machines simultaneously reading from the RAID; - in both these cases the results shown are the total throughput; 3) writing a single file to disk; 4) reading a single file from disk; 5) simultaneously 2 machines reading from, and 2 machines writing to the RAID.

The test results demonstrated that there were genuine differences in the performance of the boxes we tested, and that in most cases, the results were significantly different from the published performance specifications claimed by the vendors. Although the equipment we tested was all within the same general class of RAID devices (high performance, high capacity, moderate redundancy), there were wide variations even in the capability of the RAID boxes to meet the qualifying 10 MB/s throughput of the RedWood tape drives. Our tests required the vendors to provide at least SCSI II Fast Wide interfaces and vendors were allowed the opportunity to demonstrate higher performance I/O if they provided all required host adapters. Two vendors provided adapters and demonstrated UltraSCSI connections. On average, this class of RAID equipment was able to provide SCSI II data transfers in the range of the 10 MB/s required, with writes at a slightly lower rate than reads as expected. The UltraSCSI performance was at least 5 MB/s faster, with the winning box (UltraSCSI) achieving 16.5 MB/s with writes (RAM to RAID) and 23 MB/s with reads (RAID to RAM). The throughput for the Network Attached RAID units (NFS servers) averaged around 3-5 MB/s over the 100 Mbit Ethernet test network for a single transfer, with the winning box achieving aggregate data transfers in the 6-8 MB/s range. Only two vendor boxes were tested in the Network Attached RAID procurement due to standardization requirements on the site, with little difference between their performance.

There were several surprises both in the achieved performance as well as the vendor configurations. We found that on several transfers, there were real differences depending on the direction of the transfer, and the differences were not consistent across vendors. Some vendor boxes passed the tape to RAID test but failed the RAID to tape test; for others, the outcome was reversed. The differences are in part due to the fact that in some systems write operations are given higher priority than reads; other systems respond immediately once the data is written to cache and before it is flushed to disk

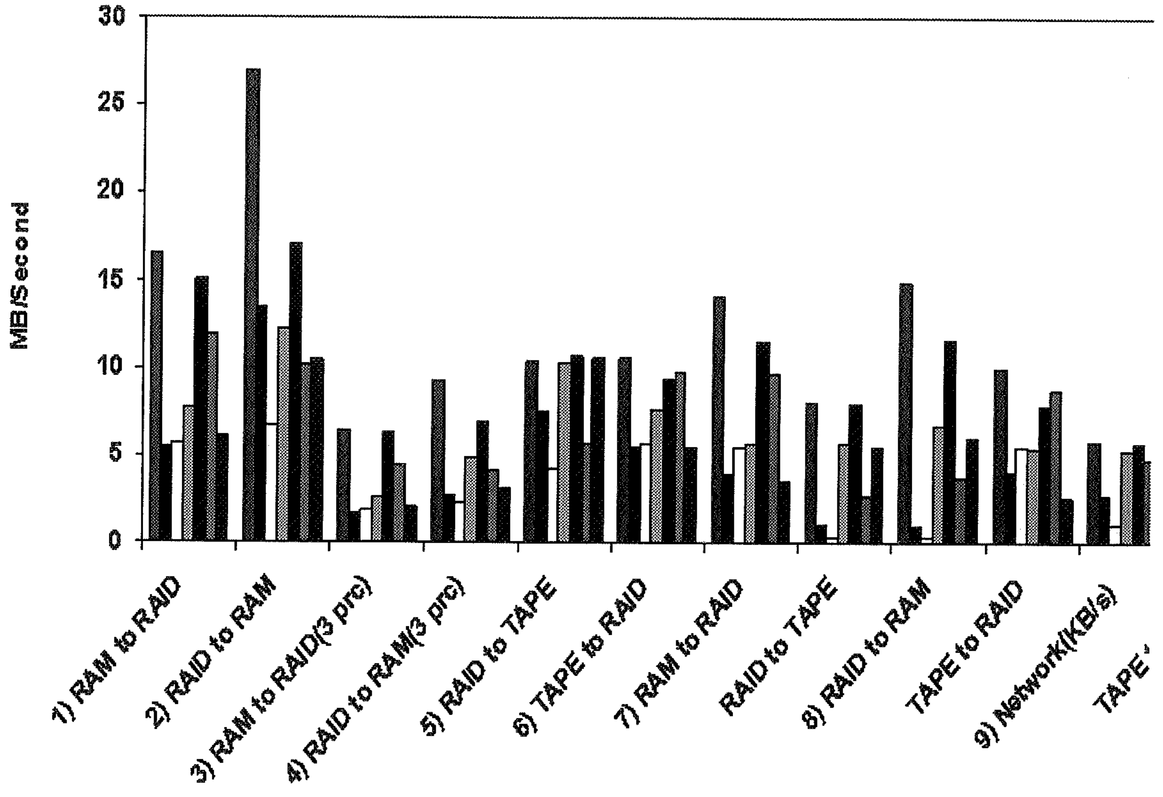


Figure 3. Host-attached RAID results

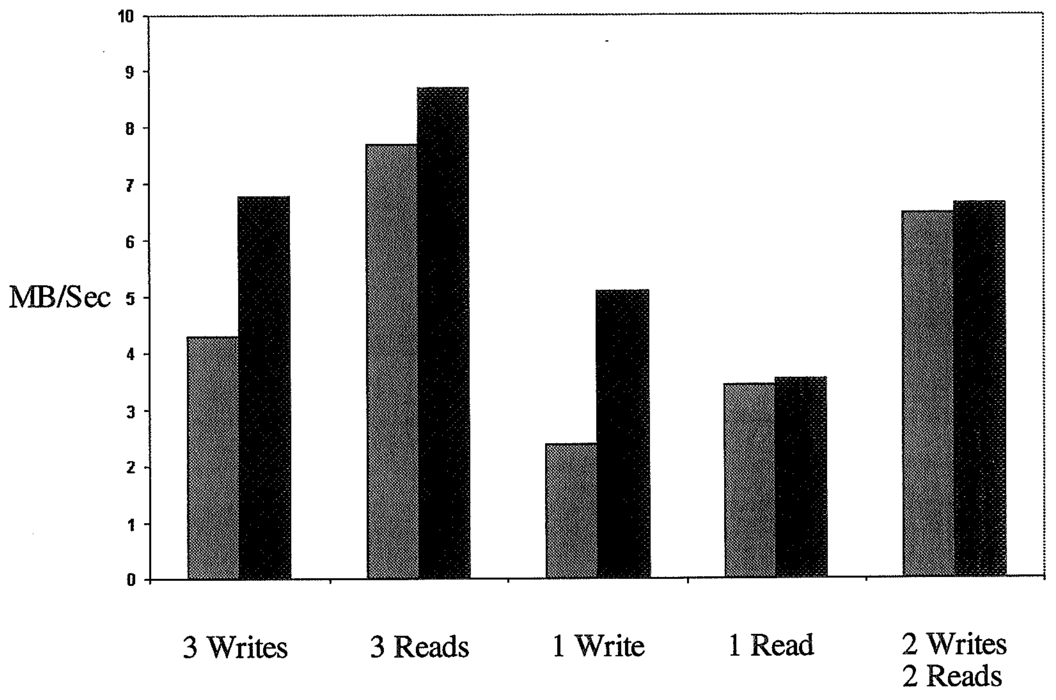


Figure 4. NFS RAID test results

Performance generally scaled well for multiple simultaneous transfers up to approximately 3 nodes, where the limiting factor was the transfer rate to the local disk on the batch node. Beyond 3 nodes, we found that the limiting factor was in fact the RAID system, with some boxes achieving overall higher aggregates than competitor equipment. Another test which provided real discrimination between the vendor boxes was a test that involved simultaneously moving 1 GB of data from a RedWood tape drive to the RAID system while reading a 1 GB file of physics data from the RAID into memory and vice versa. The data rate from the RAID system to memory should be the greater since it is not limited by the tape drive. This test for some equipment showed reductions in the data rates to and from the tape drives if other processes were heavily using the RAID system. The data rates to and from the tape drive were understandably lower than performing a sole tape test, but the better RAID systems had high data rates for both processes. Similar simultaneous tests found that RAM to RAID performance on some of the RAID systems degraded when coupled with network transfers. Our tests also revealed some tuning factors we can use to optimize our data transfers. Repeated testing during test suite development revealed that a blocking size of 256 KB provided the best overall performance on the RedWood tape transports. Furthermore, it was found that the amount of data transferred would affect the overall throughput. Large data sets completely fill buffers and slow the rate. On the other hand, small data sets end up measuring the throughput of the buffer or cache and not the RAID itself.

Vendor configurations proved equally enlightening. We found repeated examples of vendors over-tuning their equipment, which in some cases even resulted in instability and emergency reconfigurations. Our test team discovered that more than one vendor had attempted to skew the results by generating the test file system on only the outer tracks of the disks (i.e. the "sweet spot") or writing data to the raw device as opposed to a file system, both obviously unattainable in a standard production environment. One further "unattainable" specification was revealed by the vendor who explained that the reason their results did not approach the published specifications was that their in-house testing reported only RAM to cache testing, and not the final data rate achieved when storing to the hard disk. Although many vendors requested to use software RAID across multiple hardware-based RAID volume sets to demonstrate the final transfer rates obtainable, our test plan called for as much apples to apples testing as possible, not to mention the "keep it simple" goal. (As an aside, the use of software RAID in this fashion may be worth considering to achieve the high parallel transfer rates that may be required in the Jefferson Lab environment. At the moment, the increased liability in the event of multiple drive failures, plus the increased complexity of the configuration, advise against this option.)

4 Conclusions

In conclusion, the evaluation of commercial RAID offerings was significantly enhanced by the effort invested to test the equipment in the actual Jefferson Lab computing environment. The testing found real differences between vendor boxes, not only in their architecture and levels of flexibility and redundancy, but also significant variations in the performance of a variety of tests selected to mimic the stress points in our data handling operation. The preparation and logistics, while not insignificant, were well repaid by the knowledge we gained regarding the alternatives proposed in the RAID solicitations as well as potential tuning optimizations realized for our environment. In each procurement, the vendor that won was a clear winner for our environment.

The most important lesson learned is the realization that commercially reported performance results are as a rule highly skewed, the result of optimizing the box in a way few real world applications could, in order to maximize the test. The Jefferson Lab evaluation team uncovered multiple examples even in our own on-site testing process of results invalidated by the vendors' attempt to "ace the test." Building file systems on only the sweet spot on the disks, reporting results with redundancies turned off and non-RAID protected configurations, and even reporting transfer rates actually generated by cache-target transfers are but a few of the techniques vendors use to produce standardized results that bear little resemblance to the actual performance the customer will experience. The bottom line for high dollar, high performance RAID systems is that a fairly simple set of tests designed around your specific application may be the best performance indicator available. Testing in the "real world" is the one test you can't afford to forego.

References

[1] This research was sponsored by the Department of Energy contract DE-AC05-84ER40150. Views and conclusions contained in this report are the authors' and should not be interpreted as representing the official opinion or policies.

Long-Term File Activity Patterns in a UNIX Workstation Environment

Timothy J. Gibson and Ethan L. Miller [1]
Department of Computer Science and Electrical Engineering
University of Maryland - Baltimore County
1000 Hilltop Circle
Baltimore, MD 21250
{tgibso2, elm}@csee.umbc.edu
Tel: +1 410 455-3972
Fax: +1 410 455-3969

Abstract: As mass storage technology becomes more affordable for sites smaller than supercomputer centers, understanding their file access patterns becomes crucial for developing systems to store rarely used data on tertiary storage devices such as tapes and optical disks. This paper presents a new way to collect and analyze file system statistics for UNIX-based file systems. The collection system runs in user-space and requires no modification of the operating system kernel. The statistics package provides details about file system operations at the file level: creations, deletions, modifications, etc.

The paper analyzes four months of file system activity on a university file system. The results confirm previously published results gathered from supercomputer file systems, but differ in several important areas. Files in this study were considerably smaller than those at supercomputer centers, and they were accessed less frequently. Additionally, the long-term creation rate on workstation file systems is sufficiently low so that all data more than a day old could be cheaply saved on a mass storage device, allowing the integration of time travel into every file system.

1. Introduction

Physical storage devices have long been the slowest components of any computer system. While disk and tape storage devices have improved in the last decade, their performance has not kept pace with rapid increases in processor speed. This presents a challenge to storage system designers because faster CPUs encourage both more and larger files, placing a higher demand on the file storage system. This problem has long been an issue for supercomputer centers which have always had to manage huge quantities of data in large files, but the recent advances in CPU performance have brought traditional supercomputer power to the desktop. With greater power comes more and larger files. System designers must insure that workstation file systems can keep up with the increased bandwidth that this change requires.

During the last decade, CPU speed has increased approximately fifty-fold, and semiconductor memory capacities and speeds have also improved significantly. Typical workstation disk storage capacities have increased from tens of megabytes to several gigabytes in the same time period. However, disk access speeds have only improved by a factor of three [2]. The result is disks storing more and more data — being accessed more frequently by ever faster processors — but with disk access and transfer rates only slightly better than they were ten years ago. Current computers incorporate several semi-conductor cache levels and use complicated caching policies to overcome this disk bottleneck [2, 3]. At the same time, file systems are being developed to make tape migration transparent to users [4], and near-line tape robots continue to drop in price. Soon, it will be possible for smaller computing centers[1] to use on-line and near-line tape systems — or some other high capacity medium — to store files which are seldom used. Consequently, long-term disk access patterns and storage requirements will affect more and more users, making the study of these patterns increasingly important.

To study disk activity, we developed a system which allows administrators to collect file system traces at the user-level without modifying the operating system kernel. Additionally, the collection system can be run without tracing activity appearing in the traces. This means that data can be gathered without the collection process directly influencing the experiment.

This paper is organized into four sections. We begin by reviewing previous work by other authors. Monitoring disk activity is not a new activity and was first done in the early 1980's. Next, we discuss the data collection tools. These tools differ significantly from previous tools used in earlier studies. The tools track file system activity at the inode and logical filename level, providing detailed activity logs of file creations, deletion, modifications, etc. in a manner different from earlier studies. After discussing the tools, we analyze several months worth of data the tools provided. This analysis provides some interesting conclusions that reinforce earlier conclusions about file activity, and provide new insights about file modifications. Finally, we discuss the direction of out future research, which is centered on using the trace data we are collecting to drive a long-term storage and migration simulator.

2. Previous Work

Researchers have looked at disk or tape storage systems in the past. However, the nature of computing changes continually, so storage systems need periodic, if not constant, study. In the 1980's, both Smith [5], and Ousterhout, *et. al.* [6] made detailed studies of file activity on computing systems. While their observations are still useful, some of the underlying structure has lost relevance. For example, Smith primarily observed text-based user files for thirteen months; the size and nature of today's multimedia files, unforeseen when Smith collected his data, are much different from the text-based files so common on systems 15 years ago. Ousterhout's very detailed file traces were conducted over three to four day periods at a fairly low level. While Ousterhout's work is very useful, his traces were collected over such a short time that long-term trends cannot be predicted from his data. Baker's distributed file system activity logs from 1991 [7] update Ousterhout's work, concentrate on physical disk activity, and have the same short trace periods as Ousterhout's traces.

Other studies, [8, 9, 10, 11] are directly applicable to supercomputing centers, but may not apply well to smaller commercial and academic computing centers. Both the size and number of files at supercomputing centers far exceeds "normal" computing activities; more importantly, supercomputing centers usually have large tape libraries with tape robots providing near-line storage for hundreds of terabytes of data files. In contrast, most computing centers do not have tape robots and only use tape for archiving purposes.

Our work most closely resembles Strange's disk studies from 1992 [12]. We collect much of the same information as Strange, and in fact corroborate a good number of his findings. However, the traces used for this paper cover twice the time period as Strange's traces and provide additional information that Strange's did not.

We propose to study long-term file access and file usage patterns at both supercomputing facilities and on "normal" file systems — expanding the work done by previous authors. We begin our study of normal systems by examining workstation computing clusters and updating the state of research on disk storage systems in a networked environment.

3. Tools

The most basic part of the system is a modified GNU `find` utility we call `ftrace`. This program collects information on all the files in a file system or directory. The information includes the file's inode number, size, name, access time, inode create time, modification time, owner, and group. The file's name can be collected with or without the path, and either scrambled with MD-5 for privacy purposes or represented as plain-text. For example, `/JNsBhKWReJ4/9o9JCYa7VFY/2IzA6T74FOM` is an MD-5 hash of the plain-text `/ftrace/lib/Makefile`. Using MD-5 to scramble full pathnames allows us to protect privacy and anonymity on public systems while still studying file clustering and access patterns.

The tracing program can be run any time, and if the output is placed into a directory or file system that is not being studied, the tracing process is invisible to itself. Because this information is collected on every file on the selected file system, a single trace can be quite large. For example, a trace of a file system with 10,000 files using full pathnames and MD-5 scrambling requires approximately one Mbyte of disk space to store the trace data. However, by compressing the data, we can store traces in approximately 25% of the original space until they are needed. The tracing program scans 200-300 files per second, tracing 210,000 files in twelve to fifteen minutes.

We currently run this collection system *nightly* on four active file systems in the Computer Science and Electrical Engineering Department at the University of Maryland, Baltimore County.[2] The four file systems have a capacity of 7.5 GB and are used by the department's faculty and students and include many World-Wide–Web pages. The typical user on the system compiles programs, writes papers, searches the World-Wide–Web, and sends electronic mail. Additionally, most users have their own World-Wide–Web pages, so the system is also accessed by some off-campus users.

These file systems are mounted on a SGI Crimson machine with 208 Mbytes of RAM and approximately 560 users. This central fileserver machine is accessed via the Network File System (NFS) by approximately 150 other workstations or personal computers within the department, as well as many dial-in connections. There are 17 other file systems mounted on this SGI Crimson, with an additional 28 GB of storage. However, we chose the four used in this paper as representative of the others when we began collecting the trace data.

The second part of the system takes two trace files sorted by inode number, compares them, and generates a file containing only the differences, along with new file creations and old file deletions. This program, called `fdiff`, dramatically reduces the tracing system's long-term storage requirements. If only three files out of 10,000 are used in a twenty-four hour period, the difference file only has three lines — not 10,000 — one for each file that has changed. Additionally, a line in the difference file does not necessarily contain the same information as the original trace file — in fact, only when a file is created or has its name changed does `fdiff` keep all the trace information. Table 1 shows the information kept by `fdiff` for the different types of file transactions. By keeping the first day's trace as a baseline along with the difference files, an accurate model of file system activity can be built as all the subsequent transactions are retained.

By way of illustration, we developed a third program, `rebuild`, which takes a trace file and a difference file as input and uses these to "rebuild" the second trace file. For example, if two trace files from period A and period B generate a difference file C, `rebuild` uses trace file A and difference file C to construct trace file B. `Rebuild` serves two purposes. First, it verifies that `fdiff` works correctly and saves all pertinent information. `Rebuild` also reduces the number of trace files that must be kept on the system, thereby reducing the total collection system's storage requirements.

File Activity	**Statistics Kept by `fdiff`**
Access	inode number, access and create times, size
Create	inode number, last access time, inode creation time, last modification time, size, userid, group id, file name
Deletion	inode number, estimated deletion time, size, name
Modification	inode number, last access time, inode creation time, last modification time, new size, difference between old size and new size
Change in Owner	inode number, inode create time, new owner number
Change in Group	inode number, inode create time, new group number
Name Add or Name Delete (file move)	inode number, last access time, inode creation time, last modification time, size, userid, group id, file name, estimated deletion time (if applicable)
Change in inode creation time only	inode number and inode create time. Keeping this information is necessary because some programs, notably `tar`, can post-date files so their "birth date" gets older between traces.

Table 1. File statistics retained by the `fdiff` program.

As mentioned, `fdiff` uses an "estimated deletion time." We estimate the file's deletion time because all we know is that the file was present during the first trace and absent on the second — obviously it was deleted between the two traces. Exactly when the file was deleted is unknown. To make the estimated deletion times more accurate, `fdiff` distinguishes between file deletions in which the inode is reused by another file and deletions where the inode is not reused. If the inode is reused the deletion time is a randomly generated time within sixty minutes prior to the inode being reused by another file [13, 14]. On the other hand, if an inode was not reused, all we know is that the file was deleted between the traces. In this case, a time is randomly chosen within the last 24 hours as the estimated deletion time. As a variation, `fdiff` can also place 90% of the estimated deletions between 9 AM and 4 PM — normal working hours.

The last part of our collection and analysis package is the statistics program. This program programs use the difference files to generate the statistics shown in Table 2. In addition to

File Activity	**Statistics Collected**
Accesses, Creates	Total number of files, total number of bytes, average size, standard deviation (size), maximum size, minimum size. Produces histograms, grouped by file size, of the number of accesses or creations.
Deletions	Same as Access, with deletions where the i-nodes are reused tracked separately from deletions where the i-node is not reused.
Modifications	Same as Access with categories for files that are modified and increased in size, decreased in size, or remained the same.
Modification Differences (Deltas)	Same as Access with categories for files that are modified and increased in size or decreased in size. Tracks files by the amount of change. Produces a two dimensional histogram of file size versus the amount of the modification.
Change of Name, Owner, or Group	Separate categories for change of name (*Unix mv*) change of owner (*chown*), change of group (*chgrp*). Outputs the number of files only.
Long-Term Deletions	Separate data is kept for overall deletions, files that were accessed before deletion, files which were modified before deletions, how many times individual files were accessed or modified before deletion. Produces two dimensional histogram by file size and days the file 'lived' before it was deleted.
Inter-Access and Inter-Modification Period	Summary of accesses and modifications for the last 30 days. Two dimensional histogram by file size and how many days pass between file accesses (or modifications) on individual files.
Files on System at Analysis Completion	Summary of file system status at the end of the trace period. Produces two dimensional histogram by file size and number of times files have been accessed or modified. One dimensional histogram, by file size, of files that have never been used.

Table 2. Statistics collected by the analysis program.

the four programs, we use two Perl scripts in the system. The first runs the file tracing program nightly as a `cron` routine and maintains an activity log; the second automates running the differencing program on trace files.

Our statistics collection and analysis package for file systems has both strengths and weaknesses. Ousterhout [6] noted that 80% of all file creations have a lifetime of less than three minutes. Because the daemons, compilers and other programs that created these files during Ousterhout's work still exist, so do the temporary files they create. Our system traces are collected nightly at 10 PM when few users are active, so we miss most of these temporary files that are created and deleted during the day. However, our collection system is designed to gather information about long-term disk use and file system activity and growth; temporary files that exist for less than three minutes will never be moved to long-term storage and do not contribute to long-term growth.

The differencing program system also misses two other types of transaction. First, while the system detects either a change in a file's name or a modification to a file, if a file has both its name changed *and* is modified between traces, the system counts it as a file deletion and a new file creation. This is because the modified file only retains the same inode, owner, and group; everything else has changed. Our traces show that modifications occur less frequently than many other transactions, and that name changes almost never happen. As a result, we believe the combination of name changes and modifications happens infrequently. Similarly, the system does not notice how many times a file is accessed or modified — only that an access or modification occurred and when the most recent one happened. However, the only way to collect more detailed information is to either run the tracing system more often or to change the operating system kernel and generate a large detailed log file. Both of these options place a heavier load on the computer system and we deliberately decided against doing either.

The collection system is very powerful despite these minor shortcomings. First, the system does not require any operating system kernel modification. As a result, it can be run in user-space, although the tracing program is run as `root` so that all files can be traced. The ability to trace either file systems or directories means that while the entire disk can be traced, it also allows active file systems to be checked more often if necessary. Conversely, file systems or directories that receive little use or that are unimportant can be ignored. The current system allows administrators to answer questions that often provide daily headaches, such as "how many files are really being used?" or "are we using more disk space because of larger files or more files?"

The collection and statistics package allows us to study many long-term disk activities. We can catalog how many files on a system are used, and how often and how much these files are used. When a file is modified the package monitors how much the file changed in size. The package does have some shortcomings. Ignoring the temporary files observed by Ousterhout and the exact number of file accesses or modifications per file make simulations and measurements of short-term usage impossible. However, we feel the lower system overhead and the ability to run the system as a user process without kernel modification outweighs these two points, particularly for studies of long-term file system behavior. Since the file traces we collect will run a mass-storage simulator, we do not care about short-term access patterns; files that exist for seconds or have multiple accesses within minutes have little affect on movement of data between disk and tape. Temporary files will never be migrated to tape, hence we do not need to track of them. Similarly, the number of times a file is accessed per day is less important from a storage migration point-of-view — what matters in this case is whether the file was used at all.

4. Running the Collection System

The collection system has run without difficulty since we began using it in October 1996. We currently trace four file systems with approximately 210,000 files. Tracing the four file systems takes twelve to fifteen minutes, averaging 200-300 files per second; it takes longer if the system has an unusually heavy work-load. The size of a trace file varies with the number of files on the file system. However, all four trace files, with 210,000 file entries, can be compressed to 5 MB of total disk space. These traces are stored on a separate file system that is not traced.

The work of generating a difference file with `fdiff` requires the trace file to be uncompressed, sorted, and compared with another trace file. How long this takes depends on the size of the trace files and the processing machine's speed and memory – it typically takes between 30 and 60 seconds to process one day's trace from each file system – with the vast majority of the time spent uncompressing and sorting the original trace file. Processing four months worth of traces from four file systems takes about 3 hours on our SGI Crimson in a multi-user environment.

Finally, the `stats` program uses only the `fdiff` difference files, which are very small compared to the daily trace files. How long the `stats` program takes to run depends on how active the file systems are, how long a period is being examined, and the statistics being generated.

5. Results

This section has three sub-sections. The first presents some of our findings about the overall system activity. The second sub-section shows daily file system activity during the trace period. Finally, the last sub-section display our results about file activity by file size, and file modifications by file size.

5.1. Overall System Results

Our initial analysis corroborates what earlier studies found, namely that most files are not used very often. Figure 1 clearly shows that on a typical day, only 5,000 files — less than 3% — are used on a system with 210,000 files.

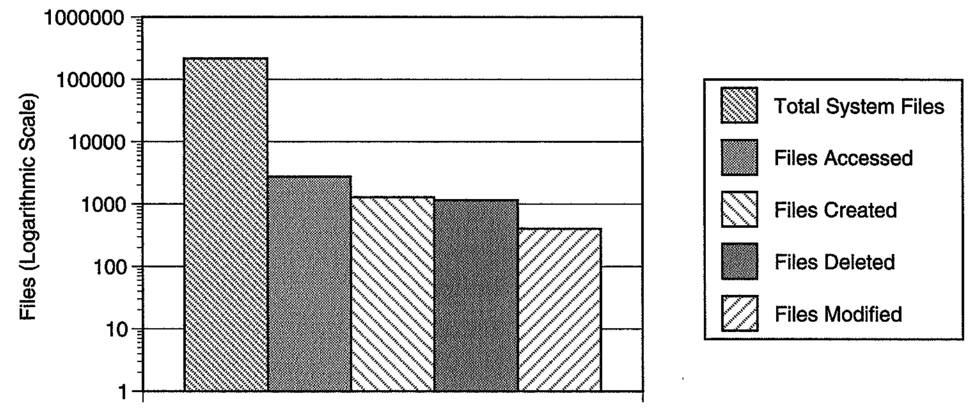


Figure 1. Average daily file usage by category (logarithmic scale).

Figure 2 shows the average daily number of files and bytes acted upon during the trace period. The left side of Figure 2 shows that more than twice as many files are accessed daily than are either created or deleted. It is interesting to note that file creations only slightly outpace file deletions. Finally, only 360 files were modified in any manner on the average day. The right half of Figure 2 shows the number of bytes used daily by the different file operations. Note that the relative ratio between files and bytes holds for file accesses, creations, and deletions. However, while the number of files modified is one-third the number of creations or deletions, the number of bytes modified exceeds the number of bytes for either of the other transactions.

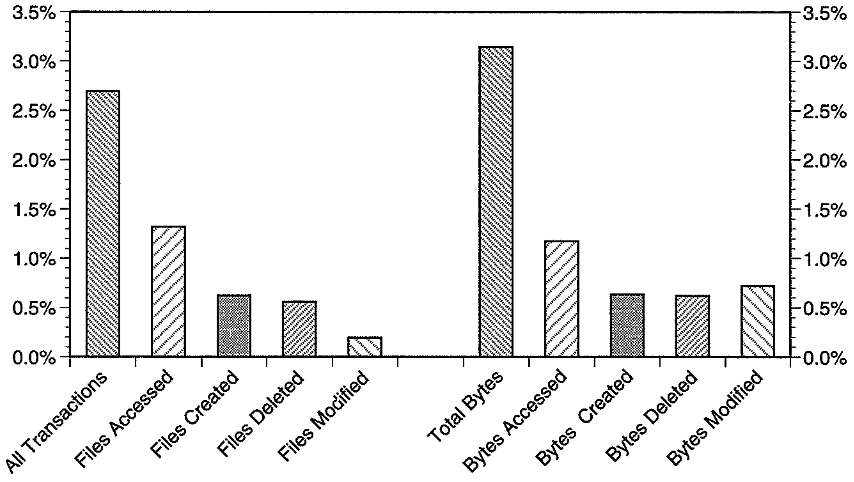


Figure 2. Average daily file system use.
Shown as the percentage of total files or bytes on the system. Note that the relative fractions of files and bytes in each category are comparable except for modifications.

Figure 3 is a breakdown of the average number of files and bytes modified daily – whether the files increased, decreased, or remained the same size is also shown. This chart leads to the conclusion that most modified files either grow or remain the same size, few files get smaller, and those that do decrease in size are fairly small to begin with. You may also conclude that modified files which remain the same size are generally small to begin with. Finally, the majority of the bytes from modifications are in the category of 'modified and increased' in size.

Figure 4 charts the amount of activity on the four different file systems. The file activity is on the left of each file system category and the bytes are on the right. The two most active file systems are `faculty` and `grad3`. An examination of Figure 4 shows that for these two most active file systems, the ratio between file modifications and other actions changes when bytes are used instead of files. This leads to the conclusion that the average files being modified on `grad3` and `faculty` must be larger than on the other two systems. Finally, the `faculty3` file system is not very dynamic; it grew to 99% capacity during the trace period and a study of its daily logs shows a continual drop in all file activ-

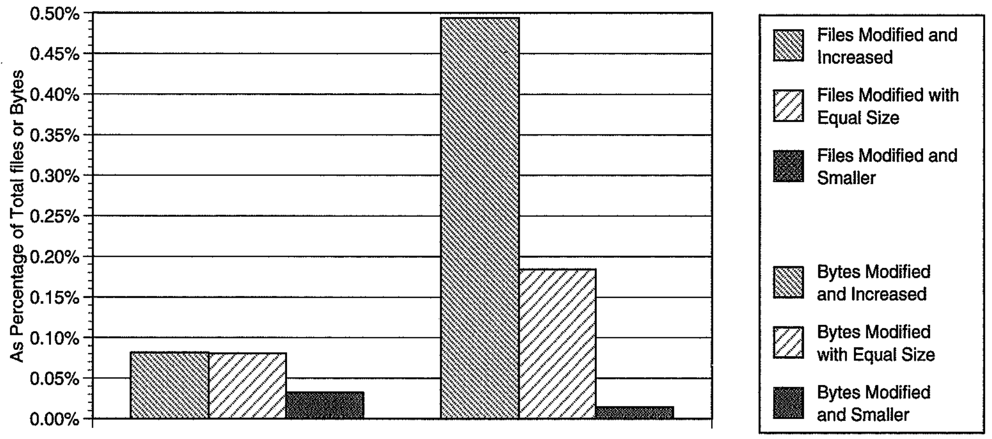


Figure 3. Average daily modifications, grouped by type of modification.
Shown as the percentage of total files or bytes on the system.

ity as it approaches capacity. This activity drop occurs because professors are pack-rats. When a faculty account fills up, the owner simply requests more space on other file systems. On a long-term storage system with automatic migration, these accounts could easily be moved to tape.

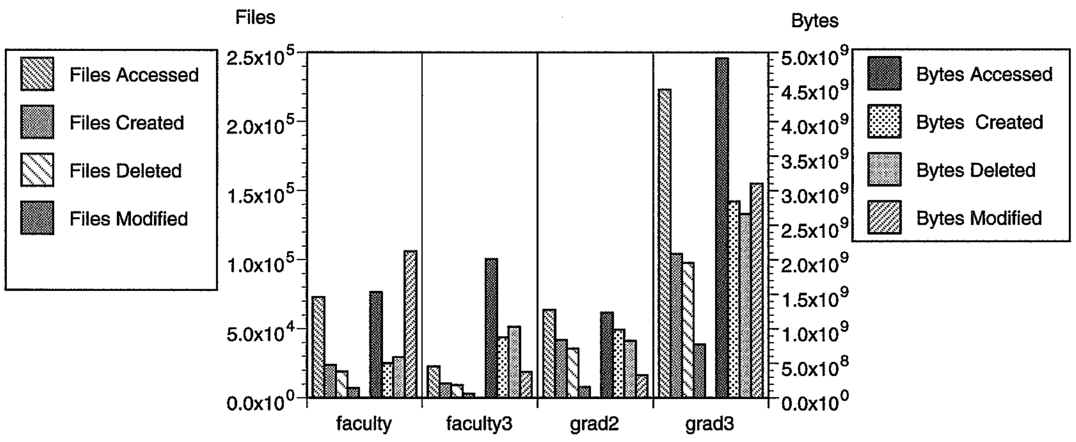


Figure 4. Total activity for the four file systems.

5.2. Daily Statistics

Daily statistics are also available, and some of these are shown in Figures 5 - 7. The time period for these three figures is a 141 day period from 23 October 1996 to 12 March 1997. Every Saturday is shown by a vertical bar in these figures and there is usually a corresponding drop in activity.

A plot of either the number of files or the number of bytes for accesses, creations, deletions, or modifications corroborates many of the conclusions from Figures 2-4. Specifically, that most file transactions are file accesses, that creations slightly outpace deletions, and that file modifications lag behind the other categories in the number of file transactions. This data is not plotted in this paper because the number of data points makes the graph very difficult to read.

By adding together the byte increases from file creations and file modification increases and comparing these numbers with the byte decreases from file deletions and file modification decreases, you can visually see the "byte creep" on the four file systems. Figure 5 shows this long term file system growth. The cumulative number of bytes on the system in Figure 5 appears to start out below zero because approximately 260 MB of data was deleted on the first two days that traces were collected. Nonetheless, file creations overcome this deficit within a month; by the time two months had passed, the system picked up nearly 500 MB from where it was on the second day. A drop occurs in early February, the beginning of a new semester. This February drop is from students and faculty deleting the previous semester's files and doing general 'house-cleaning.'

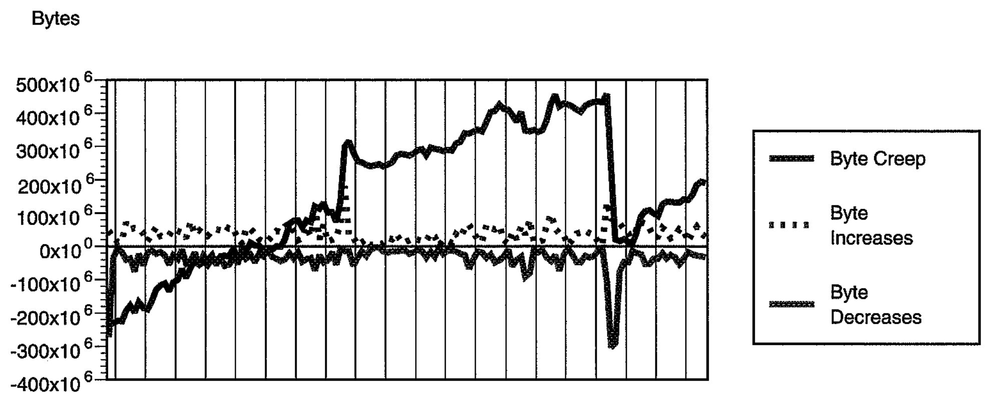


Figure 5. Cumulative daily file system increases.
Plot for a 141 day period from 23 October 1996 to 12 March 1997—vertical bar denotes Saturday.

Although not charted, a comparison of the number of bytes from file creations and deletions with the byte amounts added and removed by file modifications reveals that modifications make very little change in the amount of space on the file systems. Our traces show that modifications usually account for less than 5% of a file system's growth or contraction. The vast majority of change in both the number of files and the number of bytes on the file system comes from file creations and deletions.

Finally, Figure 6 shows the increase in the overall number of bytes on the system. The lines in Figure 6 are fairly flat because the number of bytes added is small compared to the number of bytes already on the system. The large spikes up in week 8 and down in week 18 do appear as small "bumps" in the 'Total' line, because many of the bytes added and deleted during the spikes were larger than normal. A plot of the number of files on the system shows less change because most of the files on the system were never used.

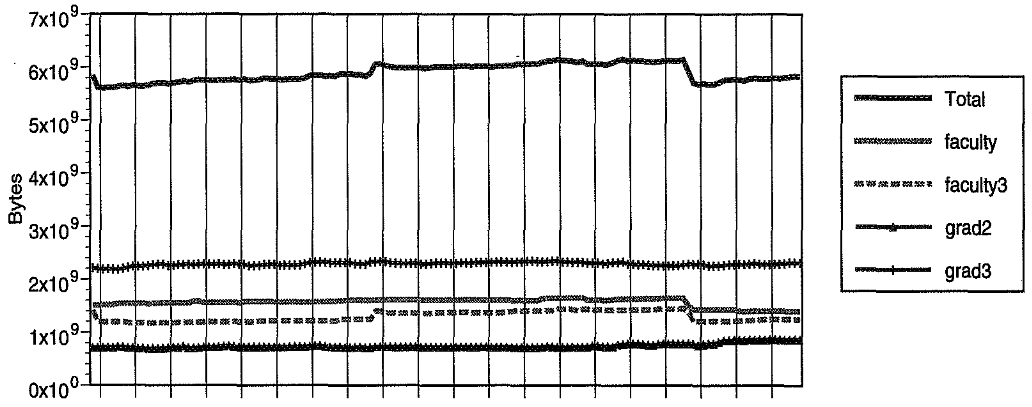


Figure 6. Total bytes on the system during the observation period.
Plot for a 141 day period from 23 October 1996 to 12 March 1997—vertical bar denotes Saturday.

However, plotting the total amount of activity on each file system provides more variance. Figure 7 is a plot of all transactions on the file systems without regard to size. The `grad3` file system is the clearly the most active — proving that graduate students do most of the work at a university. The most active days are in the middle of the week, and weekends show a significant drop in activity.

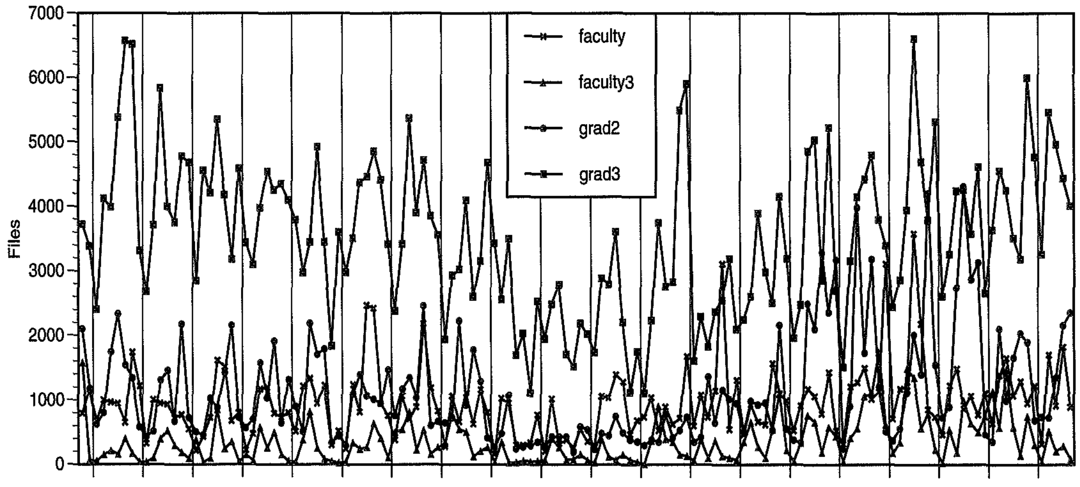


Figure 7. Total system activity — all categories.
Plot for a 141 day period from 23 October 1996 to 12 March 1997—vertical bar denotes Saturday.

5.3. Activity by file size

The statistics package also provides information on the file size the operations take place upon. For example, Figure 8 shows the breakdown by file size of all the files on the four file systems at the beginning of the trace collection process. This figure illustrates that on our file systems, the preponderance of the files are small, generally less than 8 KB, with the "less than 1 KB" category having the largest number of files.

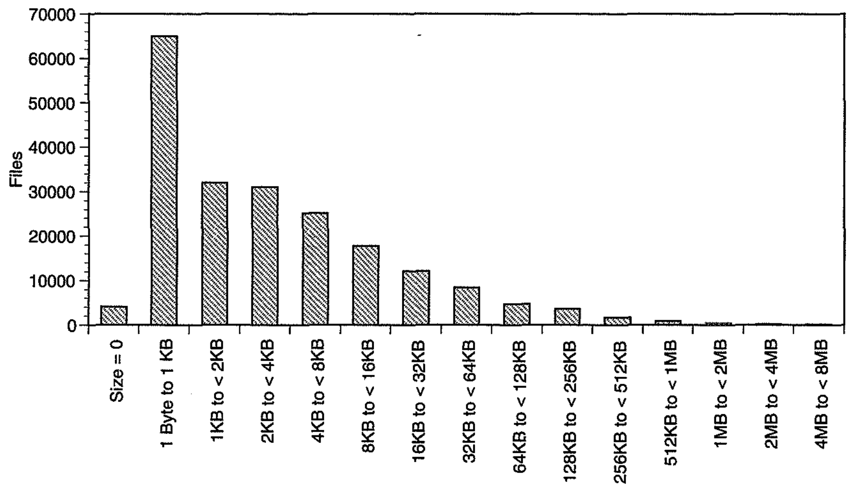


Figure 8. Total number of files on the system by size at the beginning of the trace collection

Figure 9 shows the cumulative percentage of transactions by file size. Again, most transactions occur on fairly small files of less than 8 KB. In most cases, 90% of all transactions are done on files of less than 32 KB. The only transaction category that lags behind the others is modifications, where 25% of the modifications occur on files that are larger than 64 KB.

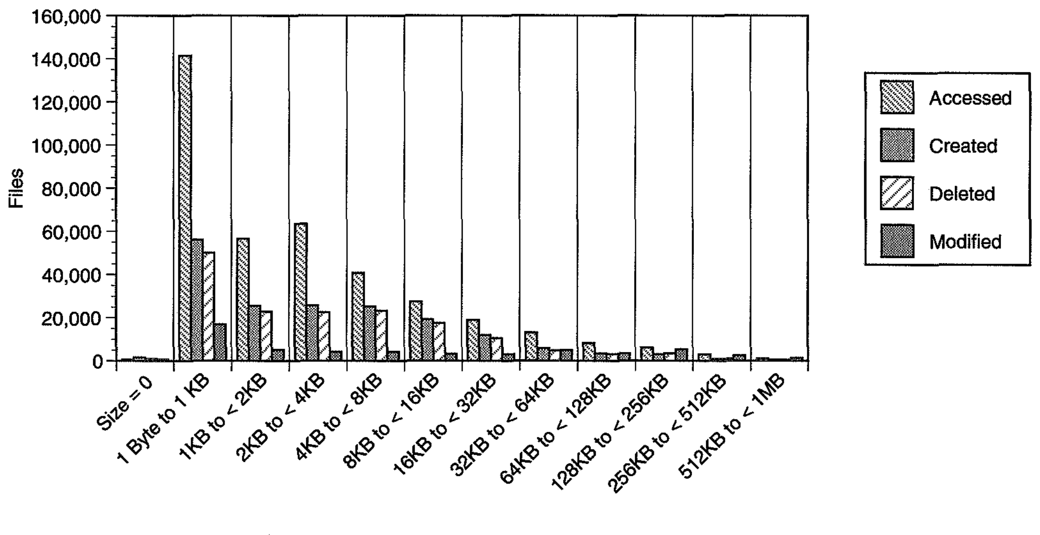


Figure 9. Total Transactions by file size.

Finally, one of the more interesting results comes from comparing the amount a file increases when it is modified with the file's size. Figure 10 shows this information. As expected, most of the files modified are small files. However, the most striking item about this figure is that the amount a file increases as it grows is very small, regardless of the file's actual size. Most files increase in size by no more than 1 KB; in fact, file increases of more than 8 KB are rare, regardless of the file's overall size. A similar chart can be made from plotting the file decreases with the file's size. The only difference is that the majority of the files that are modified and decrease in size is even smaller than those that increase.

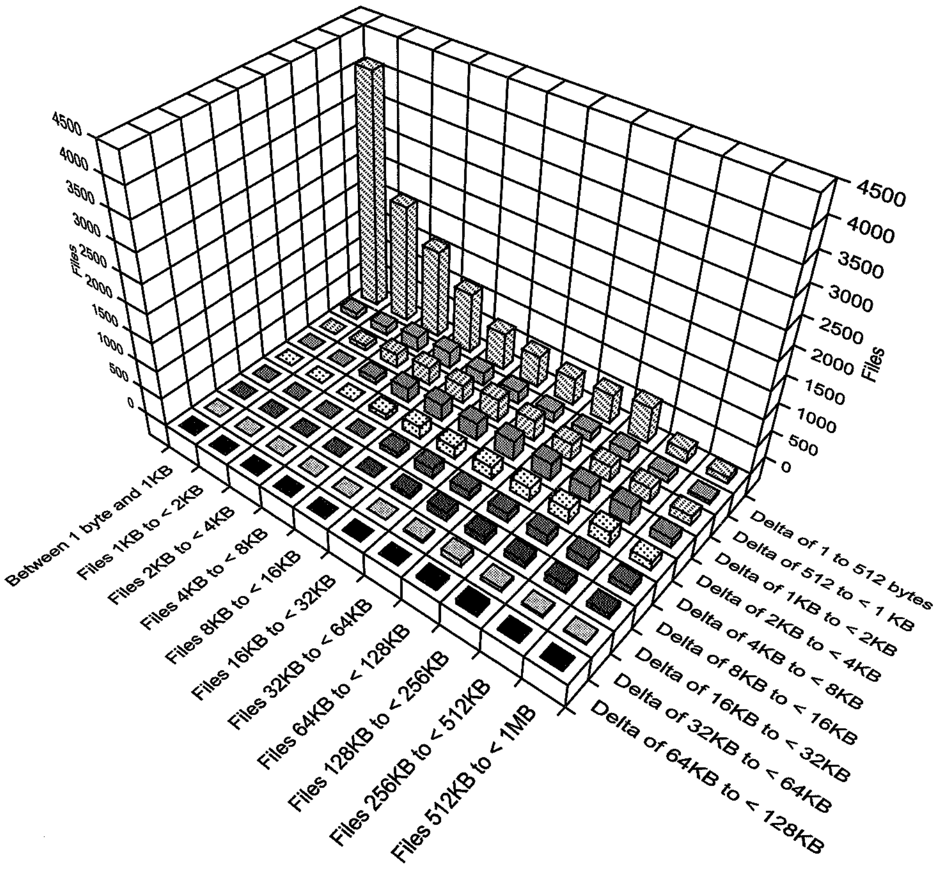


Figure 10. Number of file modification increases cross-referenced with the modified file's size

By way of illustration, Figure 11 is a graph of the file modification increases for large files only, shown as a cumulative percentage. There are six file size categories between 128 KB and 8 MB. Regardless of the category, 90% to 95% of all file modifications add less than 64 KB to a file, 80% add less than 32 KB.

6. Observations

Our initial work allows us to make some observations about file system growth and activity. Some of these observations corroborate earlier research, while others are new. First, file accesses occur more often than any other type of file transaction. An obvious implication is file accesses should work efficiently. For a mass storage system with nearline tape storage, this means that files which are likely to be accessed should be on the disk system,

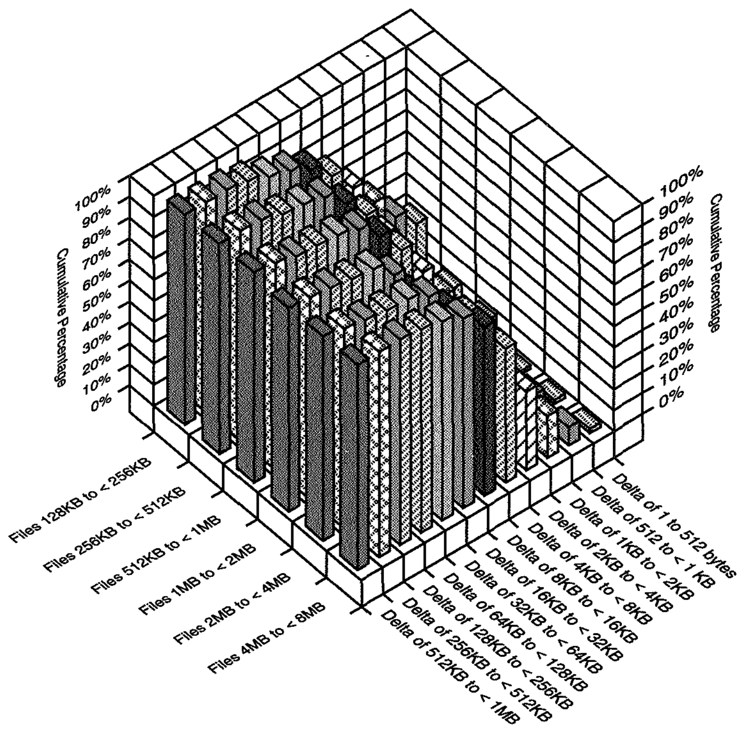


Figure 11. File modification increases for large files (shown as cumulative percentages).

and if a file is on the nearline tape it should still be quickly accessible. Another observation in keeping with previous work is that most user files on a system are small, generally less than 4 KB.

Next, a great number of files are created and deleted every day. At least some of these creations and deletions come from the fact that many text and image editors make modifications by erasing the old file and creating a new one with the same name. In essence, compilers do the same thing. However, this should not be counted as a file modification. From the file system's viewpoint, it truly is a file creation and a file deletion, and not a file modification. At first glance, it may seem wiser to treat these transactions as a file modifications from a long-term storage point-of-view. Unfortunately, the long-term storage tape may not have room to let a file grow by 4 KB.

A more telling observation is that comparatively few files are modified, and if they are modified, the amount added to the file is very small — regardless of the file's overall size. This is a new concept for file system designers. It is generally assumed that if a large file needs to grow, the amount of new disk space allocated for growth should be proportionate to the original file's size. However, this does not appear to be true. Our initial work shows that most modifications result in a very small file increase, regardless of the file's original size.

Given these observations, it is reasonable to assume that a file is more likely to be deleted than modified; and if it is modified, a file will only increase by a small amount. Most importantly, means that file accesses should be the most efficient operations. The efficiency for file creations and file deletions follow behind file accesses. Finally, file modifications do not have to be extremely fast or efficient operations because they rarely occur.

In keeping with the observation that file modifications do not need to be extremely efficient, a file system that allocates more than one disk block at a time to a growing file only makes work for itself. For example, assume the following: A file system with 512 byte blocks has a file larger than 256 KB that needs to increase in size. This type of modification happens fairy rarely; in our traces of 838,000 total file transactions there are only 23,619 modifications resulting in a file increase. Of these, only 3,278 – 0.39% of all transactions and 13.87% of all modifications with increases – fall into the category of a file larger than 256 KB needing to grow. Of these 3,278 modifications, 92.3% increase by less than 32 KB. In fact, more than half of the files that are larger than 256 KB increase by less than 4 KB, or just eight 512 byte blocks. in order to meet the file's size increase, the file system will need usually only need to allocate eight 512 byte blocks. If a file system automatically allocates more than eight blocks based on the file's original size, these excess blocks have to be reclaimed later by the operating system. Automatically allocating disk blocks or file extents based on file size becomes even more problematic with larger files — 85% of all files larger than 1 MB increase by 32 KB or less.

Files that grow in size happen rarely enough that allocating blocks as needed will not usually make much difference to users. However, it makes file system design easier because it reduces fragmentation and 'garbage collection.'

7. Future Work

We are continuing to work on the file tracing system and are expanding both the number and types of file systems we collect trace data upon. An important note for system administrators is that a computer system does not need to run `ftrace` to use either the differencing or the statistics programs. Data can come from other tracing programs or from system logs, as long as the data is in the correct format. We are actively working with a super-computing project that keeps detailed logs of the file accesses, there we will only need to modify the format of the log file.

The next step of the project is to develop a mass storage system simulator which we will then use to model different file migration strategies. The first part of the simulator will generate better statistical information than we currently have. For example, we want to be able to accurately display inter-reference periods and the number of times and ways a file is used.

8. Conclusions

We have developed a comprehensive set of tools that allow system administrators to monitor long-term file system activity and growth. We do this by collecting periodic traces, as unobtrusively as possible. The traces can then be analyzed for different types of activity. The file differencing program dramatically reduces the amount of work the analysis programs must do. While the differencing program must sort every trace file, the resulting difference file is *much* smaller than the parent trace file. This is particularly important when long-term statistics are being stored.

In this paper, we have shown that most long-term file system activity is caused by file accesses, with file creations and file deletions lagging behind. File modifications account for little of the file system's activity.

Finally, we have illustrated long-term growth on the file system, and accounted for the source of this growth. File systems grow in size because of file creations, not file modifications. File modifications do not normally increase the size of the file by more than 32 KB. The fact that file modifications do not increase (or decrease) a file's size by a large amount may allow file system designers to change the way file blocks are allocated.

[1] A computing center has 30 or more workstations, with 30 GB or more of disk storage. Obviously, the larger the computing center, the more cost effective storing seldom used files on tape becomes. Similarly, what people using the computing center do also influences this decision. For example, a business could easily move old financial accounts to tape, knowing they were accessible within a few minutes.

[2] This was true when this paper written (May 1997). As of October, 1997, we collect data from the UMBC Computer Science Department (8 file systems with 28 GB of storage), the UMBC Computing Services cluster (9 file systems with 35 GB), a DoD installation (8 file systems with 4 GB), and a supercomputing complex more than fifty terabytes.

References

[1] This work was supported by the NASA Ames Research Center under grant NAG 2-1094.

[2] David A. Patterson and John L. Hennessy, *Computer Architecture: A Quantitative Approach* (Morgan-Kaufman Publishers, San Francisco, CA, 1996)

[3] R. Hugo Patterson, et al, "Informed Prefetching and Caching," Operating System Review 29(5), *Proceedings of the 15th ACM Symposium on Operating Systems Principles* (1995) pp. 79-95.

[4] Robert L. Henderson and Alan Poston, "MSS-II and RASH: a mainframe UNIX-based mass storage system with a rapid access storage hierarchy file management system." *USENIX Winter 1989 Conference* (San Diego, CA, January 1989) pp. 65-84.

[5] Alan Jay Smith, "Analysis of long term file reference patterns for application to file migration algorithms." *IEEE Transactions on Software Engineering* SE-7(4),

(1981) pp. 403-417.

[6] John K. Ousterhout, Herve Da Costa, David Harrison, John Kunze, Mike Kupfer, and James Thompson, "A Trace-Driven Analysis of the UNIX 4.2 BSD File System." Operating System Review **19**(5), *Proceedings of the 10th ACM Symposium on Operating Systems Principles* (1985) pp. 15-24.

[7] Mary G. Baker, John H. Hartmon, Michael Kupfer, Ken W, Shirriff, and John K. Ousterhout, "Measurement of a Distributed File System," Operating System Review **25**(5), *Proceedings of the 13th ACM Symposium on Operating Systems Principles* (1991) pp. 198-212.

[8] David W. Jensen and Daniel A. Reed, "File Archive Activity in a Supercomputer Environment." *Technical Report UIUCDCS-R-91-1672*, Department of Computer Science, University of Illinois, Urbana, IL (1991)

[9] John Merrill and Eric Thanhardt, "Early Experience wit Mass Storage on a UNIX-Based Supercomputer," *Tenth IEEE Symposium on Mass Storage Systems* (Monterey, CA 1990) pp. 117-121.

[10] Ethan L. Miller and Randy H. Katz, "An Analysis of File Migration in a UNIX Supercomputing Environment," *USENIX Winter 1993 Conference* (San Diego, CA, January 1993) pp. 421-434.

[11] Ethan L. Miller and Randy H. Katz, "Analyzing the I/O Behavior of Supercomputer Applications," *Eleventh IEEE Symposium on Mass Storage Systems* (Monterey, CA 1991) pp. 51-55.

[12] Stephen Strange, "Analysis of Long-Term unix File Access Patterns for Application to Automatic File Migration Strategies," *Technical Report UCB/CSD-92-700*, Computer Science Division (EECS), University of California, Berkeley, California (1992).

[13] Maurice J. Bach, *The Design of the Unix Operating System*, Prentice Hall, Englewood Cliffs, NJ 1990.

[14] Samuel J. Leffler, et al, *The Design and Implementation of the 4.3BSD UNIX Operating System* (Addison Wesley, Reading Massachusetts, 1990).

Page intentionally left blank

Performance Tuning of a High Capacity/High Performance Archive for the Earth Observing Systems Project

Alla Lake
Lockheed Martin Space Mission Systems and Services
1616 McCormick Drive
Upper Marlboro, MD 20774
alake@eos.hitc.com
tel: +1-301-925-0626
fax: +1-301-925-0651

Abstract: The Archive for the Earth Observing Systems (EOS) Project reflects the scale of the project itself. Its data holdings over all the Distributed Active Archive Centers (DAACs) are projected to exceed two Petabytes by the year 2002. This paper describes the experience and results of the integration and tuning process for the hardware and Commercial Off The Shelf (COTS) software used today in the EOS Core System (ECS)[1] archive. Sizable performance improvements have been realized to date through the combined efforts of the vendors and the tuning team. The process is still continuing and, indeed, will continue through the life of the archive. The effort to date centered around improving the throughput for the individual data streams to the archive tape drives and the cumulative throughput to all the drives. The work was specifically directed towards the COTS hardware and software integration and tuning adjustments. Throughput performance improvement via efficient data organization on tape was not addressed during this effort, but will be in the future work.

1 Introduction

ECS archive hardware is currently installed at five distributed sites, known as Distributed Active Archive Centers, or DAACs: Goddard Space Flight Center (GSFC) in Greenbelt, Maryland, Langley Research Center (LaRC) in Hampton, Virginia, Earth Resources Observation System (EROS) Data Center (EDC) in Sioux Falls, South Dakota, Jet Propulsion Laboratory (JPL) in Pasadena, California, and the National Snow and Ice Data Center (NSIDC) in Boulder, Colorado.

Among these sites, GSFC is the largest in both the accumulation of data and in the daily data throughput. The data holdings at GSFC alone will amount to more than one Petabyte in 2002. The data rates through the DAAC archives will match the storage capacity scale of the archives at the corresponding sites. For example, per current baseline, at its peak data ingestion period beginning in the year 2000, GSFC DAAC is to ingest close to a Terabyte of data per twenty four-hour period. This includes all levels of products, from L0 through the progression of higher level products for permanent storage and the data produced by reprocessing. To satisfy output requirements, the system is designed to distribute data to users at approximately twice the ingestion rate.

Not surprisingly, the capability of the archive to absorb and to serve data at these rates depends on a well-integrated, well-tuned combination of high performance hardware and software. The task of building an archive for ECS is complicated by the amount of data accumulating in permanent storage. In order to accommodate the expected volume of data, the storage must provide high capacity, in addition to high throughput - a difficult combination. Another complicating factor is the necessity of building the archive with components that were commercially available at the time of selection and procurement, i.e. 1995, 1996.

All the ECS DAACs have the same architecture and constituent components. The DAACs differ only in the size and particulars of equipment. Therefore, properly integrating and tuning the largest site, GSFC, creates a configuration template for most of the remaining DAACs.

Figure 1, ECS Architecture, illustrates the relative position of the archive within ECS configuration and the data flows within a DAAC.

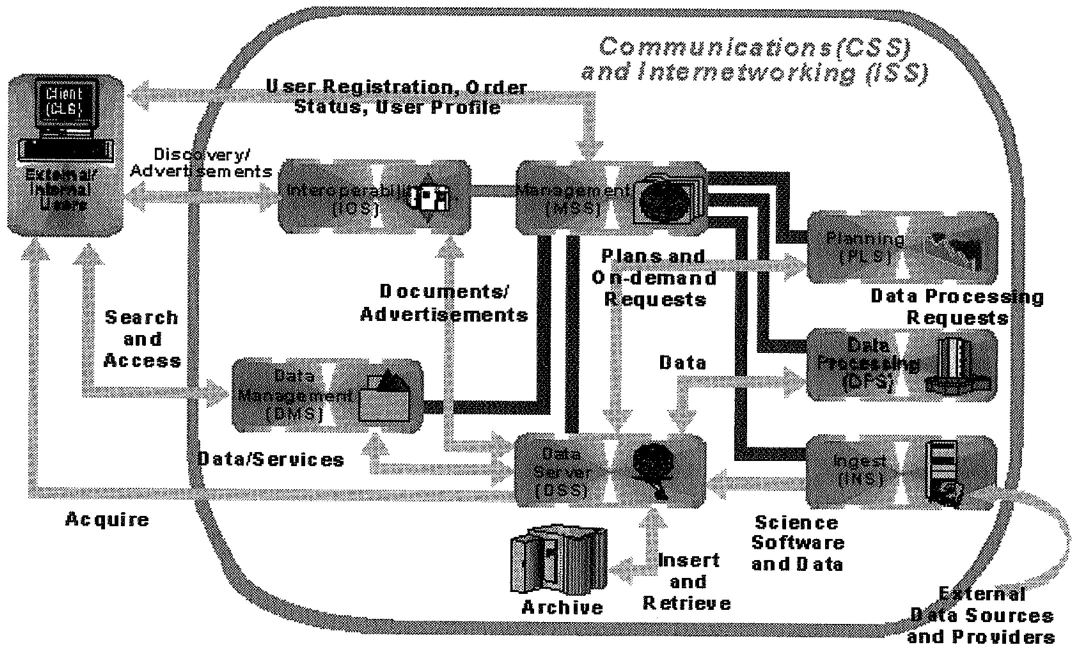


Figure 1. ECS Architecture

This paper describes integration and tuning efforts for the archive repository component of the GSFC DAAC, labeled **Archive** in Figure 1.

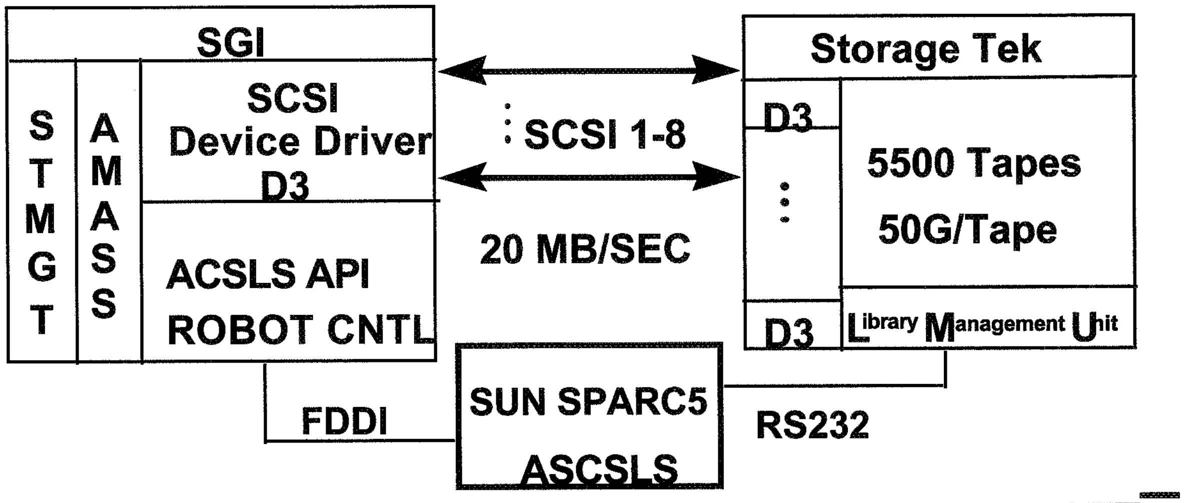
The integration and performance tuning that were undertaken could be described as a series of movements of the data from one component of the archive to another: from the data rate achieved on an individual tape drive, to the exchange rate of the robotic mechanism, to the overall data rate to and from the entire silo. The shifts were largely realized by improvements in the software design and function and by gaining additional performance from the RAID. Some of the desired enhancements are still pending at the time of this writing in November of 1997.

2 Architecture and Equipment Suite at the GSFC DAAC Archive Repository

The integration and tuning work described below pertains to the following suite of commercial products. AMASS[2] File Storage Management System (FSMS) software, from EMASS Corporation, controls the physical storage of the data collection and is hosted on a multiprocessor Silicon Graphics Challenge server. The data repository resides in STK PowderHorn robotic silos and is recorded using RedWood (SD-3) helical scan tape drives from STK Corporation. Redundant Array of Inexpensive Disks (RAID) from Silicon Graphics Corporation, configured as RAID-3, is used for the temporary caching of data en route to and from the robotic silos. The initial (at-launch) GSFC archive configuration consists of two STK PowderHorn robotic silos. Each silo is equipped with eight SD-3 helical scan tape drives.

The tape drives residing in the STK robotic silo are directly connected to the SGI host via Fast-And-Wide SCSI II channels, one channel per drive. Each channel is individually capable of the throughput of 20 MB/sec. Each of the eight tape drives is rated by the manufacturer as capable of 11.2 MB/sec sustained throughput. The drives exhibited even higher streaming data throughput rates (up to 16 MB/sec) if hardware data compression was enabled during recording. The compression feature was enabled during testing, but the degree of data compression realized in each case depended on the specific data used and, in turn, determined the data rate above the manufacturer's rating.

Two SGI multiprocessor Challenge servers, each configured with six CPUs and 512 MB of memory and each running a copy of AMASS, have one PowderHorn silo allocated to each of them. 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 SUN SPARC5 workstation. AMASS addresses the ACSLS through a network connection. The ACSLS then controls the robot directly via an RS232 line.



Note: STMGT - ECS Storage Management Code Performing the Archive Control

Figure 2. Archive Hardware and Software Configuration Under Test

All the work described here is done on one of the two silo/host/RAID suites. The final configuration is to be applied to the second suite before proceeding to system integration and test. Therefore, for the remainder of this paper, a single silo, host, RAID unit, etc. will be referred to as the *target configuration.* The target equipment and software configuration is illustrated in Figure 2, GSFC DAAC Archive Repository Equipment Suite.

To date, all of the tuning and configuration adjustments for improving the performance of the archive component were made on either the AMASS software itself, or on the SGI RAID used as a temporary cache, for the data passing to and from the archive silo. Therefore, *AMASS Tuning* is often used as a synonym for the tuning of the overall archive component. STK hardware required no adjustments beyond maintenance.

3 Performance Tuning Methodology and Tools

The stand-alone performance of the archive was assessed by measuring the data throughput to and from the tape drives in the robotic repository. At the time of writing of this paper (November 1997), the data was transferred locally. The only network transfers were of a very small volume of robotic control signals and physical inventory synchronization data between the AMASS software and the STK ACSLS robotic control software.

Figure 3, Data Flow to Tape, shows the test data flow initiated from a directory on a file system disk and directed into the archive. Data Flow from Tape is in the opposite direction. AMASS cache area is a portion of the disk dedicated exclusively for use by AMASS for the maintenance of open archive files and file staging for writes and reads to the archive [1].

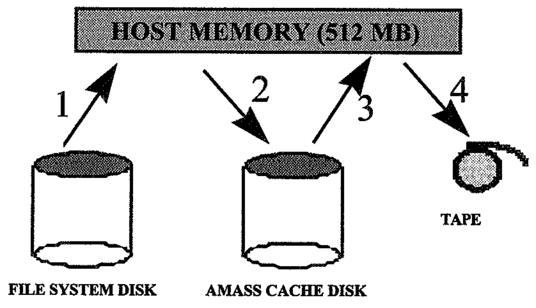


Figure 3. Data Flow To Tape

The most useful tool in assessing throughput performance was the *sysperf* tool supplied by EMASS as part of the software bundle. The tool produces the display shown in Figure 4.

Visibility into AMASS cache area is beneficial for the performance improvement exercises, as well as for problem resolution. Section 5, Software adjustments, describes how cache configuration can be tuned.

```
SYSTEM STATISTICS  -  Thu Sep 11 09:48:21
UPDATE INTERVAL    -  5 SEC
AVERAGE THROUGHPUT - 44595 KBYTES/SEC

READ REQUESTS        # OF VOLUMES
0                         0

WRITE REQUESTS       # OF VOL GROUPS
6                         6

CACHE BLOCKS    240 Total    216 Free    24 Dirty
FNODES                50 Total    43 Free    7 Used

JUKE  DRIVE  VOLFLAGS  VOLUME  VOLGRP  KBYTES/SEC
1     1         A         72      21          0
1     2         A         86      26       9420
1     3         A         78      23       9420
1     4         A         74      22       8345
1     5         A         80      24       8601
1     6         A         89      27       8806
1     7         A         92      28          0
1     8         A         83      25          0
```

Figure 4. ***sysperf*** **Tool Display**

As can be seen from Figure 4, both the individual throughput of a tape drive (under the heading *KBYTES/SEC*) and the overall throughput of the server to the silo can be assessed for the sampling interval. The measurements are taken by EMASS software and reflect its use and freeing of its raw cache segments, rather than actual measurements at the tape drive. The sampling interval can be adjusted in increments of a second from 5 to 60 seconds.

Another very valuable tool was the SGI *Performance Copilot* (*pcp*). The tool afforded an understanding of the RAID disk functioning and facilitated proper reconfiguration of the RAID to maximize throughput.

Finally, UNIX *timex* command was used extensively to gauge the efficiency of a write to or read from AMASS. The data produced by *timex* for a single write to the archive indicated the rate at which user data was being transferred out of the file system disk or into it. *sysperf* timed the outgoing write data streams from AMASS cache on RAID to the tape drives. When multiple background requests were initiated, *timex* could be relied upon to measure the average performance of the transfers to disk and to tape, rather than to memory. That was due to the cumulative size of the transfer exceeding the 512 MB of system memory.

In order to have a measure of control over the read and write data streams, volume *groups* (shown in Figure 4 as *VOLGRP*) were used extensively. A small number of tapes and a separate UNIX directory were assigned to each of the volume groups. This kind of partitioning of the storage allowed data stream direction to one or more tapes, as desired. It was useful for stress testing both the hardware and the FSMS. Simple *Perl 5* scripts with loops issuing sequential *dd* commands were used for initiating the data transfers to and from the archive.

The following sections summarize the performance improvements and give a high level description of the configuration changes that produced them.

4 Initial and Current Performance Comparison

Table 1, Individual Tape Drive Throughput Improvements, and Table 2, Cumulative Archive Throughput Improvements, serve to illustrate data rate performance changes as the result of the integration/tuning activity.

Individual Rate	January 1997 *	August 1997 *
Peak Write (MB/sec)	~ 2	16
Peak Read (MB/sec)	~ 2	16

*Note: Compression is enabled.

Table 1. Individual Tape Drive Throughput Improvements

As mentioned previously, data compression was enabled on the tape drives. The peak performance figure in Table 3 is a reflection of data compression.

Cumulative Rate	January 1997 *	August 1997 *
Peak Write (MB/sec)	7	47
Peak Read (MB/sec)	9.5	29

*Note: Compression is enabled.

Table 2. Cumulative Archive Throughput Improvements

The rise in the peak performance rates is dramatic. It must be noted, that for larger files or for groups of files read from or written to the same tape volume such individual peak streaming performance may be sustained for longer periods of time than for shorter files that are found on a tape or directed to a tape one at a time. Again, as can be expected, individual throughput drops to the cumulative performance limit for multiple concurrent streams. Cumulative performance is very much a function of total concurrent accesses of the AMASS cache disk partition. The peak cumulative throughput to and from the tape drives is measured at times when no *write* transfers from the system disk to the AMASS cache are taking place. Depending on the overall load and where the individual drives are in the load and search cycle, the rates at any instant of time may vary anywhere between zero and the peak. The average, while a function of obtainable peak rates, will also depend on the system use profile.

5 Summary of the Tuning and Configuration Adjustments Efforts

5.1 RAID Adjustments

Summarized below are adjustments that led to the improved peak rates. The greatest measure of performance improvement, due to the enlarging and tuning of the RAID configuration, was realized in the cumulative performance of the system. It must be noted that during the design phase, a RAID configuration rather than *UNIX striped* disk was selected to maximize the reliability of the data. With the projected data rates, and given such large numbers of disk, that would raise the failure incidence, and every failure of non-RAID disk would impose further load on the system. Disk striping was essential in order to meet the data rates. The top throughput rate of the disk configuration is one of the two factors that determine the ceiling on the peak cumulative throughput of the stand-alone archive configuration.

Aside from the sheer data flow rates, another factor affecting throughput performance is the degree of utilization of the tape drives in the robotic silo for reading and writing of the data, as opposed to the drives being idle during tape fetches or spending time for tape mounts, dismounts, rewinds and positioning to the beginning of data. This factor is, to a large extent, an immutable function of the selected tape drive and robotic equipment. An improvement may be realized by trying to maximize data co-location on the tapes or, conversely, data dispersal in the archive. This portion of the work, involving analysis of the processing profiles, is not in the scope of the current paper. It is the former factor, the throughput of the RAID that was targeted by the tuning process.

Figure 5, Expected Data Flow through the FSMS RAID During Operation, is a diagram of the currently anticipated archive data flows on RAID to and from the tape drives once the system comes on-line. As can be seen in the diagram, due to the combination of the AMASS buffering in the *AMASS Cache* and ECS system design requirement to place data in the area labeled as *Staging*, most data is *double hopped* on the RAID. The result is, in the worst case, two writes to and two reads from the same RAID configuration for each archive access.

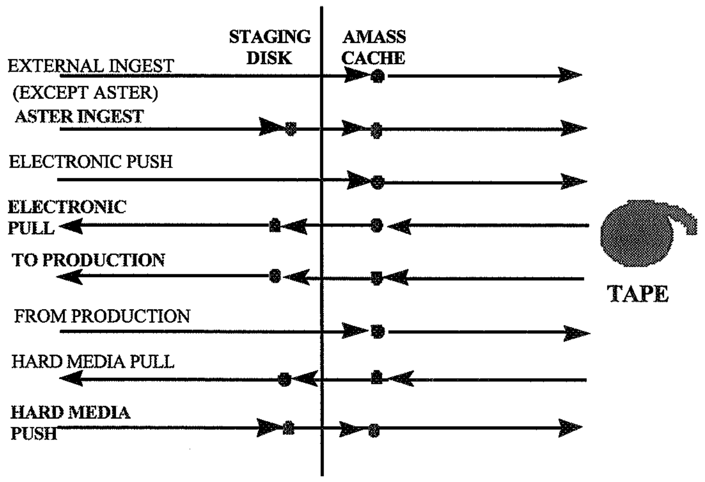
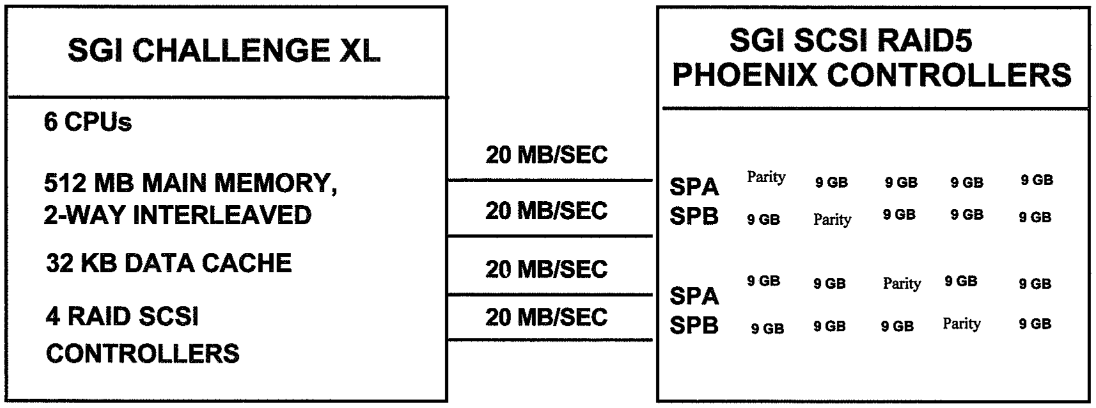


Figure 5. Expected Data Flow through the FSMS RAID during Operation

The change in the RAID configuration level from 5 to 3 accounted for the initial improvement in the individual throughput to and from tape. Most of the individual tape drive performance improvement was due to changes in the AMASS software that allowed for a configurable blocking factor on the tape.

Figure 6, Hardware Configuration under Test as of January 1997, illustrates the hardware configuration that produced the initial throughput rates listed in the January *1997* columns of Tables 1 and 2. As shown, the RAID level configured initially was level 5. Command Tag Queuing (CTQ) was not enabled. Command Tag Queuing, when enabled, provides for request queuing to the disks and request interleaving. The drives were formatted with the default 4 KB block size.

The available RAID disk was divided into 6 GB of raw disk for AMASS Cache with the remainder for the user file system. The stripe element used was 256K blocks. Each block is 512 Bytes. During a test of a write to tape a data file was copied from a directory in the *Staging* user file system partition to the raw *AMASS Cache* partition and subsequently to tape. In the read from tape test the direction of data is reversed (in the direction opposite to the one in Figure 3).



SGI CHALLENGE XL
6 CPUs
512 MB MAIN MEMORY, 2-WAY INTERLEAVED
32 KB DATA CACHE
4 RAID SCSI CONTROLLERS
20 MB/SEC
20 MB/SEC
20 MB/SEC
20 MB/SEC
SGI SCSI RAID5 PHOENIX CONTROLLERS
SPA Parity 9 GB 9 GB 9 GB 9 GB
SPB 9 GB Parity 9 GB 9 GB 9 GB
SPA 9 GB 9 GB Parity 9 GB 9 GB
SPB 9 GB 9 GB 9 GB Parity 9 GB

Figure 6. Archive Hardware Configuration Under Test as of January, 1997

An enlarged RAID configuration currently in use is shown in Figure 7, Archive Hardware Configuration Under Test as of August, 1997. The additional four controllers serve to expand the bandwidth as well as to reduce contention due to multiple disk hits. The same is true for the additional disk capacity.

Aside from physically enlarging the RAID configuration and adding four more SCSI RAID controllers, other RAID configuration parameters were adjusted as follows. 1) Command Tag Queuing (CTQ) was enabled on the RAID with the CTQ depth set to 24; 2) The stripe element for both the raw and the file system portions of the disk was set at 1024 K blocks, thus enlarging the amount of data transferred per disk access; 3) Block size of 64 KB was used to format the drives with *mkfs_xfs*; 4) The raw and the file system portions, were each sized 128 GB and allocated to a set of four separate RAID controllers to minimize disk hit contention; and 5) in sequencing the SCSI address allocation to the A and B controllers or Storage Processors (SPs) during disk partitioning,, the entire A side was listed first, then the entire B side, in order to minimize the detrimental effects of A/B interaction. Thus, when the A side is under load, the B side experiences the resonance effects but is not under load at that time and vice versa.

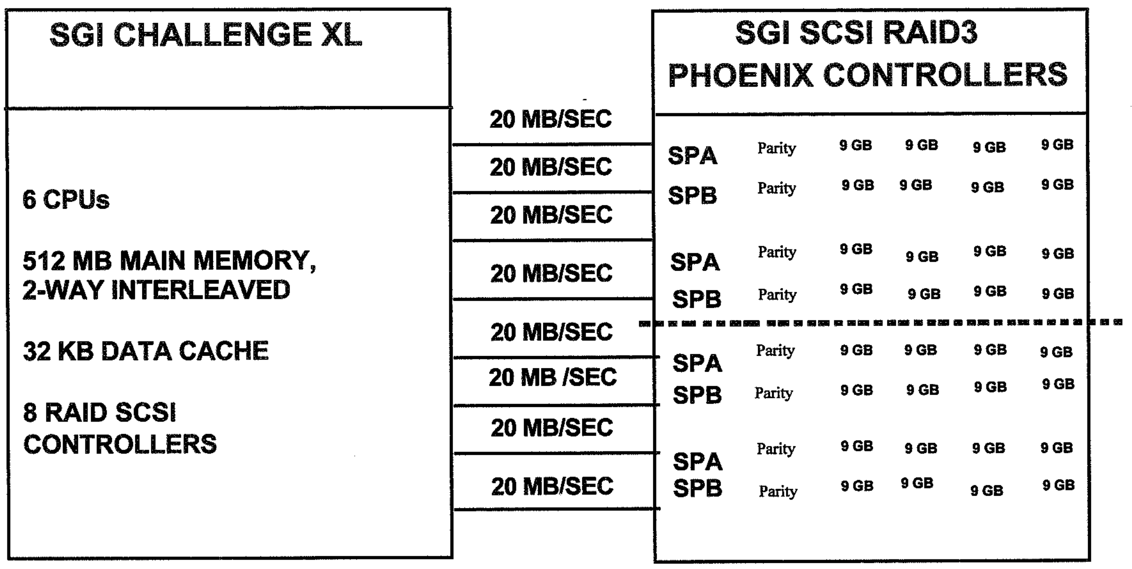


Figure 7. Archive Hardware Configuration Under Test as of August, 1997

Table 3, RAID3 Benchmarks, is a summary of *timex* measurements of reads and writes against RAID3 configuration before and after the last tuning adjustments as of November, 1997. Request size of 1024 KB was used.

	April 1997	October 1997
Best Write (MB/sec)	16	50 (file system); 35 (raw)
Best Read (MB/sec)	30	75 (file system); 80 (raw)

Table 3. RAID Benchmarks

5.2 Software Adjustments

5.2.1 Understanding the correct use of AMASS

Some of the initial throughput problems were the result of the lack of user sophistication in the use and configuration of the product. The standard product documentation requires extensive study and discussions with vendor for configuration. The manuals are geared more towards low volume - performance - indifferent sites, versus a facility like an ECS DAAC. However, the advanced user training offered by the vendor was most helpful and technical support of the configuration changes informative. The following adjustments were made in the ECS use of AMASS.

1) Use of *dd* data copy. Surprisingly low initial throughput results can be partially attributed to the use of the UNIX copy command, *cp,* for data transfers to and from AMASS directories. The block size used by the *cp* command for data transfers was initially 4 KB. Such transfers were, understandably, very slow. The *dd* command with a block size of 1024 KB is now used for all AMASS data transfer tests and tuning.

2) Appropriate tuning of AMASS cache. AMASS cache parameters were tuned to the prevalent file size in order to optimize recording of that file to tape. As an example, in order to optimize the transfer of a 1 GB file to tape, each of the four segments comprising the total portion of data going to tape at one time was made to be 256 KB. Such tuning for larger files would result in excessive disk allocation if smaller files prevailed in the system.

This adjustment was required due to the effect that the tape drive buffer flush delay had on the total write to tape throughput. Disk transfer would block for the particular process while its tape drive buffer was being flushed. Since a single buffer flush, lasting for up to 20 seconds, occurred after each four *dirty cache blocks* written to tape, as well as at the end of a file, it was important to minimize the number of such flushes.

Cache size tuning to improve write performance for large files can, however, limit the number of simultaneous processes which can use that cache (the NFNODE parameter, that the vendor defines, as the number of files that could be open in the AMASS file system at the same time [1], is inversely related to the cache block size). Fortunately, in the near future, the vendor is discontinuing the synchronous buffer flush to tape and is going to an asynchronous buffer flush. This will not only greatly improve the performance of an individual write to tape, but also make the fine adjustment to the prevalent file size unnecessary.

5.2.2 Performance Related AMASS corrections by the Vendor

In the interval from the time when the testing of AMASS on the target configuration commenced, at the start of 1997, and the time of this writing, November of 1997, several product deficiencies with respect to data throughput were corrected. Each major correction, as well as resolution of other non-design software problems as encountered, was accompanied by extensive testing on the ECS side, stretching to a year of near-full time involvement by the author and other team members as necessary. The three significant performance related corrections are detailed below:

1) Adjustable size blocking factor to tape. Initially, the size of the data block written to tape was *hardwired* at 16 KB. The drive manufacturer, STK, recommends a block size of 256 KB for maximum throughput. Configurable blocking factors for tape writes contributed to the major performance improvement, as can be seen from Tables 1 and 2.

2) Asynchronous library operation. When first tested, AMASS did not allow for asynchronous STK PowderHorn library operation on mount and dismount; no library activity took place during the time when any one of the drives was performing a tape load, positioning, or rewind. Needless to say, the associated time loss was unacceptable. The correction now allows for a fully asynchronous rewind and unload of tapes.

The asynchronous tape mount correction was accompanied by time-out problems because the hardwired retry parameter did not quite account for the time required to rewind a 50 GB tape cartridge. There is now an adjustable parameter controlling the number of retries of the drive access after the tape mount. Having configurable control over the number of retries solved the time-out problem on the mount side.

3) Introduction of Asynchronous I/O. Asynchronous I/O contributed to raising the overall performance ceiling by allowing multi-threaded transfers to AMASS cache and from it.

4) Removal of a 2 GB limit on the size of a single AMASS cache partition. Beginning with Version 4.9 for IRIX 6.2, AMASS supports a maximum total cache partition size of 1 TB,

and the total cache size up to 2 TB in as few as two partitions[2]. Although the larger cache partition size has no direct effect on the individual channel throughput rate, large total cache size allows larger number of simultaneous archive processes. In turn, the large number of simultaneous reads and writes affords a greater queuing and multithreading efficiency.

6 Conclusion

The paper contains an overview of configuration adjustments and the corresponding performance improvements in the GSFC ECS DAAC archive configuration. The information presented here is a snapshot for November, 1997 and represents work in progress. Further performance improvement are expected in December, 1997, as a result of removal of a 2 GB per disk partition limitation and elimination of tape drive buffer flush time expenditure. Although the tuning described here took place on a specific set of equipment and software, the functional components involved are generic to large archive systems. High capacity/high performance systems are becoming more common, and, increasingly higher performing hardware is becoming available. It is hoped that our experience may be of use to integrators with similar architectures.

7 Acknowledgements

The author wishes to acknowledge Stacey Brown and Brad Koenig of Lockheed Martin Space Mission Systems and Services, and Byron Peters of Hughes Information Systems for their invaluable contributions in getting the archive configuration to the performance it has achieved today. Appreciation is due to Randy Kreiser of SGI, and Andy Richards and Leo Profilet of EMASS for guidance with their respective products. And, finally, the author thanks Robert Howard of Hughes Information Systems, Ray Simanowith of Lockheed Martin Space Mission Systems and Services, and Jean-Jacques Bedet of Hughes STX for the review of the text.

8 References

[1] Installing AMASS, EMASS, Inc. Version 4.8, December 1996, Document Number 600422

[2] Release Notes for AMASS Version 4.9, November 14, 1997, EMASS, Inc. Document Number 600459

---

[1] ECS is developed by the National Aeronautics and Space Administration (NASA) through Contract NAS5-60000 as part of NASA's Mission to Planet Earth.

[2] Here and subsequently, AMASS and EMASS are either trademarks or registered trademarks of EMASS, Inc. Silicon Graphics, Challenge, IRIX, and Performance Copilot are registered trademarks of Silicon Graphics, Inc. PowderHorn, RedWood (SD-3), and ACSLS are trademarks of Storage Technology Corporation. SPARC5 is a trademark of SUN Microsystems, Inc. UNIX is a trademark of AT&T Bell Laboratories.

Page intentionally left blank

Building a Database for the LHC – the Exabyte Challenge

Jamie Shiers
CERN
1211 Geneva 23
Switzerland
E-mail: Jamie.Shiers@cern.ch
Tel +41 22 767 4928
Fax +41 22 767 8630

Abstract: CERN, the European Laboratory for Particle Physics, is currently building a new accelerator, the Large Hadron Collider (LHC). Scheduled to enter operation in 2005, the experiments at the LHC will generate some 5PB of data per year with data rates ranging from 100MB to 1.5GB per second. Data taking is expected to last 15 or more years, leading to a total data sample of some 100PB. Designing a system that can handle such enormous data volumes implies a solution that can theoretically handle at least one order of magnitude more data than is currently anticipated, namely 1EB (exabyte). Although the production phase of the LHC is still in the distant future, elements of the proposed system have been used in physics experiments at CERN since 1996. In addition, a number of pre-LHC experiments, both at CERN and at other laboratories including SLAC in the US, have adopted the strategy described below. We expect some tens of TB to be stored during 1998 (CERN-NA45) and a few hundred TB per year in 1999 and beyond (the COMPASS experiment at CERN and the BaBar experiment at SLAC).

A strong goal of the project is to use standard, commodity solutions wherever possible. We describe the progress on the project to date, the standards and solutions that are currently being used, performance and scalability measurements as well as plans for the future.

Further information on this project may be found via http://www.cern.ch/ via the link Research and Development and then RD45.

1 Introduction

In this paper we describe on-going work that is aimed at finding a solution to the data management problems that future experiments at CERN and elsewhere face. The main challenges are the volumes of data (100PB or more) and the timescales of the project (some 25 years in total). Our current activities focus on the use of a standards-conforming Object Database Management System coupled to a Mass Storage System. We show how these two systems can work together offering a solution that is significantly more powerful than a more traditional file-based approach.

2 The Large Hadron Collider (LHC)

High Energy Physics (HEP) can be loosely defined as the study of the fundamental particles that make up matter, and of the forces that act between them. Over the years, the so-called "standard model" has emerged, which comprises 3 families of quarks (up, down, strange, charm, top, bottom) and 3 families of leptons (the electron and its neutrino, the muon and tau ditto). In addition, there are a number of force-carrying particles, such as the photon, Z^0, $W\pm$, gluon and graviton. However, a number of mysteries remain, including the origin of mass itself, and the reasons behind the existence of the 3 observed families. Do these hide yet further sub-structure? It is to answer questions such as these that CERN is building the LHC - a proton-proton collider that will be housed in the tunnel currently occupied by the Large Electron Positron collider (LEP) at CERN, and will enter operation around 2005. The LEP ring has a circumference of 27km and lies some 100m below the surface, straddling the border between France and Switzerland, just outside Geneva. The LHC will host several experiments, including two general purpose facilities, ATLAS and CMS, and an experiment to study the nature of the bottom quark, named LHC-B. The LHC will also be able to accelerate heavy ions, and not just protons, and the ALICE experiment will study the interaction of these ions. The data volumes that will be generated at the LHC will be dominated by the two general-purpose experiments and ALICE - each of these experiments will generate around 1PB of data per year. Including the data from LHC-B, as well as processed and simulated data, a total data volume of around 5PB/year is expected. The LHC itself is expected to run for 15-20 years, giving rise to a total data volume of between 75-100PB. Despite these huge figures, it is unclear which is the more difficult problem - the sheer volume of data involved or rather the long timescales. It is impossible to predict the changes that will occur between now and the end of LHC data acquisition; it is even harder to foresee those that will take place between now and when the LHC starts up, in some 8 years time!

3 The RD45 Project at CERN

In order to investigate possible solutions to these problems, the RD45 project was established at CERN in early 1995. The fact that such a project was established 10 years prior to the first data taking indicates how seriously the problem was viewed within our community. Unlike previous data management projects at CERN, which were largely based on the use of homegrown solutions, often implemented in Fortran, RD45 from the start assumed that the LHC experiments would adopt object-oriented solutions and C++ as the implementation language. This represented a very significant change over previous practice, and opened the door to certain solutions that would not have been possible had we decided to stick with more traditional approaches. Given the very long timescales involved, a first step was the identification of the relevant standards. As indicated by the choice of object-oriented solutions, we were particularly interested in standards related to object persistency - i.e. the ability of an object to persist longer than the lifetime of the creating process. Possible solutions included the Object Management Group's (OMG) Persistent Object Service (POS) and the Object Database Management Group's standards for Object Databases. In this case, the choice was rather straight-forward - the OMG POS was felt by many to be flawed - even unimplementable (it has since been withdrawn!), whereas there were numerous products on the market that claimed to be ODMG-compliant. However, we did not limit ourselves to a study of object databases (ODBMS) as potential solutions. Using books such as Cattell's Object Data Management and Barry's DBMS Needs Assessment as guides, we investigated a wide range of possible solutions, including language extensions, so-called *light-weight* object managers, and full-blown ODBMSs. Based on a rather simple set of initial requirements, such as the need of support for platform heterogeneity and the full C++ object model, we were rapidly able to eliminate both language extensions and object managers. However, there was considerable skepticism within our community that a commercial ODBMS could handle our requirements, if only in terms of data volume. At this point, it is perhaps interesting to note the initial milestones given to the project:

RD45 should be approved for an initial period of one year. The following milestones should be reached by the end of the first year.

1. *A requirements specification for the management of persistent objects typical of HEP data together with criteria for evaluating potential implementations.*
2. *An evaluation of the suitability of ODMG's Object Definition Language for specifying an object model describing HEP event data.*

3. *Starting from such a model, the development of a prototype using commercial ODBMSs that conform to the ODMG standard. The functionality and performance of the ODBMSs should be evaluated.*

It should be noted that the milestones concentrate on event data. Studies or prototypes based on other HEP data should not be excluded, especially if they are valuable to gain experience in the initial months.

Milestone 1 was subsequently amended to:

- *Produce statement of the probable capabilities of a HEP persistent object manager based upon commercial object database management systems (ODBMS) and large-market mass storage systems (MSS).*

3.1 Design Principles

A guiding principle behind our activities was the following statement from the ODMG book "The Object Database Standard" :

"The programming language-specific bindings [...] are based on one basic principle: The programmer feels that there is one language, not two separate languages with arbitrary boundaries between them."

This principle contrasts sharply with what was possible previously in HEP. Programs were typically written in Fortran, where as the I/O system was handled by a separate package with a non-intuitive interface. A similar situation is also true when relational databases are used to provide persistency - the programs are written in, *e.g.*, C, whereas the interface to the database is either provided via a proprietary interface, or by embedded SQL, with associated data copies. A general rule of thumb is that approximately 30% of the code is dedicated to this interface, which indeed matches our experience. Aside from the extra development and maintenance involved, an associated problem is the possible lack of data integrity that this involves.

3.2 Using an Object Database

On paper, an ODBMS offers many of the features that we require. The various language bindings that are provided are well integrated with the corresponding programming languages themselves, they offer an extremely rich set of functionality, including support for hardware, operating system and even compiler heterogeneity, schema evolution, object versioning and user data replication. They even offer the promise of language independence - in other words, data that is written today using, *e.g.*, C++, will still be accessible in the future, perhaps using Java. Such features make them extremely attractive as possible building blocks for a multi-PB object store. However, can such systems scale even to the TB range, let alone 100PB? Although there are many examples of production use of object databases, the amount of data stored seems to be small - typically not exceeding a few GB or tens of GB. How can such a solution be applied to a problem many orders of magnitude greater? To explain how we believe that this can be achieved, it is important to emphasize some of the particular characteristics of our data. Firstly, although the volume of data is very large, there are very few dependencies between different parts of the data. To a very large degree, the information recorded from the final-state products of an interaction between two accelerated particles in one of the LHC detectors, each interaction being called *an event*, is completely independent. Furthermore, the data itself is largely read-only and the transaction and concurrency rates extremely low. In addition, there is a very natural hierarchy within the data. Much of the data for a given event - the raw data - is processed typically once, or once per year, in a sequential manner. Increasingly small subsets of a given event are accessed ever more frequently, depending on the key characteristics of the event. These characteristics in turn provide another way of "sorting" the data - different event categories having markedly varying access patterns and frequencies. Basically, our strategy is to use a single, logical ODBMS that is constructed out of many physical databases - in other words, a distributed object database. The individual databases themselves may be distributed over many database servers in the LAN or WAN. Depending on the characteristics of the data stored in these databases, they may even be offline, e.g. on tape.

3.3 Building a multi-PB Object Store

As described in "Object Databases and Mass Storage Systems: The Prognosis" , we do not expect any major technological problems in building a multi-PB store in 2005. Certainly, in the case of disk storage, the trends are such that very large - perhaps as large as several hundreds of TB - disk farms will be both possible and affordable. Less clear, however, is the evolution of tape technology, as there appears to be little demand for capacity much beyond what is currently available. Despite the trend in disk capacity/$, we nevertheless feel that a multi-PB disk farm in 2005 can certainly not be assumed. Hence, our strategy is to combine an ODBMS with a suitable mass storage system, and thus permit multi-PB object stores to be constructed. The issues that remain are the selection of the individual components, namely an ODBMS and MSS capable of scaling to the required region, and their integration. This is described in more detail below.

3.4 Tests with Real Data

At the end of 1995, we were contacted by an existing CERN experiment, NA45/CERES, that had just taken the decision to move to C++. They were therefore searching for a solution to their object persistency needs. Although the data volumes that were involved were rather small - a few tens of GB - this provided an excellent opportunity for testing our ideas in practice, with real physics data and a long time before the start up of the LHC. Until then, the tests that had been made were performed using legacy data, which did not have an associated object model, or generated data that did not necessarily correspond to a realistic object model. In addition, we were able to test the impact of providing persistency via an object database on issues such as program design. The results were very largely positive - despite using a version of the operating system (Solaris) that was not supported by the database that we were using, and running on an unsupported platform (Meiko/Quadrics CS-2), we were nevertheless able to use the product in production, and store some 20GB of data. In addition, the data were processed by some 16 nodes running in parallel, something that is not possible on our existing systems, which do not support concurrent write access to individual files.

As a result of these activities, the project was approved for a second year, with the following new milestones:

1. *Identify and analyze the impact of using an ODBMS for event data on the Object Model, the physical organization of the data, coding guidelines and the use of third party class libraries.*
2. *Investigate and report on ways that* [ODBMS] *features for replication, schema evolution and object versions can be used to solve data management problems typical of the HEP environment.*

3. *Make an evaluation of the effectiveness of an ODBMS and MSS as the query and access method for physics analysis. The evaluation should include performance comparisons with* [existing systems used in HEP].

3.5 Product Choices

As a result of the investigations performed in producing , we were able to identify a number of potential solutions to our problem. Basically, the biggest problem that we needed to solve was that of scale - we needed an ODBMS with architecture capable of scaling to the multi-PB region, and an MSS with similar capabilities. Having ruled out non-ODBMS based solution on functional grounds, we were left with a number of products claiming ODMG compliance. Of these products, we feel that only two have the possibility of scaling to the PB region. The fact that even one such product existed, let alone two, was certainly very encouraging. This is particularly important given the relatively small size of the ODBMS market, and the ODBMS vendors themselves. Indeed, we believe the long-term survival of any given vendor to be one of the largest risks in our strategy. Here, the use of an ODMG-compliant product helps - in theory, at least, an application that uses one of the ODMG bindings can be ported to another compliant product by a simple recompile, and the data itself can be migrated using the ODMG's vendor-neutral exchange format, the Object Interchange Format (OIF). In reality, there are likely to be many issues that significantly complicate such a move. Not least is - again - the volume of data involved. Migrating a few MB or GB of test data is one thing, whereas converting a few PB of data is another. In addition, the ODMG does not attempt to define implementation, but only interface. Hence, the architectures of the various products claiming conformance are very different. Thus, an important aspect of our future work will be to investigate the issues involved in such a migration, which may well be needed given the timescales involved. Some of the issues that need to be addressed include support for data clustering, which will clearly be crucial if one is to achieve efficient access to such large volumes of data.

On the positive side, both of the ODBMSs that we believe to be scalable to the PB region sell strongly into the telecom and financial markets. Whereas their long-term survival can still not be guaranteed, it is unlikely that either of these markets will disappear, and give reason to believe that at least one product meeting the requirements of these markets will continue to exist.

On the MSS side, things are arguably less rosy. Although there are a number of mass storage systems that are available, there is, as yet, no set of standards for the interfaces to such systems, although there is a model - the IEEE reference model on which all recent systems are built. A single MSS, namely HPSS, is designed to scale to the PB region. However, whereas ODBMS vendors count the number of deployed licenses in the hundreds of thousands, this is far from the case with mass storage systems - at least today. The selection of a mass storage system, and the provision of such services at CERN, is outside the scope of the RD45 project. However, an important issue that we do address is the integration of the ODBMS with the MSS. This issue is covered in more detail below.

3.6 Scalability Tests

Having identified a potential solution - at least on paper - it is clear that the overall scalability of the architecture must be verified. At the time of writing, this has only been done using one of the two potential ODBMSs, namely Objectivity/DB. However, by the time of the symposium, we will have repeated these measurements against our alternative solution, namely Versant.

In performing these measurements, we have tested every explicit limit of the architecture of Objectivity/DB, and attempted to find arbitrary, non-documented limits which could constrain us. To explain these tests, a slight digression into the overall architecture of Objectivity/DB is required.

Objectivity/DB supports a so-called *federated database* - in their terminology, this corresponds to a distributed database with consistent, shared schema. In Objectivity/DB, each persistent object has a 64-bit object identifier (OID) associated with it. The 64-bit OID is divided up into 4 fields, each of 16 bits. These sub-fields are used to indicate the (physical) database, the "container" (a set of contiguous pages within a database), the logical page and the slot within the page on which an object resides. Up to 2^{16} databases are permitted per federation, and each database currently maps to a single file. We were able to verify that it was possible to generate 2^{16} databases in a federation, and that individual databases of up to 9GB were possible. This in itself is not a limitation of Objectivity/DB, but is the maximum database size that we believe is reasonable today. The largest federation that we have created to date is 0.5TB - limited by the available disk space. As described below, we have plans for multi-TB test federations and production federations of many hundreds of TB in the near future.

3.7 Enhancement Requests

Although the above measurements confirmed that multi-PB federations are indeed theoretically possible, the maximum achievable size of a federation is clearly limited by practical considerations, such as the maximum database or file size. As a rule of thumb, we estimate that it should be possible to migrate or recall a file in 10^2 - 10^3 seconds. At 10MB/second, it would take 1000 seconds to recall a 10GB database. Even using techniques such as striping, it is unlikely that one can reduce this time much beyond 10^2 seconds. Using such arguments, we believe that a maximum practical file, and hence database size, is today limited to a few GB. By 2005, it is fairly safe to assume that this will have risen to around 100GB. However, 2^{16} databases of 100GB only permit federations of 6.5PB in size - our requirement is for federations up to 100PB. Furthermore, so as not to introduce any arbitrary constraints, we feel that architecturally, the solution should scale by at least one order of magnitude more than is required, i.e. to 1EB or 10^{18} bytes. Hence, our principal enhancement request concerns architectural changes that would permit EB federations, without, for example, requiring massive files. Essentially, all of the possible options involve permitting more files/federation, by increasing the length of the OID, by allocating more bits to the DBID (fileid) within a 64-bit OID, or changing the logical-physical mapping, such as permitting multi-file databases. We are currently discussing this issue with Objectivity, and hope to have a solution in place by the end of 1998.

3.8 Mass Storage Interface

As described above, we believe that one cannot assume that affordable, manageable disk farms in excess of a few hundred TB will be possible by the year 2005, and hence a means of integrating the ODBMS with a mass storage system is required. The solution that we have identified, and that has been implemented - so far as a proof-of-concept prototype by collaboration between Objectivity and members of the HEP community - is to perform the integration at the level of the Objectivity server. This server is implemented as a page server, and knows nothing about the objects themselves - such knowledge is the responsibility of the Objectivity client. Thus, the server performs basic I/O functions - opening files (databases), seeking to the right offset, reading blocks (database pages) and transferring this information to the client. As such, this offers a fairly natural interface to a mass storage system such as HPSS. Rather than use the standard file system calls, one "simply" has to replace this layer with calls to the HPSS client API - itself modeled on the Posix filesystem calls. "Heterogeneous" federations are possible, namely ones where some databases are stored in HPSS-managed storage, and others - presumably the bulk of the data - are managed by HPSS. Apart from the need to store some data, such as metadata, permanently online - which could also be achieved using HPSS - it would not be possible, or even practical, to store all of the possible 2^{16} databases in HPSS. One may wish to store some databases locally on systems for which no HPSS client yet exists, some databases will be stored at remote sites, where again one cannot assume HPSS - one can even foresee databases on mobile computers that are not permanently connected to the network.

At the time of writing, the HPSS client is only available for IBM systems, although ports to other platforms are known to be in progress. By performing the interface at the level of the Objectivity server, it is still possible to use non-IBM client machines, including many other Unix platforms and also Windows NT. Furthermore, it isolates the client from the somewhat complex environment that is needed today to support HPSS, including DCE.

3.9 Current Activities

At the time of writing, the focus is on a demonstration that an ODBMS+MSS-based solution can satisfy the key requirements of the entire LHC production chain - from data taking to physics analysis. The current milestones, which need to be addressed by the next project review in April 1998, are given below:

1. *Demonstrate, by the end of 1997, the proof of principle that an ODBMS can satisfy the key requirements of typical production scenarios (e.g. event simulation and reconstruction), for data volumes up to 1 TB. The key requirements will be defined, in conjunction with the LHC experiments, as part of this work.*
2. *Demonstrate the feasibility of using an ODBMS + MSS for Central Data Recording, at data rates sufficient to support ATLAS and CMS test-beam activities during 1997 and NA45 during their 1998 run.*
3. *Investigate and report on the impact of using an ODBMS for event data on end-users, including issues related to private and semi-private schema and collections, in typical scenarios including simulation, (re-)reconstruction and analysis.*

Since these milestones were set, changes to the accelerator schedule at CERN have shifted the above-mentioned NA45 run from 1998 to 1999. NA45 expects to acquire some 30TB of data during this period. In addition, a new experiment at CERN, COMPASS, which is due to start taking data in 1999, has adopted for the same solution, but will have somewhat more demanding requirements. Although 1999 will still be a preliminary run, they expect to take roughly 360TB of data per year when they enter full production, perhaps as early as 2000.

The requirements of the LHC experiments are somewhat harder to quantify. Perhaps the easiest numbers to give are the I/O requirements for data taking and first-pass processing. Here, data rates of 100MB/second need to be supported for ATLAS and CMS, and perhaps as high as 1.5GB/second for ALICE. It is assumed that many systems will be used to perform the processing, with estimates varying from 100-500 nodes. Thus, the data rates per node are somewhat modest - a mere 1MB/second. The ALICE experiment nominally requires much higher data rates, but over a much shorter time period - just one month per year. A viable solution would be for them to process this data at correspondingly slower rates throughout the rest of the year, yielding the same I/O rates - as ATLAS and CMS. Existing experience from NA45 shows that 32 parallel streams are indeed possible, and thus we are confident that we will be able - in 8 years if not before - to handle 3 times as many streams.

Some of the more exotic options that are being considered for the LHC are recalculation rather than storage - the needed data are dynamically and transparently reprocessed using the latest algorithms and calibration constants, and automatic data reclustering based upon access patterns. It is clear that such scenarios could result in very high demands, both in terms of I/O bandwidth and also in available processing power, and have to be studied further. However, more traditional styles of access, with controlled reprocessing and reclustering at perhaps monthly intervals, again requires data rates of just a few MB/stream, but up to 100 or so streams.

3.10 Object Clustering

In Objectivity/DB, complete database pages are transferred between client and server. Not only is the overhead for transferring a complete page rather small compared with the transfer of individual objects, but this also decreases the load on the server systems. In addition, if the page in question contains not only the object that is being requested, but also other objects of interest, there is both decreased server load and network traffic. Conversely, if the page contains just one object of interest, the overhead will increase. Similar arguments are also true concerning the recall of complete databases (files) from tape - if the file being restored contains only a few objects of interest, the overhead will be significant and the system will perform poorly. Thus, it is clearly highly advantageous if efficient data clustering can be performed.

In the ODMG standard, a clustering hint may be given when a new object is created. This tells the database to store the new object "near" another object - be it on the same or adjoining database page, or perhaps just in the same container or database. As this hint is given on a per object basis, it is possible to define complex clustering strategies. For example, it is possible to store the various objects that make up an event and even the objects that make up events with certain characteristics according to different policies. Of course, it is not possible to define a strategy that is ideal for all possible types of access. However, by optimizing for the main access patterns, the overall efficiency of the system can be greatly increased.

3.11 Metadata

Although usually described as being "data about data", we define "metadata" to mean "data that makes other data useful/usable". There are numerous examples of such data in HEP, ranging from calibration data, needed to correct for variations in the response of the detectors, to the schema of the various persistent objects, to data that helps users find the data that they are interested in. Clearly, metadata and the corresponding data are associated with each other. When using an object database, such relations can be represented directly - by associations between the metadata objects and the data themselves. Although there are sound arguments for the separation of data and metadata, these arguments are valid for physical, rather than logical, separation. Clearly, it is not useful to have to restore a 100GB database from tape simply to discover from the metadata that the data are in fact not needed. However, maintaining the logical connection between the two has immediate advantages. Firstly, navigation from the metadata to the data is trivial. Secondly, if implemented using bi-directional associations, then the database itself can ensure consistency. Neither of these is true if separate systems are used.

Calibration data are expected to total some 100GB per year (compared to 1PB of data). The schema themselves will not represent a significant amount of data - it is not yet clear how many persistent-capable classes there will be, but it is likely to be of the order of 10^3. Metadata are expected to be widely used as the entry point into the system. We expect users and analysis groups to store collections of pointers to objects that correspond to specific event selections. The criteria used to make these selections will also be stored in the database. We anticipate that these event collections will be named according to some simple - probably hierarchical - naming schema. This will allow straightforward access to collections of events, e.g. "MyCollection" or "Higgs_Events". It will also permit selection based on the criteria used to define these collections. "MyCollection", for example, could refer to all events with a transverse energy greater than a certain quantity and 2 electron candidates. Not only is it useful to have the database manage the definition of "electron candidate", which may well vary according to context, but such a collection is clearly a better starting point for a selection requiring the same E_t cutoff but 4 electron candidates than the entire 100PB store!

3.12 HEP-Specific Extensions

Although an ODBMS offers a rich set of features, we have found a number of areas where extensions have been necessary. Largely speaking, these fall into two categories:

1. Database Administration (DBA) Tools.
2. Class libraries that extend the programming interface.

The DBA tools that have been developed within the RD45 project are written using the Java ODBMS binding. Although the basic functionality is also provided via "command-line" tools and also through the various language bindings, it can be convenient to provide a more user-friendly interface, particularly when dealing with some of the more complicated options, such as data replication. These tools also allow for comprehensive error checking and allow for site customization. For example, data replication, which is today supported in Objectivity/DB at the level of a physical database, is likely to be managed according to a specific policy, which is clearly unknown to the database vendor. Given the large number of databases and sites that will exist, it would be extremely inconvenient to manage replication with the low-level tools that are available. These require the DBA to explicitly state which database(s) should be replicated to which node(s). A more likely scenario is that a set of databases, *e.g.*, those corresponding to a certain physics channel, will be replicated to a set of sites, *i.e.*, the ones participating in the analysis in question. Clearly, it is relatively simple to build a tool with an intuitive drag-and-drop interface that uses the database itself to manage the necessary configuration information.

The class libraries that have been developed so far help to minimize the dependence on a given database vendor and smooth out the changes between various releases. In addition, they reduce the changes that are necessary in the conversion of a transient application to a persistent one. Most importantly, they provide support for sophisticated data clustering strategies, event collections and associated metadata, and so forth. They have been used, for example, in the NA45 production runs to provide a lock-free strategy that permitted up to 32 nodes to write to the database in parallel. In the past, such an operation would have required the use of separate files with an expensive "merge" step at the end.

3.13 Production Plans

Although the production phase of the LHC is still many years away, large-scale production is expected to start in early 1998. In addition to ongoing work with the NA45 and other experiments, the LHC experiments are expected to increasingly use the ODBMS+MSS solution for the storage of simulated data and for test-beam runs. To accommodate these production plans, we intend to set up production services for 4-6 different experiments from the beginning of 1998. Each of these experiments will use a separate Objectivity/DB federation, with its own set of servers. Initially, we will establish a single lock-server per experiment, with one or more data-servers, each handling a few hundred GB of disk space, as required. At a later stage, the data-servers will also run HPSS disk movers, and will be connected to the tape servers - shared across the different experiments - by HiPPi-class networks. Explicitly, we expect a production version of the Objectivity/DB - HPSS interface no later than Q4 1998.

3.14 Comparisons with Existing Systems

The systems that are used in today's high energy physics experiments provide considerably less functionality than is described above. Not only are they poorly integrated with the programming language, but a variety of separate packages, with inconsistent interfaces, are used for different aspects of the problem. For example, to locate the raw data corresponding to the events that give an anomalous signal in a particular analysis requires an ad-hoc and labor-intensive process. Using an integrated approach based upon an object database, such an operation is trivial.

A good example of the manpower savings made possible by using an object database is that of a calibration management system. Existing experiments have typically implemented their own systems. These systems, which are significantly less complicated than a complete data management system, have been estimated to take up to 10 man-years, of highly experienced personnel, to write and 1-2 full time equivalents to support. Using an ODBMS as a basis, a working system can be built by a student in less than one year. Indeed, it is believed that the effort previously required within an experiment to support just their calibration data may well be sufficient to manage their entire data sample!

4 Summary and Conclusions

We have demonstrated the feasibility of using off-the-shelf, commercial solutions, as the building blocks for a multi-PB distributed object store for HEP data. Although a small amount of code has had to be written to integrate and extend the various components, the savings with respect to "roll-your-own" solutions are considerable. Not only does such a solution offer considerably more functionality than existing, home-grown solutions, but it is also markedly more scalable.

5 Acknowledgements

The work described in this paper is the result of many people's effort over several years both within the RD45 project and outside. Eva Arderiu Ribera wrote the DBA tools described above. Dirk Düllmann largely wrote the class libraries, developed to extend the functionality of the ODBMS, with contributions by Marcin Nowak and others. Some of the classes were based upon examples obtained from Objectivity. Objectivity and Andy Hanuchevsky of SLAC implemented the interface between Objectivity/DB and HPSS. Members of the CERN IT/PDP group run the HPSS system installed at CERN. Finally, we owe much to the NA45 collaboration, who were brave enough to entrust their data to something new.

6 References

[1] RD45 - A Persistent Object Manager for HEP, LCB Status Report, March 1998, CERN/LHCC 98-x
[2] Using an Object Database and Mass Storage System for Physics Production, March 1998, CERN/LHCC 98-x
[3] RD45 Project Execution Plan, 1997-1998, April 1997, CERN/LCB 97-10
[4] RD45 - A Persistent Object Manager for HEP, LCB Status Report, March 1997, CERN/LHCC 97-6

[5] Object Databases and their Impact on Storage-Related Aspects of HEP Computing, the RD45 collaboration
[6] Object Database Features and HEP Data Management, the RD45 collaboration
[7] Using and Object Database and Mass Storage System for Physics Analysis, the RD45 collaboration
[8] RD45 - A Persistent Object Manager for HEP, LCRB Status Report, March 1996, CERN/LHCC 96-15
[9] Object Databases and Mass Storage Systems: The Prognosis, the RD45 collaboration, CERN/LHCC 96-17
[10] Object Data Management. R.G.G. Cattell, Addison Wesley, ISBN 0-201-54748-1
[11] DBMS Needs Assessment for Objects, Barry and Associates (release 3)
[12] The Object-Oriented Database System Manifesto M. Atkinson, F. Bancilhon, D. DeWitt, K. Dittrich, D. Maier, and S. Zdonik. In Proceedings of the First International Conference on Deductive and Object-Oriented Databases, pages 223-40, Kyoto, Japan, December 1989. Also appears in 13.
[13] Object Management Group. The Common Object Request Broker: Architecture and Specification, Revision 1.1, OMG TC Document 91.12.1, 1991.
[14] Object Management Group. Persistent Object Service Specification, Revision 1.0, OMG Document numbers 94-1-1 and 94-10-7.
[15] The Object Database Standard, ODMG-93, Edited by R.G.G.Cattell, ISBN 1-55860-302-6, Morgan Kaufmann.
[16] ATLAS Computing Technical Proposal, CERN/LHCC 96-43
[17] CMS Computing Technical Proposal, CERN/LHCC 96-45

Page intentionally left blank

A Data Handling Architecture for a Prototype Federal Application

Chaitanya K. Baru, Reagan W. Moore, Arcot Rajasekar, Wayne Schroeder, Michael Wan
San Diego Supercomputer Center, 10100 Hopkins Drive, La Jolla, CA 92093
(baru,moore,sekar,schroede,mwan@sdsc.edu)

Richard L. Klobuchar, David M. Wade
Science Applications International Corp.(SAIC)

Randall K. Sharpe, Jeff Terstriep
National Center for Supercomputing Applications, Urbana-Champaign, Illinois
(rsharpe,jefft@ncsa.uiuc.edu)

Abstract: The Distributed Object Computation Testbed (DOCT) Project is a collaboration led by the San Diego Supercomputer Center (SDSC) and funded by the Defense Advanced Research Projects Agency (DARPA) and the US Patent and Trademark Office (USPTO). The DOCT project was initiated with the objective of creating a testbed system for handling complex documents on geographically distributed data archives and computing platforms. The project focuses on technologies that apply to the information needs of federal agencies, such as the National Science Foundation, the National Institutes of Health, the Nuclear Regulatory Commission, the Environmental Protection Agency (EPA), the Department of Energy, and the Department of Defense. In particular, the patent filing and amendment processing application of the USPTO is used as the prototype application for this testbed. This paper describes the DOCT system architecture and the USPTO application.

1. Introduction

The Distributed Object Computation Testbed (DOCT) enables the study of distributed processing issues in a geographically dispersed, heterogeneous computing environment. The DOCT project has been studying issues in implementing robust information processing systems in such environments. The testbed includes distributed computational and storage resources interconnected via high-speed networks, including OC-3 to OC-12 class networks. The computer systems which comprise the testbed include a 23-node IBM SP-2 at SDSC and an SGI PowerChallenge at the National Computational Science Alliance (NCSA) in Illinois; the High Performance Storage System (HPSS) archival system at SDSC[1] and at the California Institute of Technology (Caltech); database management systems at the offices of the Science Applications International Corporation (SAIC) in Virginia, SDSC, and NCSA; and, the vBNS and AAI national networks through NSF, ARPA, and the Naval Command, Control and Ocean Surveillance Center (NCCOSC) in San Diego. Software systems that have been prototyped as part of this project include, a Java-based framework for developing and deploying distributed software agents[2]; a Storage Resource Broker (SRB) middleware, which provides uniform access to distributed, heterogeneous storage systems[3]; the Network Queueing Environment (NQE) scheduling system[4] and the Network Weather Service (NWS) network monitoring system[5]; and authentication, encryption, and intrusion detection systems[6]. The testbed also includes a number of commercial off-the-shelf software systems such as the Texcel document management system[7], the OpenText text indexing/search system, and Oracle[8], DB2[9], and ObjectStore[10] database management systems. A prototype version of DB2 is also used, which integrates DB2 with HPSS[12].

The USPTO application requires support for electronic filing, searching, and archiving of complex, multi-mode documents in a geographically distributed system. In addition, there is a need to provide a workflow system to automate office operation; maintain an archival audit trail of all office actions; provide Web-based access to the published patent database; and, convert legacy patent data. These application requirements are similar to those found in a number of other federal agencies. To support these requirements, the DOCT infrastructure provides mechanisms for handling very large, complex databases in a robust, fault-tolerant, metacomputing environment. In addition, we have also developed prototype software specifically to support secure electronic filing of documents; document validation; document classification; and, data mining.

The next section discusses the DOCT system in terms of the document flow for the USPTO application, and also highlights the DOCT data handling systems. Section 3 discusses the security infrastructure of the DOCT testbed. Section 4 describes the agent framework, which allows easy development of client applications in this environment. Finally, Section 5 provides a summary.

2. Document Flow

Figure 1 shows the various DOCT components involved in supporting the USPTO application. The following subsections describe the document flow in detail, beginning with the arrival of the patent application at the *Electronic Mailroom*, and ending with the publication of the final patent in the Patent Database.

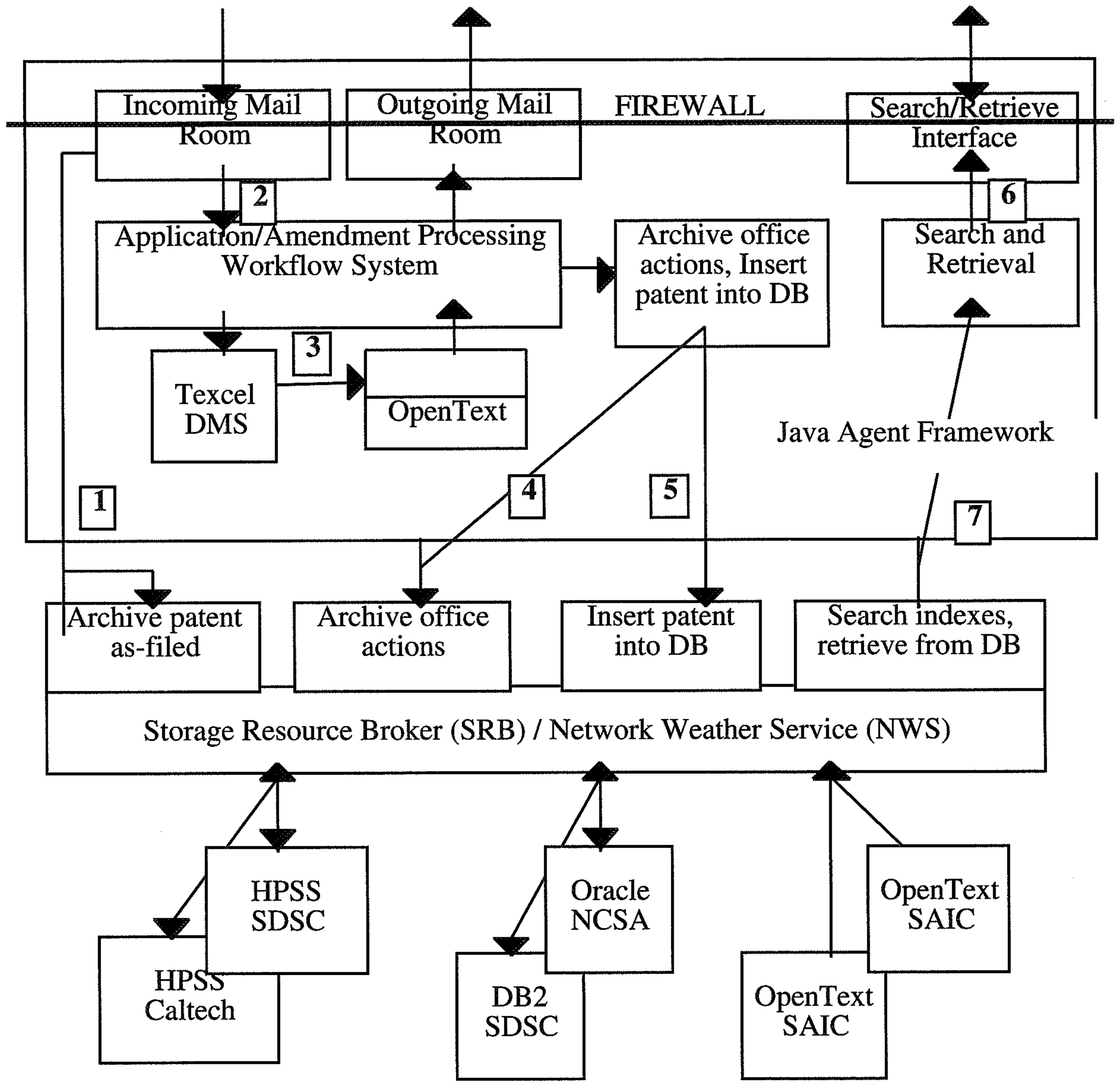
2.1 The Electronic Mailroom

The key requirement of the USPTO application is the ability to support electronic filing of patent applications and amendments via the Internet. Applications and amendments arrive at the *incoming Electronic Mailroom.* Text files are in SGML form while images are in tif format. File and document validation is performed on all received files. Validated files are sent across the security firewall to the internal workflow systems (see Section 2.2). The security aspects associated with these communications are described in Section 3.

During application processing, there may be several "office actions" initiated by patent examiners. These include requests for further information from the applicants and amendments to the patent application, such as deletion of claims. Each of these office actions results in communication with the patent applicant. Communications originating from USPTO to the applicant are routed via the *outgoing Electronic Mailroom.* Each office action, including type of action and date/timestamp, is recorded and archived along with the application, in order to maintain an audit trail of office actions.

2.2 Application Processing Workflow

Validated applications are transferred from the Electronic Mailroom to the internal systems, which operate behind a security "firewall". Here, the patent application is first classified based upon its subject matter. This determines the area/group within USPTO that will be responsible for examining this application. Each area within the USPTO is associated with a separate workflow. The application is inserted into the corresponding workflow (Step 2 in fig.) and, simultaneously, it is indexed by a text indexing system to enable efficient text-based searching of active applications and to support *interference searching* (Step 3 in fig). An interference search is used to determine if a newly submitted patent application has overlapping claims with any of the other currently active patent applications.



2.2.1 Document Classification

We undertook a study at NCSA to evaluate the potential of employing neural network-based techniques for automatic classification of patent applications. The project investigated the possibility of using *Kohonen Self-Organizing Maps (SOM)*[13] in the "reclassification" problem and the use of *Learning Vector Quantization (LVQ)* technique[14] in the "preemptive classification" problem.

The SOM technique is based upon *feed-forward* neural networks. It lends itself to "unsupervised" clustering of existing data, from which classification labels may be obtained. Using this approach, it is possible to collect together documents with similar content, while separating them from other documents with dissimilar content. LVQ is based upon the *feed-forward back-propagation* neural network and lends itself to "supervised" targeting of new data into pre-existing classification schemes. It is good at evaluating document content based on its context, and is able to use this evaluation to quantify document similarity. We used the public-domain implementations of the SOM and LVQ implementations, viz. SOM-pak and LVQ-pak which are available from the Neural Network Research Center[15]. Further discussion and descriptions of this software, including source code, can be found at this location.

For patent documents, the *Claims* and *Abstract* sections provide the most relevant content for the purposes of classification. Therefore, the concatenation of text from these two sections is used for classification. For SOM, the entire collection of patent documents (from 1976-1997) is used as the input for *feature vector* generation. For LVQ, a random sample of 10 documents from each USPTO-defined patent class was chosen as the input data. We employed the "histogram" method for document encoding, based on successful results from other studies conducted at NCSA[16].

Except for the difference in the choice of document corpus, the method used for creation of document feature vectors for SOM and LVQ was the same. For each document, each input word is checked against a "stop word" list, and words in this list were ignored. If the word is not a stop word, it is then *stemmed* using the stemming procedure provided in FreeWAIS[17]. This stem is then put into two count lists. The *global count list*, which keeps track of the number of documents in which a stem occurs, and the *document count list*, which keeps track of the number of occurrences of each stem within the given document. After a document is completely processed, its document count list and associated document label are pushed onto a stack. This process continues for all of the input documents. When finished, the result is a global count list that contains every distinct stem that occurred in any document, and the number of documents in which that stem occurs. Also, for each document, there is a document count list which contains every distinct stem in that document as well as the number of occurrences of that stem in the document.

Once this global information has been gathered, the *normalization* process begins. This process yields the actual feature vectors. First, the global count list is sorted in descending order of the counts. The first M items are removed from this list, where M is a predetermined number. Then the first N stem/count pairs are taken, where N is again predetermined. These provide the foundation for the normalization/projection of the document count lists. The combination of the word stemming and the "first N offset M" stem choices is used as the "blurring" agent. Each document count list is then parsed for occurrences of the stems in this reduced global count list, in the order that they occur in the global count list. If the *i-th* stem in the global count list occurs in the document count list for a given document, then the count associated with that stem is the *i-th* component of the raw feature vector for that document. Otherwise, a zero is placed in the *i-th* component. After all of the N stems of the global word count have been checked, the raw vector is normalized to unit length and then paired with the associated document label. This process continues for each of the document count lists. Each of the resulting vectors in pushed onto a stack and this stack is returned as the collection of feature vectors and associated labels for the input document corpus. These features and their associated labels are then written out to disk to be used subsequently in the SOM or LVQ processes.

The SOM approach was applied to the collection of patent applications discussed above, as an attempt to aid in the resolution of the classification problem faced by the USPTO. The idea is to use the output of SOM classification as a decision support tool in the effort to modify the classification scheme. Over the course of several years, there is the possibility that, as a result of changes in science, technology, and engineering, there is *content drift* within a classification, as well as "over stacking" of a classification. Sub-fields of specialization may arise within existing fields of invention and new fields of invention may be created. This may require splitting of some existing classes and subclasses, and may cause further reclassification. In this project, we specifically studied the clustering of documents within two existing classes, to examine the possibility of creating further subclasses. Each of the sets of documents for each class was treated as a separate corpus and each document in each class was labeled by its current subclass. The results from this classification were made available as a VRML-based visualization, which shows proximity between classes, as derived by the neural network-based system. The results of this study are currently under evaluation by the USPTO.

The LVQ approach has been incorporated in the DOCT testbed, as an integral part of the document workflow. A neural network was trained using the *LVQ-pak* with the LVQ corpus described earlier. Each node is associated with a class and each feature vector of the training set was labeled with its pre-established class. After the network was randomly initialized, the network was trained by first running the features through the OLVQ1 training algorithm and the network was further conditioned by tuning it with the same input vectors and a chain of training algorithms (from LVQ3 to LVQ1). The resulting neural network became the basis for the *presumptive classification* portion of the workflow.

In presumptive classification, a new document (i.e. a new application) would be submitted to a function which encodes the document using the same global document count used by the training. Next, the vector (without a label, since it is yet unclassified) is compared with all the nodes in the neural network. The closest *n* nodes (where the value of *n* can be specified as input) are found in order, and the associated labels are returned. These labels can then be used to classify the document. If the results from this classification are deemed incorrect, the patent examiner can resubmit the document with an explicitly specified label. In this case, the neural network retrains itself using the new input vector. The effectiveness of this mechanism is also under study.

In summary, automated classification can be used both as a means of creating new classifications in existing data, as well as an aid to classifying new data into pre-existing classifications.

2.3 Archiving Data

There are two points in the document dataflow where data is archived. First, the patent application is archived as soon as it is validated in the incoming Electronic Mailroom (Step 1 in fig.), to satisfy the legal requirement of archiving all patent applications "as filed". Second, after a patent application has gone through the entire workflow, the final application is archived, along with all related office actions (Step 4 in fig). In addition, the final patent is also inserted into the Patent Database (Step 5 in fig). An office action is any action in the workflow related to the patent application, including electronic mail communication with an applicant. For each office action, the workflow system obtains a date/timestamp and a unique hash record from a digital notary service. This is stored as auxiliary information related to the patent application. By archiving all office actions, we can provide an audit trail for future use, either for legal discovery, or other investigations and studies.

The DOCT infrastructure incorporates the SDSC *Storage Resource Broker (SRB)* middleware[3], which is used for data storage and archiving. The SRB uses a metadata catalog, MCAT, to record information on where the data sets are located. Using this information, it is able to provide seamless access to heterogeneous, distributed storage systems. MCAT stores information about the contents of the database as well as the "system-level" information needed to access the database. It also implements the concept of *logical storage resource*, which is used by the SRB to support data replication across multiple *physical storage resources.* A logical storage resource may contain two or more physical resources. Any data that is written/stored into a logical resource is replicated across the corresponding physical resources. When reading data from a logical resource, the SRB picks any one of the component physical resources. The user also has the option of specifying a particular physical resource. The patent application "as filed" and the final application are archived in an SRB logical resource containing an HPSS system at SDSC and another HPSS system at Caltech. The SDSC HPSS installation includes a 23-node IBM RS/6000 SP, 1TB disk cache, and two tape robots with a total capacity of 120TB. The Caltech HPSS system runs on an IBM RS/6000 workstation.

2.4 The Published Patent Database

Published patents are stored in an SRB logical resource containing two databases, one of which is an Oracle database at NCSA and the other is an IBM DB2 database at SDSC. Both databases employ the same database schema to represent patent documents. Patent information fields such as, Title, Inventor, Organization, Legal Representative, are all stored as tables in a relational database. In addition, the SGML text of the entire patent application, and all related tif images, are stored as *binary large objects* within the databases. This scheme allows full *ad hoc* query capability on various information fields within a patent, as well as the ability to easily extract patent images, or the complete patent in SGML form. To provide full-text search capability, the Patent Database is also indexed using a text indexing engine.

While the schemas are the same for the Oracle and DB2 databases, the implementations are different. The data in the Oracle database at NCSA is stored entirely on disk. In the DB2 system at SDSC, the structured columns (i.e. integer, character columns) in the tables are stored on disk, while the text and image objects are stored in HPSS. This is done using a prototype version of DB2, which provides transparent access to HPSS by allowing HPSS files to be defined as DB2 *tablespace containers.* This allows users to create database tables in which data for some columns are stored on disk, while data for other columns are stored in HPSS.

2.4.1 Database Interface

We have provided a Web-based search interface to the Patent Database (Step 6 in fig). The interface allows users to specify search conditions for retrieving patents of interest. An appropriate strategy is selected for answering the query, either by dynamically generating a SQL statement which is issued against the relational database, or by using text indexes to perform full text search (Step 7 in fig). For accessing the SGML text and the images corresponding to a patent, we have experimented with two different approaches. In one, we use the prototype DB2/HPSS system mentioned above, and in the other, we use the SRB to provide seamless access to data stored in HPSS. The latter case works as follows. First, the search interface issues a query to MCAT, to determine which available databases contain patent data. Next, it issues a query to determine available access functions for the database, which can retrieve individual patents and generate corresponding HTML pages for display. We have implemented one such function where the HTML pages generated contain links to the patent inventors and to other patents referenced by the current patent. Links are also included for accessing the *Brief Summary* and *Detailed Description* sections of a patent, and any related images. Following any of these references, causes a corresponding SQL query to be executed for retrieving the relevant information from the patent database.

3. Security

Computer security is of central importance in a distributed system such as DOCT, especially when the application involves sensitive information. Modern computing practices have begun to address these needs, and several systems are now available that can be combined to create reasonably secure metacomputing systems. These systems include PGP[18], SSL[19], HTTPS[20], the Secure Shell[21], Kerberos[22], intrusion detection software[23], integrity checking tools, and underlying encryption technology such as RSA, DES, and RC5[24].

Since the economic value of patent application information is high, strong security and encryption techniques are needed to protect this information from illegal access. The software systems must be able to withstand strong attacks from third parties. Among other requirements, this may require the use of relatively long encryption keys to reduce the chance of messages being decrypted using exhaustive analysis. Our security research and prototype effort in DOCT has focused on three areas: secure electronic filing, secure infrastructure, and intrusion detection/response. These are described briefly in the following subsections. A more complete description is available in the DOCT Goal Security Architecture document[6].

3.1 Secure Electronic Filing

Secure electronic filing via the Internet is supported by means of a certificate-based security scheme. The steps involved are, (1) *Acquire a certificate from the DOCT certificate authority.* The certificate authority restricts access to approved individuals. The applicant connects via a secure web browser (HTTPS), with one-way authentication (and encryption), to the Certificate Server and requests a certificate. After manual approval, the applicant is added to the access control database, and an X.509 certificate[25] is generated and sent by electronic mail to the user, using PGP-encryption. (2) *Register with USPTO.* Using the X.509 certificate, the applicant connects to the DOCT Filing Web server, via a two-way authenticated HTTPS session. The applicant completes a registration form (via a Java application) which is then submitted to the Web server. The registration is processed, and an ID number is assigned and recorded, and an electronic mail message is sent to the applicant. This ID is used with all future filing activities. (3) *Download data.* The applicant uses the X.509 certificate to download additional information such as, filing application information, PTO Public Key, and FTP userid/password for application submission. (4) *Submit application.* First, the applicant sends his/her PGP-generated public key to the USTPO. Next, the electronic filing software guides the user through the application submission procedures. All files that are part of the application are compressed (via WinZIP), signed, encrypted, and transferred to the USPTO FTP server, which is write-only to general users. An electronic mail notification is returned from the USPTO to the applicant. The application is then validated and a second electronic mail message is sent to the applicant. The application is then routed to the internal systems where it is processed, as described earlier in Section 2.

3.2 Secure Infrastructure

At each step of the document dataflow of Figure 1, an appropriate security mechanism is employed, to satisfy the security requirement of that particular step. When the patent application first arrives at the electronic mailroom from the Internet, we use the secure electronic filing mechanisms described above. The other place where the system interacts with the external world (i.e., Internet) is at the search interface (Step 6 in fig). Here, a secure HTTP protocol (HTTPS) is used to support secure connections to the system. For internal DOCT communications, both wide-area as well as local-area, we employ the SDSC SEA security system described below. Since these communications occur mainly among Java-based software agents, an alternative is to employ Java-based encryption. One option would be to use a Java DES encryption implementation, based on a shared secret key. This would provide security against network monitoring, although the secret key protection itself is not strong. Another option would be to use a Java SSL implementation, which would provide the necessary network security, as well as better key protection by exchanging the symmetric key, e.g. DES, via RSA.

3.2.1 The SDSC Encryption and Authentication (SEA) Library

The SEA library is a strong but light-weight authentication and encryption package similar to Kerberos or SSL but tailored specifically for metacomputing and high performance computing environments. For example, it provides non-time-limited credentials, which are suitable for use in batch queuing and metacomputing environments, and multiple trust models for initial registration of users including, *password, trusted host,* and *self-introduction* schemes. The SEA library is based on RSA and RC5 encryption technologies and includes RSA key management components. It provides user/process authentication and encryption capabilities between two processes communicating via TCP/IP sockets. Authentication is accomplished via an RSA challenge/response using a random key value. Encryption is accomplished via an RSA exchange of a random key, which is used for RC5 encryption/decryption. The default SEA encryption/authentication configuration uses 512-bit RSA keys and 14-round RC5, with SDSC-added Cypher Block Chaining. This is meant to provide a moderately strong scheme, which is relatively efficient. This can be modified to provide stronger encryption, using longer RSA keys and/or more RC5 rounds.

The SEA library is available on a wide variety of Unix systems including, SunOS, Solaris, IRIX, DEC OSF1, AIX, Sun/Cray CS6400, and the Cray C90 and T3E (this includes a port of RSAREF 2.0 to the Cray C90 architecture). It is used by the SRB client/server software to (1) authenticate clients to servers, across a TCP/IP socket, (2) authenticate between SRB servers and, (3) to, optionally, encrypt control and data communications.

3.3 Intrusion Detection/Response

Regardless of the security mechanisms employed, a system must also incorporate defensive mechanisms, such as intrusion detection and response, to account for the situations when the security systems may have been compromised. The DOCT system incorporates intrusion detection software as the final defensive mechanism to detect security attacks. The intrusion detection and analysis tools employed include well-identified security contacts at each site, secure (encrypted) electronic mail between security contacts, enabling of system logs at every site, and use of integrity checking tools such as Tripwire[26]. In addition, the system also employs centralized logging using the PICS syslog facility[27]; integrity and configuration checking tools, such as MD5[24], COPS[28], Tiger[29]; and additional analysis tools such as *lsof*[30] and *ifstatus*[31]. More details on the security architecture and intrusion detection/response are provided in the DOCT Goal Security Architecture document[6].

4. Software Agent Framework

To support application development, the DOCT system provides a Software Agent Framework (SAF) along with a software Workbench Server, which provides clients easy access to various DOCT system services. The DOCT SAF is a software layer which operates on top of the DOCT data handling and security systems. It consists of a set of agent *base classes*, an agent communications infrastructure, and a collection of APIs that provide access to the various services in the metacomputer testbed. In addition, the SAF provides interfaces to external clients and software agents, which allow the clients/agents to connect into the framework and communicate with other agents in the framework.

The DOCT Workbench Server provides a Java-based interface for connecting clients to the metacomputer testbed. The clients are Java-based applications, which log in with the Workbench Server upon first connecting to the system, and which use services offered by multiple other agents. The Workbench Server performs user/client authentication upon first connect, and subsequently also monitors the status of agents in the system. At first connect, a client receives an X.509 security certificate, which allows it to access and use other agent resources. A Workbench Service can be a general use interface (e.g., monitoring status of the metacomputer), or a specific application used within the workflow and document management functions provided by the metacomputer (e.g., an in-box and electronic file wrapper for a particular document that is being processed).

4.1 Types of Software Agents

Software agents may be *transient* or *persistent*. Transient agents are created to respond to specific requests and are terminated upon completion of that request. Persistent agents run as *daemons* and can be of two further types. First, persistent agents with "internal" services are those in which the services provided are executed within the single agent process that is started. Second, persistent agents with "internal and transient" services are those where services provided may be executed within the original persistent agent process, or transient processes may be spawned to perform the requested services, or both. The transient processes can be run within the metacomputer or across the Internet, as appropriate.

Each agent has a set of base classes that allows for common security, communications, monitoring, and persistent storage capabilities. The SAF provides a scheduler and interfaces for starting and running agents in the metacomputer. The agents implement specific tasks and intelligence models in Java or C++, depending on the tools with which the agent communicates. The client component of the agent is always written in Java, allowing for a lightweight, heterogeneous client. The Java clients can be combined to create DOCT Client Workbenches for specific classes of users of the system. Clients communicate with the corresponding servers incorporated in the Workbench Server, which serves as an *agent factory*. The agent client/server communication is handled using the Java Remote Method Invocation (RMI) system.

On the server side, both Java and C++ server components may be needed, depending upon the server functionality. In general, a server-side component is needed for each distinct server operating environment. For example, if the server supports a Legion environment, then a Legion component would be present. Communication between the Java and C++ server components is currently implemented either with Java Native Interface (JNI) calls, or using a standard socket interface, with Java sockets on the client and C++ sockets on the server. It is also possible to create CORBA interfaces to the agents. This would allow for interfaces other than sockets to be used for Java/C++ integration.

4.2 Example: Workflow Agent

An example of a SAF agent is the Workflow Agent, which provides access to the workflow information and the patent *Application Database,* which is the database of all currently active patent applications. The workflow information and application database are managed by the Texcel Information Manager, which provides C language APIs to access documents and workflow information within Texcel. To support the flexibility of having remote Java agents and clients access the workflow and Application Databases from anywhere in the metacomputer system, we had to extend the Texcel APIs by creating a socket interface for accessing the API functions. In addition to this Workflow Agent, several other agents have also been created in DOCT, to support various features mentioned earlier such as electronic "wrapping" of patent applications, archiving of audit trail information, and patent search.

5. Summary

This paper described the DOCT project and the USPTO patent and amendment processing application that was implemented using the prototype distributed, heterogeneous testbed provided by DOCT. The project required distributed development of software across geographically dispersed locations (California, Illinois, Virginia), and across multiple companies and organizations. The project has resulted in the construction of a distributed, heterogeneous, metacomputer testbed containing geographically dispersed computational and storage resources interconnected via high speed networks. Prototyping the USPTO application in this testbed demonstrated the feasibility and advantages of employing such a testbed for processing of complex documents.

Acknowledgements

We thank Larry Cogut and Pamela Rinehart, of the USPTO, for their constant guidance and help in this project. A number of other USPTO employees have also helped in defining the various applications requirements. We also acknowledge the help and support provided by Gary Koob of DARPA. The DOCT project is funded jointly by DARPA and the USPTO under Project F19628-96-C-0020.

References

[1] The High Performance Storage System (HPSS), http://www.sdsc.edu/HPSS/.

[2] *Agent Framework Requirements*, Technical Report G1-1, DOCT Project, SDSC, 10100 Hopkins Drive, La Jolla, CA 92093.

[3] The Storage Resource Broker, http://www.sdsc.edu/SRBhello/.

[4] *Evaluation of Queuing Systems*, Technical Report F1-1, DOCT Project, SDSC, 10100 Hopkins Drive, La Jolla, CA 92093.

[5] The Network Weather Service, http://www.cs.ucsd.edu/groups/hpcl/apples/nws.html.

[6] *DOCT Goal Security Architecture*, Technical Report E1-1, DOCT Project, SDSC, 10100 Hopkins Drive, La Jolla, CA 92093.

[7] Texcel, http://www.texcel.no.

[8] OpenText, http://www.opentext.com.

[9] Oracle, http://www.oracle.com

[10] DB2, http://www.software.ibm.com/is/sw-servers/database.

[11] Objectstore, http://www.objectstore.com

[12] The DB2/HPSS Integration project, http://www.sdsc.edu/MDAS.

[13] Kohonen Self Organizing Maps, Documentation for SOM_pak software tool, http://nucleus.hut.fi/nnrc/som_pak

[14] Learning Vector Quantization, Documentation for LVQ_pak software tool, http://nucleus.hut.fi/nnrc/lvq_pak

[15] Neural Network Research Center Home Page, Neural Network Research Center, http://nucleus.hut.fi/nnrc

[16] Informal Conversations with Michael Welge and Nina Mishra in 6/97, NCSA

[17] FreeWAIS, http://www.cnidr.org/welcom.html.

[18] Pretty Good Privacy, Inc. http://www.pgp.com/ and The International PGP http://www.pgpi.com/.

[19] "Secure Sockets Layer" http://www.netscape.com/newsref/ref/netscape-security.html.

[20] "The Relationship Between SSL And SHTTP", http://www.netscape.com/newsref/ref/netscape-security.html#rel.

[21] Secure Shell (SSH), http://www.cs.hut.fi/ssh/.

[22] "Kerberos: The Network Authentication Protocol", http://web.mit.edu/kerberos/www/.

[23] "Intrusion detection software," by Tom Perrine, *SDSC Gather/Scatter*, http://www.sdsc.edu/GatherScatter/GSspring96/perrine.html.

[24] "Applied Cryptography," by Bruce Schneier, 1997.

[25] "Question 165. What is X.509?," http://www.rsa.com/rsalabs/newfaq/q165.html.

[26] Computer Operations Audit and Security Technology (COAST), Purdue University, http://www.cs.purdue.edu/coast/coast-tools.html.

[27] The Pacific Institute of Computer Security (PICS), http://www.sdsc.edu/Security/.

[28] "Computer Oracle and Password System," ftp://ftp.cert.org/pub/tools/cops.

[29] Tiger, ftp://ftp.win.tue.nl/pub/security/tiger-2.2.3.tar.gz

[30] The lsof utility and ifstatus, ftp://vic.cc.purdue.edu/pub/tools/unix/lsof/.

[31] The ifstatus utility, ftp://coast.cs.purdue.edu/pub/tools/unix/ifstatus/ifstatus.tar.Z.

Scientific Data Archive at the Environmental Molecular Sciences Laboratory[1]

Daniel R. Adams, David M. Hansen, Kevin G. Walker
Pacific Northwest National Laboratory
PO Box 999 MSIN: K7-28
Richland, WA 99352
dan,dm_hansen,kg_walker@pnl.gov
tel +1-509-375-6579
fax +1-509-375-3641

John D. Gash
Lawrence Livermore National Laboratory
PO Box 808 MS L-103
Livermore, CA 94550
gash@pilgrim.llnl.gov
tel +1-510-422-0708
fax +1-510-423-8274

Abstract: The U.S. Department of Energy's Pacific Northwest National Laboratory (PNNL) has developed a Scientific Data Management system (SDM) that is an integral component of the DOE's Environmental Molecular Sciences Laboratory (EMSL). EMSL is a national scientific user facility that is supported by the DOE Office of Biological and Environmental Research. It is designed to focus Environmental Molecular Science on the puzzles and challenges of environmental restoration. The EMSL provides several major facilities to begin addressing these challenges, including the Molecular Sciences Computing Facility, a laser surface dynamics laboratory, a high field nuclear magnetic resonance laboratory, and a mass spectrometry laboratory. Each of these component laboratories will generate vast amounts of data that must be reliably stored and made available on demand for decades.

The SDM system addresses this challenge by providing a hierarchical storage management system to store data files, as well as a database that captures information about those files (i.e., metadata). The SDM system is a distributed client-server application that runs in a heterogeneous hardware, software, and networking environment. It provides EMSL scientists with 20 terabytes of near-line archival storage together with a metadata database describing the contents of the archive and the context within which archived files exist. This makes it possible to locate information based on what the data is about, not just the name someone gave to a file. The SDM system has been in use at the EMSL since July, 1997.

1. Introduction

EMSL is a collaborative research facility for investigating fundamental molecular processes relevant to the processing, storage, and remediation of hazardous waste and associated health effects at DOE sites. EMSL scientists are engaged in research ranging from *ab initio* molecular orbital calculations on supercomputers (such as the 512-node IBM SP machine) to molecular spectroscopy using one of a kind, ultra high frequency nuclear magnetic resonance instruments. All of these activities are focused on developing new science that can transform the environmental management and restoration processes at DOE nuclear facilities as well as other hazardous environments around the world.

The projected cost of environmental restoration at nuclear facilities is tremendous. In an article on environmental remediation at hazardous waste sites, *Science* magazine reported that "[the] projected costs of remediation of hazardous waste sites continue to mount and now range in the neighborhood of a trillion dollars or more, not counting legal fees. Even with such expenditures, it is unlikely that most sites will be restored to pristine conditions."[2] These estimates are based on current technologies and on reasonable extrapolations and extensions of these technologies within the framework of our present understanding of the underlying sciences. Consequently, new science that supports the development of new cleanup technologies is essential if hazardous waste sites are to be restored to benign states.

In doing this we expect scientists from institutions worldwide to generate several terabytes of data annually, much of which is suitable and appropriate for long term storage and retrieval. The formats of the data files that make up this research data will be as diverse as the scientists who create it. We anticipate binary data streams from proprietary data acquisition systems, word processing files of publications, results from standard numerical models, as well as one-of-a-kind files from custom programs written by visiting scientists and post-doctoral researchers. Some of the experimental data will be from experiments and measurements that are difficult or impossible to reproduce at future dates. Similarly, computational chemists will be storing results from long running (e.g., weeks) supercomputer jobs. Traditional techniques for tracking and managing this data – laboratory notebooks and collections of Zip disks – will simply not be sufficient for the task at hand. Furthermore, we need to make it possible for researchers to locate information that may be germane to current research years after that work was archived.

In order to satisfy these requirements, EMSL has procured a commercial hierarchical storage management system from EMASS and is developing a substantial set of software tools to manage interactions with the data archive. A key feature of the SDM software is the capture and maintenance of metadata about files that are archived. Whenever possible we have automated the process of extracting metadata from files as they are archived. SDM manages this metadata in an object-oriented database that is a component of the SDM software. The EMSL SDM system preserves the appearance of a standard Unix®-like file system, enabling users to locate and retrieve files based on this structure. The real long-term value SDM brings to the archive, however, is the ability to locate files based on their content (as captured in the metadata) and to quickly identify and explore related files. This content based search capability over terabytes of data is the most distinctive feature of the SDM system.

2. Our Work

The SDM system is a distributed client-server application that runs in a heterogeneous hardware, software, and networking environment. Supported client hardware ranges from desktop PC's running Windows 95 to UNIX workstations to an IBM SP supercomputer. It has been in production use at the EMSL since July, 1997. The SDM software is responsible for file transfer, metadata storage and search, and access control.

The SDM system is strictly file-oriented; data is stored and retrieved in units of whole files. We do not support retrieving segments or cross-sections of large datasets from the archive (i.e., no processing of the archived data is performed by the archive system). Metadata must be provided for each file that is archived at the time the file is archived. The SDM software automates much of the metadata extraction task in order to minimize the impact of this requirement on the users. Archived files cannot be directly accessed via ftp or as a network mounted file system (i.e., the archive file system is not available to users for direct manipulation); files can only be accessed via the SDM software. This makes it possible to ensure that metadata is provided for each file, and it precludes the use of the archive as a large, low-cost scratch disk. It also facilitates the implementation of access control on archived files.

The SDM software enforces access control restrictions that may be placed on archived files by the owners of the data. The SDM software is built using the Open Group's Distributed Computing Environment (DCE) software and thus has reliable and trustworthy Kerberos-based authentication credentials for users. SDM uses DCE Access Control Lists (ACL) and a reference monitor implementation to restrict connections to the servers. However, SDM does not use DCE ACLs to restrict access to archived files once a user has been authenticated to use the SDM software. Instead it maintains and enforces file and directory ACLs within the database component of SDM. The owners of archived data may restrict who has access to files and directories as well as constraining who may view the metadata associated with their files.

While it is possible to use DCE to encrypt the data in files as they are transferred to and from the archive, we have chosen not to do that. Also, in support of monitoring and controlling access to archived data, owners may request e-mail notification from the archive when specific access events occur (such as an attempt by an unauthorized user to retrieve a file).

2.1 Hardware Environment

The data archive is an EMASS hierarchical storage management system using their FileServ Hierarchical Storage Management (HSM) software product running on twoSilicon Graphics, Inc. (SGI) Challenge XL 10000 servers. The secondary storage is contained in an EMASS AML/E robot with a single quadro-tower and 4800 slots for IBM 3590 media. We presently have the tape robot loaded with 2000 tapes for a storage capacity of 20 TB. Each SGI server has 1 GB of RAM, 191 GB of RAID 3 disk cache, 32 GB of local disk, 2 OC-3 ATM interfaces, 2 HiPPI interfaces, and controls four IBM 3590 tape drives in the EMASS robot. This is illustrated in Figure 1.

The ATM network provides the backbone of the EMSL network. Individual workstations typically connect to the network via switched Ethernet connections and the Ethernet traffic is carried over ATM using LAN Emulation. The HiPPI interfaces are part of a private network which presently includes the archive, a 512 node IBM SP supercomputer (which has eight HiPPI interfaces) and SGI graphics and visualization workstations. All network traffic uses the TCP/IP protocols. When fully loaded the archive is capable of transferring data from network to disk at a sustained rate greater than 124 Mbytes/sec. Once the RAID disk caches are full, the tape drives become the bottleneck in throughput and the sustainable data transfer rate drops to 52 Mbytes/sec.

The SDM archive is accessed by a variety of computers. These include a variety of SGI, Sun, and IBM Unix workstations as well as numerous PCs running Windows 95 and Windows NT. A few of these machines are directly connected to the ATM backbone via OC-3 interfaces but the majority use standard 10 Mbit/sec Ethernet interfaces.

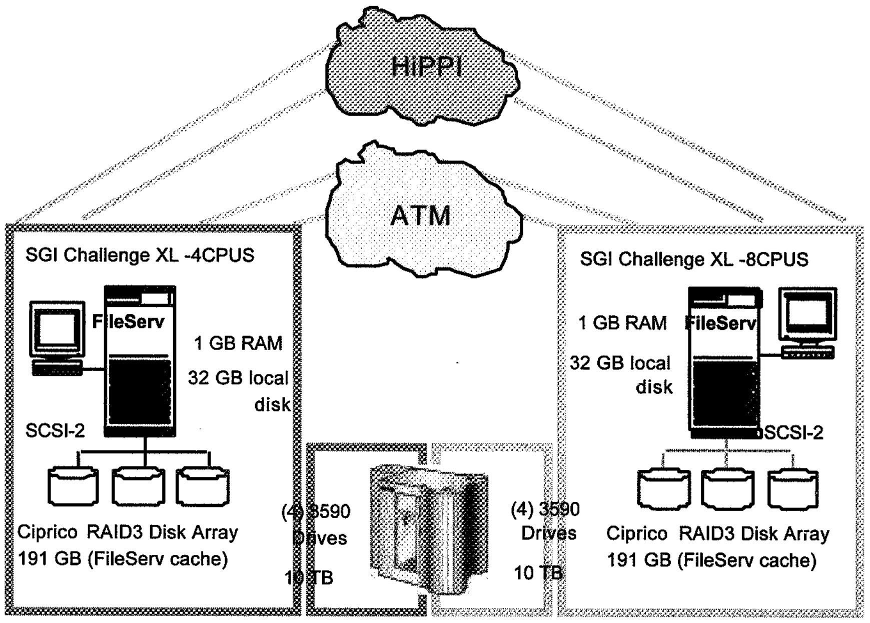


Figure 1. SDM Archive Hardware Environment

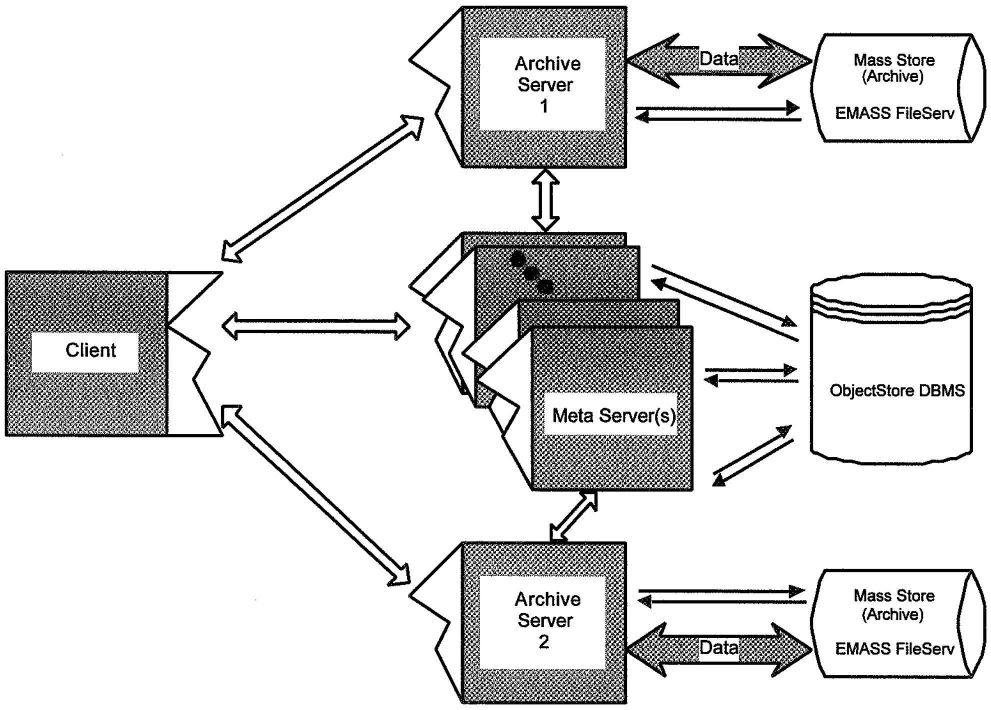


Figure 2. SDM Software Architecture

2.2 SDM Software Architecture

The SDM software is a client-server system built using DCE remote procedure calls. It has three major components: the client, Meta Server, and Archive Servers (see Figure 2). Each client workstation runs a copy of the SDM client that handles authentication of the user to the SDM system (through DCE) and processes user commands such as directory listings, storage, and retrieval.

When an SDM client is started, it makes a connection to the SDM Meta Server (actually one of possibly many Meta Servers, for reasons explained later). The Meta Server lies at the heart of the SDM software in that it provides most of the functionality of the system.

The Meta Server includes the client interface to an ObjectStore object oriented database management system manufactured by Object Design Inc. (ODI). The Meta Server mediates queries and interactions between the SDM client and the database.

One requirement of the SDM system is that the users see a single file system, without regard to the actual hardware configuration of the archive. In our case, the EMASS hardware provides two distinct Unix file systems, so the Meta Server – by way of the associated ObjectStore database – maps files into a single, unified file system. Users store files in their directories and the Meta Server allocates files to a specific server based on criteria such as balancing utilization of each SGI server.

While the ObjectStore server is multithreaded and can handle multiple concurrent queries, the same cannot be said for the ObjectStore client that is part of the Meta Server. Because we were unwilling to single-thread all interactions with the database in the Meta Server portion of the SDM software, we instead implemented multiple Meta Servers as multiple heavyweight processes. Each of these Meta Servers can safely connect to and interact with the ObjectStore database, thus allowing us to circumvent the single-threaded nature of the client. We use the DCE Cell Directory Service to transparently allocate client connections to Meta Servers. Unfortunately, even this was not sufficient to resolve all of our problems with multithreaded applications.

Even though we implemented multiple heavyweight Meta Servers, each server is still a DCE application and is thus intrinsically multithreaded. Initially, the Meta Servers and ObjectStore database were hosted on the more powerful of the SGI Challenge servers. However the ObjectStore client for SGI was not able to work within a DCE application due to conflicts in handling Unix signals. ODI provided us with a patched client for SGI that worked around these issues but we eventually moved the Meta Servers and ObjectStore database from the SGI to a Sun Solaris platform where DCE is better supported and accommodated.

An SDM Archive Server is running on each of the SGI servers. The Archive Servers are responsible for actually reading and writing to the archive file systems. Metadata is streamed to the Meta Server and data is streamed from(to) the SDM client to(from) an Archive Server by using DCE pipes. The Archive Server handles all interactions with the FileServ software through Application Programming Interface calls or separate executables for operations such as staging files from tape to disk for subsequent retrieval.

Because archive operations involve processes running on (typically) three different machines, we needed some sort of distributed transaction manager to keep the SDM database, archive file systems, and current operations synchronized. Commercial products such as Encina were either not available on all the platforms we support or their costs were prohibitive. The SDM software manages distributed transactions through a "contract" mechanism. A contract represents an operation and its associated components (files) and the status of the operation on each component. We take advantage of the atomicity property of the database to maintain current state and employ a conservative, fail-safe, approach to errors.

For example, when the SDM client requests to store a directory hierarchy to the archive, it contacts the Meta Server with this request and the Meta Server creates a contract in the database with information regarding the files to be stored. The Meta Server sends the contract to one of the Archive Servers, informs the client of the contract identifier, and directs the client to the Archive Server to actually transfer the data. When the client contacts the Archive Server it presents the contract identifier as part of the initial negotiation. The Archive Server verifies the contract with the Meta Server and begins transferring the first file. After each file is transferred to the archive, the Archive Server sends a status update to the Meta Server that reports on the status of the transfer of that particular file. The Meta Server notifies the database and, on successful completion of the transfer, the database makes a permanent entry for the file and marks that portion of the contract as having been completed.

Thus, if there is a network or hardware failure during a file transfer the database will never contain an entry for a file that was not fully and successfully transferred. Because the contracts are stored in the database, a partially completed contract will not be lost if there is a hardware failure. The SDM software can identify partially completed contracts and roll back any file operations that might have been in progress at the time of the failure. Throughout the design and implementation of the contracts, the central principle was to make sure that we never asserted that a file was successfully stored in the archive when in fact it was not.

2.3 Metadata

The metadata that is captured in the SDM database is the key to keeping the archived data accessible and useful by researchers[3]. We have partitioned the metadata into three categories: file oriented, content oriented, and context oriented. File oriented metadata contains information about the file system objects in the archive such as:

- name of file as seen by users
- name of file in the archive
- original owner
- current owner
- size
- date stored
- date of last access
- access history
- access restrictions
- notification expectations.

Content oriented metadata is typically keyword=value pairs that have been extracted from files as they are stored into the archive. Examples of this type of metadata would be:

Comment	=	This spectral data was gathered by the light of the silvery moon using dew drops on gossamer threads.
Operator	=	Dave Dataman
Calibration date	=	1/21/98
Pressure	=	3 torr
Output Energy	=	3 mJoule
Peak power	=	2 terawatts
Instrument	=	12 tesla ICR

Finally, context metadata is information about how a given file or directory is related to others. For example a directory might contain data files from a certain experiment. If these data had been subjected to analysis and quality control checks and the results of these computations were also stored in the archive, then a user with access to the data would be able to establish a contextual link between two archive objects (i.e., files or directories). Later, if the results of the analysis are used as part of a refereed publication, that publication can also be archived and a link established between it and the analysis data. A user who accessed any one of these components (possibly as a result of searching for files related to dew drops) would also be led toward the other parts of the chain. Other relationships are possible and the establishment of relationships is bounded only by the willingness of users to document the relationships or of software to discover and establish them.

Somewhere between the content and context metadata is the notion of annotative metadata. Users may also place the electronic equivalent of "yellow sticky notes" on archive objects. These are simply text records that are associated with archive objects. An example of this would be a warning that the calibration of an instrument used in an archived experiment is suspect. To promote collaboration, a user is usually allowed to annotate any file he can read. However, to guard against abuse, every user is not necessarily granted permission to read annotations placed on files. Thus one user could annotate another's results and say that they are totally bogus, but everyone else who viewed those files would not see the accusation.

Due to the variety of users and academic disciplines represented, we do not maintain a rigid ontology of keywords. Similarly, because the types and formats of the files are not constrained, we have adopted a very promiscuous approach to handling metadata. Specifically, we take whatever the users say is a keyword and track it as such. Software that is logically part of the SDM client attempts to recognize file formats and automatically extract keyword=value pairs from the files as they are streamed to the archive. This approach definitely has significant drawbacks – such as spelling errors, namespace collisions, redundancy – however it appears to be the most workable solution for accommodating the variety and variability we expect in the data. The database schema we use for tracking the file system objects and their metadata is illustrated in Figure 3.

We have attempted to turn this promiscuous metadata approach into an asset by enhancing techniques available for searching the metadata. In particular we "documentize" the metadata wherever possible – turning it into text streams. Throughout the SDM database, where we have textual fields, we build searchable indices using text indexing and search tools made byVerity. Users can query the archive with keyword searches similar to those used for web searches and these searches look through the metadata in finding matches.

3. Related Work

Lawrence Livermore National Laboratory's Intelligent Archive[4,5] (IA) project is addressing similar problems although their approach differs from ours in the association of metadata with the archived data. IA maintains metadata in a database much as our SDM system does, however the acquisition of the metadata is not directly tied to the archiving of files, neither is the submission of metadata a mandatory requirement for use of their archive. Their approach is similar in that they have distinct applications that attempt to automate the extraction of metadata from user files.

The San Diego Supercomputing Center's Massive Data Analysis Systems[6,7] (MDAS) is another project with related but much broader goals. Similar to SDM they are coupling descriptive metadata with files to facilitate the location and retrieval of data based on content as opposed to file system semantics. The MDAS project scope goes beyond this to address the issues of making the managed data self-describing and coupling computational methods with the data and metadata.

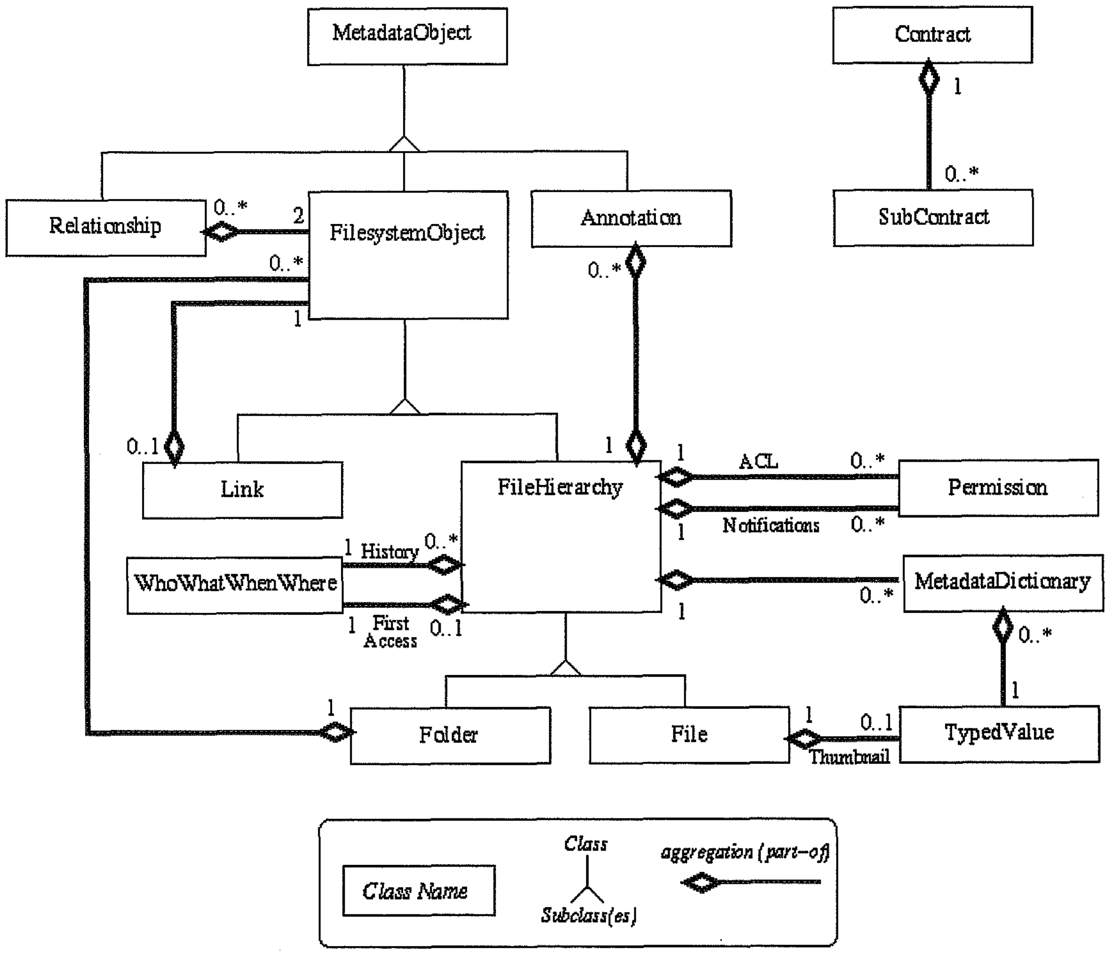


Figure 3. SDM Database Schema

4. Future Work

The SDM system has only recently been deployed so we expect to gather quite a few suggestions from our users as they begin to take advantage of the system. In the near term we will be focusing on operational issues such as increasing the number of file types from which metadata can be automatically extracted, enhancing the tools for performing metadata searches and queries, improving data transfer rates, and enhancing graphical interfaces to the SDM archive. We also have ongoing research activities to develop techniques for identifying and exploiting relationships between distinct archive objects and developing tools to exploit these relationships.

5. Conclusions

We have developed a data archive which couples metadata (file-oriented, content, and contextual) with files as they are archived. The software couples DCE client-server programs with an object oriented database. It insulates users from needing detailed knowledge of the archive structure in order to use the system and it captures and maintains metadata about every file that is archived. Using our Scientific Data Management software it is possible to locate relevant information in the archive without having to know who created or owns the information or what file naming conventions were used when the data was archived. By using annotative metadata and establishing relationships between elements of the archive, the system offers a capability for capturing information about the context in which data exists in addition to simply documenting the contents of the archive.

---

[1] This work was funded by the Environmental Molecular Sciences Laboratory construction project at Pacific Northwest National Laboratory (PNNL). PNNL is a multiprogram national laboratory operated by the Battelle Memorial Institute for the U.S. Department of Energy under contract DE-AC06-76RLO 1830.
[2] "Remediation of Hazardous Waste Sites", *Science*, Volume 255, number 5047, page 901, February 21, 1992
[3] David M. Hansen and Daniel R. Adams. *A Database Approach to Data Archive Management.* In Proceedings of the First IEEE Metadata Conference, Silver Spring, MD, April 1996. IEEE Computer Society Mass Storage Systems and Technology Technical Committee, IEEE Computer Society Press.
[4] http://www.llnl.gov/ia/
[5] Springmeyer, R., Nancy Werner, and Jeffery Long, *Mining Scientific Data Archives Through Metadata Generation.* In Proceedings of the First IEEE Metadata Conference, Silver Spring, Maryland, April 1996. IEEE Computer Society Mass Storage Systems and Technology Technical Committee, IEEE Computer Society Press.
[6] http://www.sdsc.edu/MDAS/
[7] C. Baru, R. Frost, J. Lopez, R. Marciano, R. Moore, A. Rajasekar, M. Wan; *Metadata Design for a Massive Data Analysis System.* In Proceedings of CASCON '96 (11/96)

Page intentionally left blank

Virtual Storage Manager

Stephen H. Blendermann
Storage Technology Corporation
Louisville, Colorado 80028-6202
steve_blendermann@stortek.com
Voice: +1 303 673-3184
Fax: +1 303 661-8462

Dennis F. Reed
Storage Technology Corporation
Louisville, Colorado 80028-6202
dennis_reed@stortek.com
Voice: +1 303 673-6491
Fax: +1 303 661-8462

Abstract: Storage Technology Corporation (StorageTek®) has developed Virtual Storage Manager™ (VSM) as a solution to the problems of inefficient use of tape media and tape transports. VSM solves the media use problem by stacking multiple user tape volumes onto a single, physical tape cartridge. The stacking operation occurs transparent to the user and the actual movement of data to accomplish volume stacking occurs outboard of host systems being serviced. VSM solves the tape transport use problem by emulating more tape transports than physically exist. VSM satisfies peak concurrent transport demand without requiring additional transports that would not be needed or used at other times. Physical tape transports are attached to an Automated Cartridge System (ACS). The solution is comprised of hardware, microcode, host software and PC software.

History

In the late 1970s, StorageTek developed an outboard Virtual Storage System (VSS). This system consisted of host-based software, a 370/138 class processor running special software known as the Virtual Control Processor (VCP), a Channel Adapter (CA) with built-in data compression for connecting the VCP to the host, and 3350-class DASD and 3420-class tape devices connected to the VCP. The VSS host software system controlled allocation and provided the capability to create virtual volumes. The data was compressed in the CA, re-blocked in the VCP's memory and then written to the attached DASD. The data was automatically backed up to tape and migrated and recalled as necessary.

The technology available at the time did not allow this product to be brought to the commercial market. However, in 1984, a patent[1] was issued to Storage Technology for an outboard storage system that used virtual volumes, built-in data compression, and automatically managed disk space by moving data between the disks and tape.

With the advent of automated tape libraries and fault tolerant disk subsystems developed by StorageTek, the technology is now available to bring a virtual storage system to the commercial market. At the same time, tape cartridge capacities have increased without a corresponding method to utilize the full capacity of the cartridges. Virtual Storage Manager provides the ability for applications to make use of these cartridges.

In 1995, a small team was chartered with building a prototype of a VSS to demonstrate the feasibility of the concept. That team modified Iceberg microcode and HSC software to produce a working system that presented a 3490E tape transport image to the host. The host code managed the disk buffer and migrated/recalled virtual volumes to/from tape. In the prototype, manual tape transports were used as StorageTek had been automatically mounting cartridges since 1987 and it was not necessary to prove that could be done.

In conjunction with the prototype, requirements for virtual storage system were gathered from several customers around the world. Those requirements were the basis for the initial functions of the system. These functions were presented at a Customer Advisory Board for validation and further refined to ensure that the product met customer needs.

On October 13, 1997, Storage Technology announced the Virtual Storage Manager.

Solution - Overview

VSM is comprised of hardware, microcode, host software and PC software. VSM hardware and microcode present the image of 64 IBM 3490E tape drives to one or more ESCON-attached MVS-based hosts. This hardware includes RAID-6 disk storage that serves as a buffer for holding user-created (virtual) tape volumes (VTV).

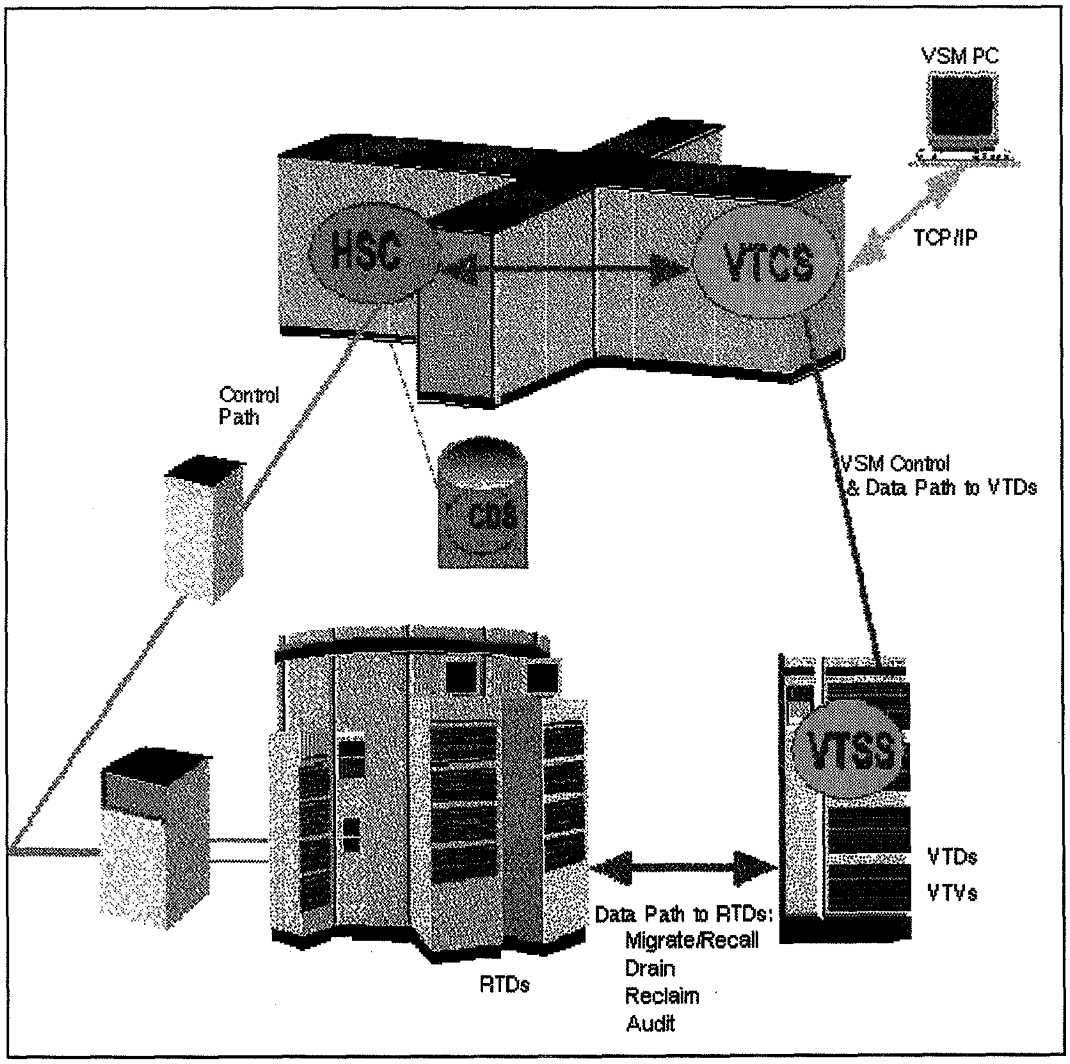

VSM PC
HSC
VTCS
TCP/IP
Control Path
CDS
VSM Control & Data Path to VTDs
VTSS
VTDs
VTVs
Data Path to RTDs: Migrate/Recall Drain Reclaim Audit
RTDs

Figure 1. VSM Solution

The VSM host software directs tape jobs to the VSM hardware, influences the allocation of the virtual 3490E tape drives (VTD), and instructs the VSM hardware to perform mounts of virtual tape volumes on the appropriate virtual tape drive (VTD). The VSM host software manages the disk buffer, and instructs the VSM hardware to migrate virtual tape volumes to real tape cartridges, stacking several VTVs onto a single, physical (multi-volume) cartridge (MVC). When needed to satisfy a tape job, VSM host software automatically instructs the VSM hardware to recall a VTV from a MVC back into the VSM hardware disk buffer. VSM host software maintains the locations of all VTVs, whether resident on a VSM hardware disk buffer or a MVC. All potential MVC contention is mitigated by VSM host software. All real tape drives (RTD) used by VSM hardware to satisfy VTV migrate and recall actions, are library-attached to fully automate the solution. VSM PC software provides a Windows 95 or NT platform with a graphical user interface for administration and reporting of VSM functions, including installation, configuration and maintenance

The features of VSM include:

- Efficient utilization of tape media by stacking multiple (user) VTVs onto a physical MVC.
- Emulation of multiple, host-compatible tape control units and drives, backed by a high-performance RAID-6 disk buffer, to satisfy peak tape drive allocation as well as deliver superior VTV read access when resident in a disk buffer.
- Efficient data throughput to minimize the number of physical tape drives needed to satisfy a larger number of emulated tape drives.
- Transparent user access to tape data, regardless of physical data location, either on a disk buffer or a MVC.
- Mapping of user VTV data from emulated, host-compatible devices to other storage device technologies.
- Support for multiple physical cartridge and RTD technologies.
- Outboard (of host) movement of compressed VTV data for migration and recall.
- Increase tape automation capacity. All RTDs are library-attached. The library may be shared with non-VSM activity.

VSM is a natural extension of StorageTek's core technologies: tape automation and device virtualization. The combination of these technologies yields a solution that can dramatically lower the cost of storage and storage management. VSM takes advantage of current, underutilized device and tape cartridge technologies (investment protection), as well as providing a seamless technology migration path for future device and media technologies. VSM further increases the effectiveness of existing tape automation, while saving on precious resources such as floor space, tape racks and device power/cooling.

VSM Hardware

VSM hardware consists of one or more virtual tape subsystems (VTSS™), which are built on the Iceberg® virtual storage platform. Each VTSS:

- Emulates either 32 or 64 virtual 3490E tape drives.
- Emulates standard 3480 cartridges (400 MB capacity after compression). A single VTSS can support up to 100,000 VTVs resident in the disk buffer.

- Supports either 8 or 16 ESCON ports which are configured as host-attached control unit ports or RTD-attached I/O channels.
- Includes a RAID-6 disk buffer with effective user data capacities ranging from 180 GB to over 930 GB.
- Attaches 2 to 8 RTDs, which must be library-attached on a single StorageTek ACS, and can be either TimberLines or RedWoods or a mix of the two.
- Moves data between the VTSS disk buffer and RTDs without using host processing cycles or channel bandwidth.
- Supports all TimberLine® and RedWood® media types, and can be mixed.
- Provides the throughput performance, fault-tolerance and non-disruptive hardware serviceability of the latest generation of Iceberg hardware and microcode.

VSM tape device virtualization, disk buffer and outboard data movement are accomplished on an Iceberg virtual disk array platform with new microcode. This platform provides a robust engine, both in performance and durability, to meet the demands of large tape data processing centers. A VSM solution consists of host software and one or more virtual tape subsystems (Iceberg).

VSM Host Software

Virtual Tape Control Software (VTCS™) is the VSM host software that works as an extension to StorageTek's ACS host software component (HSC). VSM virtual tape processing may be fully integrated with other automated and manual tape processes, including sharing access to ACS robotics with non-VSM activity. VTCS functionality:

- Extends the current HSC database to track all key information concerning VTVs, RTDs, MVCs and VTSSs. This includes maintaining all configuration and operating parameters for the VSM solution.
- Manages the location of all VTVs in the VSM solution.
- Influences MVS allocation of VTDs to satisfy tape job requirements.
- Manages migration and recall of VTVs to/from RTDs
  - ⇒ Insures adequate VTSS disk buffer space available to accommodate new data
  - ⇒ Issues robotic commands via HSC
  - ⇒ Issues migrate/recall orders to appropriate VTSS
  - ⇒ Selects candidate VTVs for automatic migration based on LRU and size.
  - ⇒ Optionally migrates VTVs to 2 MVCs to protect migrated VTVs from media failure
- Manages MVC space reclamation as migrated VTVs expire or are updated.
- Manages recovery of lost HSC control dataset (CDS) information if needed. The CDS itself has multiple levels of data protection as currently provided by HSC.
- Manages recovery from RTD or MVC media errors reported by a VTSS. This includes logging of device and media failures.
- Provides a standalone recovery utility to allow retrieving VTVs from MVCs without benefit of a VTSS.

VTCS requires OS/390 1.2 with Open Edition.

VSM PC Software

VSM PC software provides a graphical user interface on a Windows 95 or NT platform to administer the VSM solution. VSM PC software communicates with VTCS over TCP/IP to the MVS host where VTCS is resident. VSM PC software on a single PC can connect to any of the systems VTCS is installed on. Multiple copies of VSM PC software, on one or more PCs, can be active at the same time. VSM PC software functionality includes:

- Setup of VSM configuration parameters.
  - ⇒ Automatic migration threshold as percent of disk buffer fullness
  - ⇒ Generation of new HSC control statements and parameters
  - ⇒ Definition of MVC cartridge pools and VTV volser ranges
  - ⇒ Definition of maximum depth for stacking VTVs to a MVC
- VSM usage reporting
- Demand migration requests.

VSM Configurations

The VSM solution consists of host (MVS) software, PC software, one or more VTSSs and library-attached tape drives.

A VTSS can be connected to a maximum of 15 MVS hosts systems. Hosts may be attached via ESCON Directors. Each VTSS can be attached to a minimum of two tape drives (RTD) up to a maximum of eight tape drives. If an ESCON Director is used between a VTSS and a RTD, it must be a static connection. A dynamic connection between a VTSS and RTD is not supported in VSM release 1.0. RTDs can be statically shared between MVS hosts and VSM, i.e. a RTD varied online to VSM must be varied offline to MVS, and vice versa.

The VSM solution can support multiple VTSSs shared by multiple MVS hosts.

The following are examples of VSM configurations. These examples show simple configurations even though complex configurations are supported. These examples show one or two RTDs, but actual configurations can have up to eight RTDs attached to each VTSS. Each VTSS must have a minimum of two library-attached TimberLine or RedWood tape drives. Each example shows a single LSM, but multiple LSM configurations are supported. All RTD's connected to a specific VTSS must be served by a single ACS. Dual-ported RTDs can be shared by two VTSSs. An ACS can be shared by the VSM solution and non-VSM MVS usage. A LSM can contain MVC cartridges and non-MVC cartridges.

The VSM solution supports the following LSMs via the existing HSC support for these libraries: 4410, 9310 and 9360.

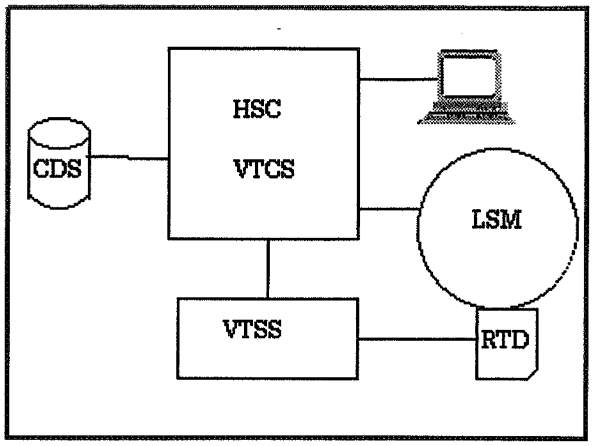


Figure 2. Single Host - Single VTSS

In Figure 2, one VTSS is connected to one host. The VTCS on that host manages all activity of the VTSS.

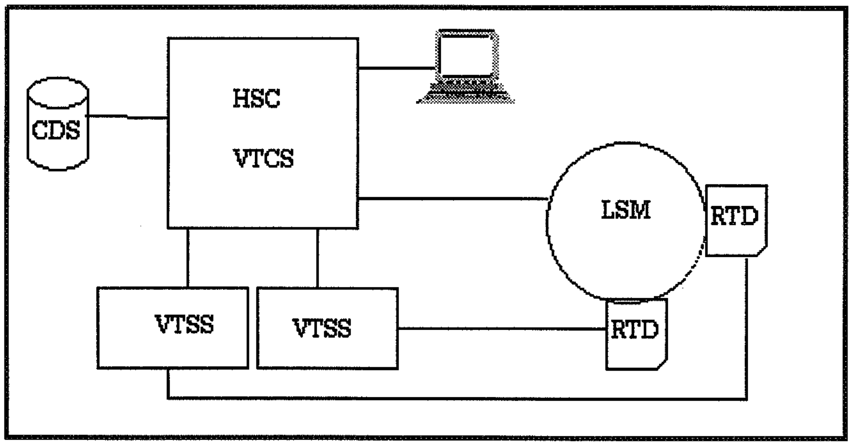


Figure 3. Single Host - Multiple VTSSs

In Figure 3, two VTSSs are connected to one host. The VTCS on that host controls both VTSSs.

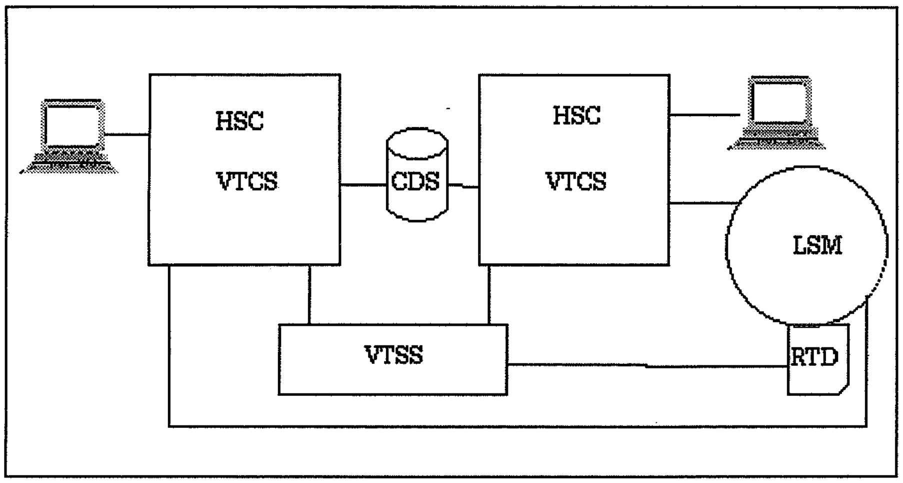


Figure 4. Multiple Hosts - Single VTSS

In figure 4, two hosts share the management of a single VTSS. Each host has a HSC and VTCS, and each host is responsible for managing the VTV requests, including recalls. VTV migration is controlled by the host that first detects the fullness threshold of the VTSS has been exceeded. Up to fifteen hosts can share a VTSS in this configuration.

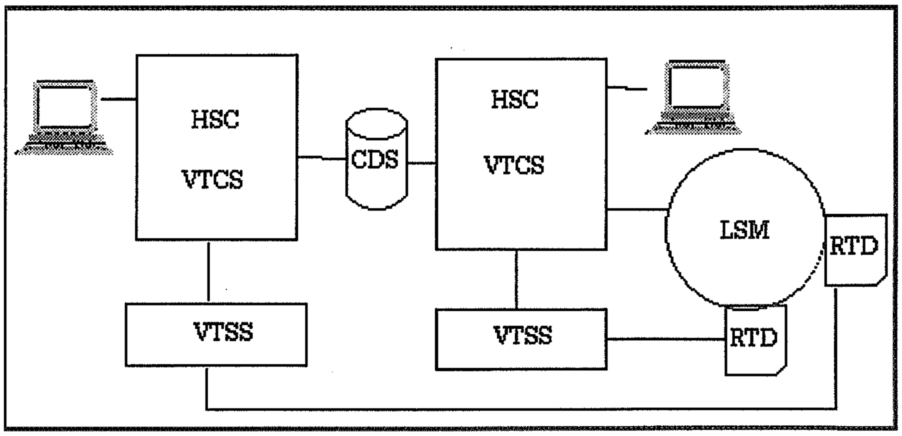


Figure 5. Multiple Hosts - Multiple VTSSs

In figure 5, each host manages its attached VTSS, as in the figure 2 example. If both hosts were attached to both VTSSs, the system would operate like the figure 4 example. Each host would be responsible for its VTV activity, and either host could handle VTV migration on either VTSS.

In this example, the hosts are shown as having separate connections to each VTSS. Each host could be connected to each VTSS and the VTDs shared by those hosts.

Performance

By emulating 64 virtual 3490E tape drives, presenting virtual tape volumes to tape users and caching VTV data on a large disk buffer, VSM optimizes:

- Access time. Virtual mounts occur in less than a second.
- Data throughput. Data is compressed at the Iceberg I/O channel prior to being stored in cache memory, the disk buffer, or migrated to physical tape. Read requests for VTVs resident in the disk buffer are satisfied immediately at disk speeds.
- Physical media utilization. VSM host software stacks multiple VTVs on a single MVC to maximize media utilization, and reduce the number of physical mounts required by the solution.
- Real tape transport requirements and utilization. VSM uses fewer physical drives to satisfy the I/O load handled by the VTDs. VSM requires fewer physical tape mounts to accommodate a larger number of fast, virtual mounts.

Virtual scratch tape mounts are typically satisfied in less than 1 second. The VTSS can support write data rate of 9 MB per second on each of the 8 VTSS data paths at the same time. When a tape mount to read an existing VTV is received, and the data is still resident on a VTSS disk buffer, the mount is satisfied at in less than one second. A VTSS can support read data rate of 10 MB per second on each of the 8 VTSS data paths at the same time.

Conclusion

VSM is another significant advance in StorageTek's successful Nearline® product series. VSM extends the benefits of the unique Iceberg virtualization platform, adds state-of-the-art host software, and dramatically increases the useful storage capacity of existing customer tape libraries. VSM offers equally dramatic improvements in tape job performance while reducing tape management overhead. The benefits that StorageTek Nearline users enjoy today, including leading-edge robotic and tape device technology, are extended for years to come. Seamless integration of future advances in these technology areas is assured. VSM eliminates the constraints of conventional tape subsystems and allows users to fully exploit their existing storage potential.

Futures

VSM hardware and microcode will continue to track and benefit from updates to base Iceberg hardware and microcode. Updates may include:

- Non-disruptive microcode update capability.
- Increased disk buffer capacities.
- Additional channel attachments to non-MVS host platforms.
- Additional RTD device types and associated MVC media types.

VSM Host Software updates may include:

- Full Sysplex support.
- MVC export from VSM and import to VSM.
- Dynamic sharing of RTDs between VSM and MVS.
- User-defined options for controlling how VTVs are grouped on MVCs.
- Support for non-MVS client systems.

References

[1] White, Barry B., United States Patent Number 4,467,421 "VIRTUAL STORAGE SYSTEM AND METHOD", issued August 21, 1984.

Trademarks

Page intentionally left blank

Petabyte Mass Memory System Using The Newell Opticel - Part 2

Chester W. Newell
Primelink Technologies Inc.
#208, Advanced Technology Centre
9650 - 20 Avenue
Edmonton, Alberta, Canada T6N 1G1
Email: primelink@atc.edmonton.ab.ca
Phone: 403-440-0111
FAX 403-440-0110

Abstract: A random access system for storage of up to a Petabyte was described by the author in a paper given at the Fourth NASA Goddard Conference on Mass Storage Systems and Technologies, March 28, 1995. This system is called the "Opticel" system.

The Opticel has been modeled in feasibility building blocks of a tape drive and associated optics. The Opticel cartridge geometry is compatible with the standard magnetic DLT cartridge, and can be interfaced with a standard robotic library system to provide open-ended expandability into a Petabyte system in nine square feet of floor space. All essential parameters described in the earlier paper have been verified in subsystems.

The currently proposed Opticel system is being developed to employ 19 mm WORM optical tape and sixteen parallel red laser heads in a 5-1/4-inch half-high drive. A comparison of the Opticel in the proposed Petabyte System is made, using Opticel drives in two typical DLT-compatible multiple cartridge systems as reference (see Table 1).

A parallel advanced development is now beginning using magnetic erasable metal-evaporated tape and thin-film heads. Studies to date indicate all parameters of the optical system will be equaled or exceeded with this combination.

Review

In Part 1[1] the author first proposed the "Opticel System" considering two alternate proprietary optical tape-handling principles in a dust-sealed cartridge. The system combined phase-change WORM optical tape with conventional optical heads employing IR lasers and state-of-the-art data encoding/decoding methods. One version of the Opticel contemplated using a simple endless tape loop in a "scramble bin.". An alternate higher-capacity version was discussed, based on the proprietary NII™ Principle.[2]

The Petabyte System would require fewer than 1000 Opticels employing the simpler endless-loop version with 75 meters of tape in a conventional stack loader, with 89 columns x 125 rows, in approximately 45 square feet of floor space. At the proposed search speed of 25 meters/sec., the maximum search time of the loop would be 3 sec.

Results to Date

The Optical Head Assembly

A state-of-the-art optical head using 680 nm wavelength IR and holographic optics with numerical aperture of 0.6 was obtained, giving a spot size at the 50% density level on the Gaussian energy curve of

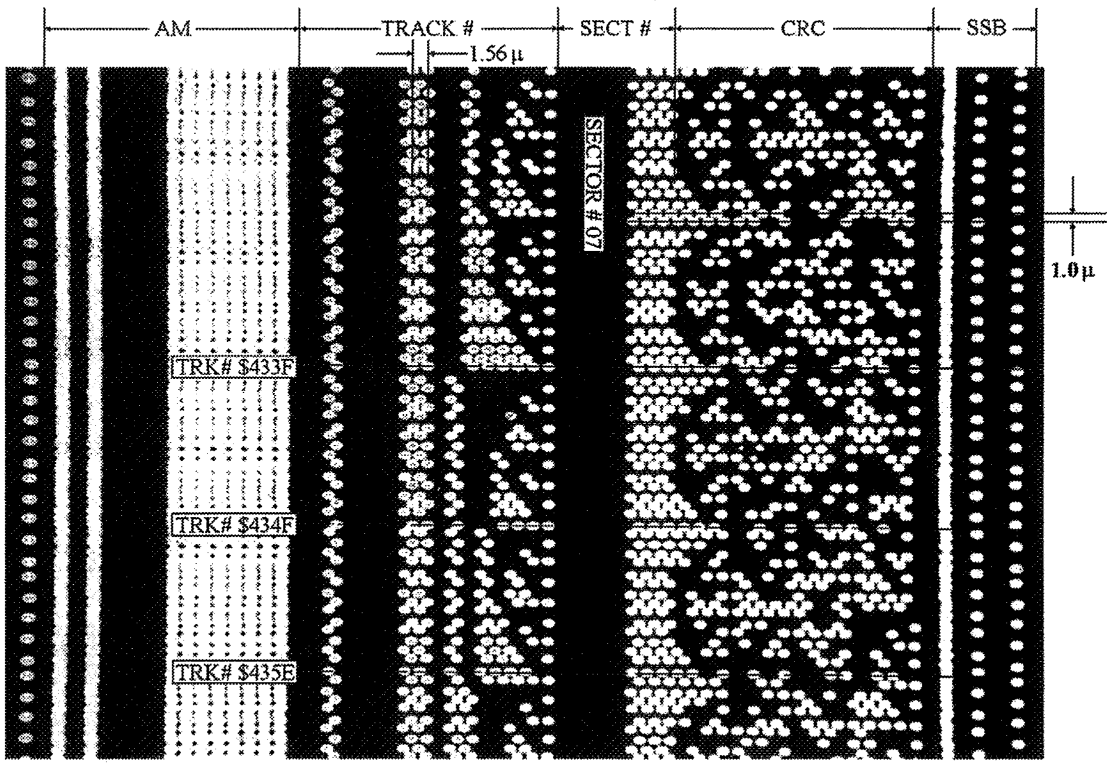
$$d_{pit} = \frac{680 \times 10^{-9}}{2 \times .6} = 0.57 \text{ micron} \qquad (1)$$

This head was tested in an optical test stand at various energy levels, to validate that the fundamentals assumed in Part 1 of this paper are practical.

i.e. track pitch < 0.87μ
cell pitch < 1.56μ
mark edge resolution < $0.25\ \mu$

These dimensions were verified in the test stand--see microphotograph Figure 1.

Figure 1
FORMATTED OPTICAL TAPE , IR LASER, N_A=0.6 , CELL PITCH=1.56µm

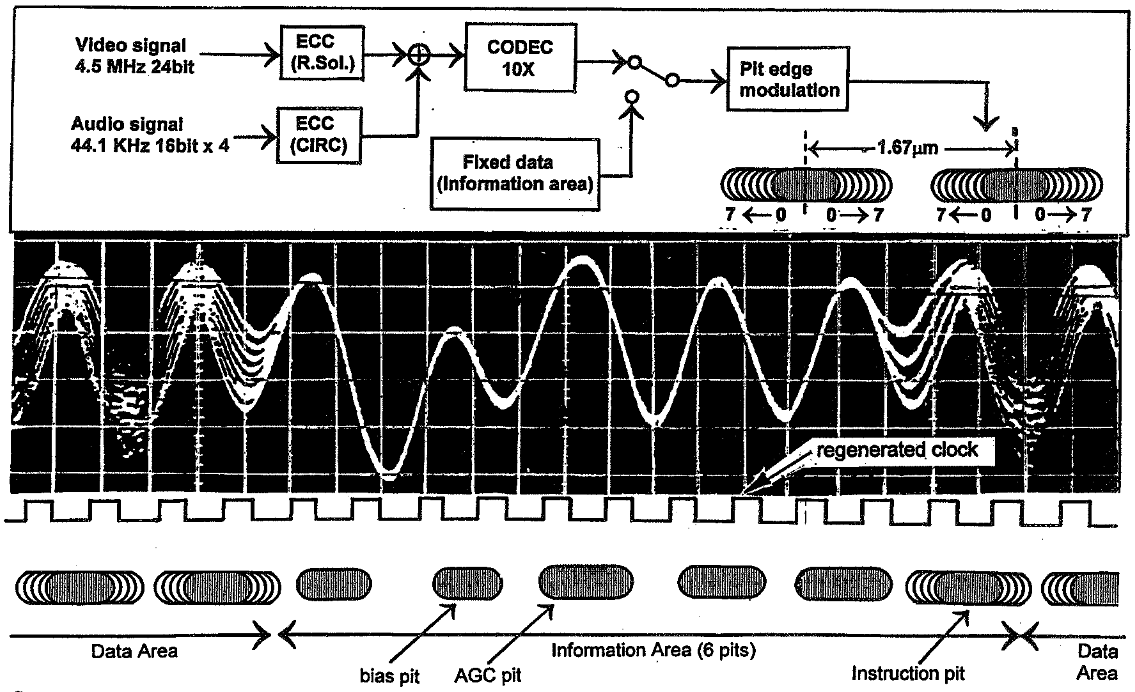


ADDRESS MARK (AM), TRACK#, SECTOR#, CYCLIC REDUNDANCY CHECK (CRC) & SAMPLE SERVO BYTE (SSB)
READ POWER = 0.24 mW cw (DAC SETTING = $7C), WRITE POWER = 3.2 mW cw (DAC SETTING = $32), SURFACE VEL = 10.05 m/s
TK4300_B.PSD DIMENSIONS IN MICRONS (µ), TRACK & SECTOR #'s IN HEXADECIMAL

The Encoding/Decoding Method

Using the proprietary Mark Edge Recording Format developed by SOCS Research and licensed to Primelink, up to eight edge pairs have been accomplished by SOCS and their other licensees for lower bit-transfer rates--see Figure 2.

Fig. 2. Primelink Mark Edge Recording Format



Source: Sony Corp.

Primelink's studies, in cooperation with our vendors, predict that with custom ASICs, at least six edge pairs at our transfer rate should be achievable. Remaining building blocks are being completed in Part 3 of the program to verify that the following target parameters are feasible:

- Servo-controlled light intensity modulation (LIM) of a red recording laser, to be used in the introductory product, will enable a recorded cell width of 0.25µ.
- Cell length of 1.56µ is adequate for at least six mark edge pairs.
- 1.5 raw bits can be recorded on each edge pair, using RLL 2,7 code.
- 40% total overhead for headers, intertrack crosstalk, and tape SNR is projected to yield a BER of 1 x $10^{-15.}$

From these parameters in combination we have the potential of providing a user bit area of

$$A_{bit} = 7.2 \times 10^{-14} \text{ meters}^2 \qquad (2)$$

The Optical Recording Medium

An ultra-thin tape substrate concept is being developed by Primelink to demonstrate a practical tape less than 1.0 micron thick[2]. This would support bending (shear) stresses well within an elastic limit which would allow a dramatic increase in the loop length to as much as 1000 meters in the "stretch" version of the "Scramble Bin" Opticel described in Part 1 of this paper.[1] Such a loop, 1/2" wide, would give a capacity of:

$$C_{user} = \frac{1000 \times 12.5 \times 10^{-3}}{7.2 \times 10^{-14} \times 8} = 21\ \text{TB} \tag{3}$$

By parking the loop at its mid-point, the statistical average search time to any file within the bin would be

$$\text{Time}_{search} = \frac{1000}{25 \times 6} = 6.\dot{6}\ \text{sec.} \tag{4}$$

60 such Opticels would provide a Petabyte. Although this performance would be adequate, it was decided during Part 2 of the program that the most certain and fastest route to the Petabyte would be via the NII™ Principle.

The NII™ Principle[3]

An NII™ cartridge-drive system called the "Opticel A" was developed to be form-factor compatible with standard Digital Linear Tape (DLT) cartridges:

i.e. L = 4.28 in. W = 4.28 in. H = 1.00 in.

This cartridge incorporates the NII™ reel-to-reel isoelastic drive principle with fiberglass belts and 19mm tape. It will plug and play in a stand-alone 5-1/4" half-height Opticel drive or interface with standard DLT stack loaders (see Figures 3-6).

The Opticel A uses 1/2-mil optical tape, storing 218 meters. With the user bit area of equation (2), this would give a capacity of

$$C_{user} = \frac{218 \times 19 \times 10^{-3}}{7.2 \times 10^{-14} \times 8} = 7\ \text{TB} \tag{3'}$$

The drive can accelerate the Opticel A to 25 meters/sec. in 12 millisec. Mid-point parking gives a statistical average search time to any file within an Opticel A in any one drive of:

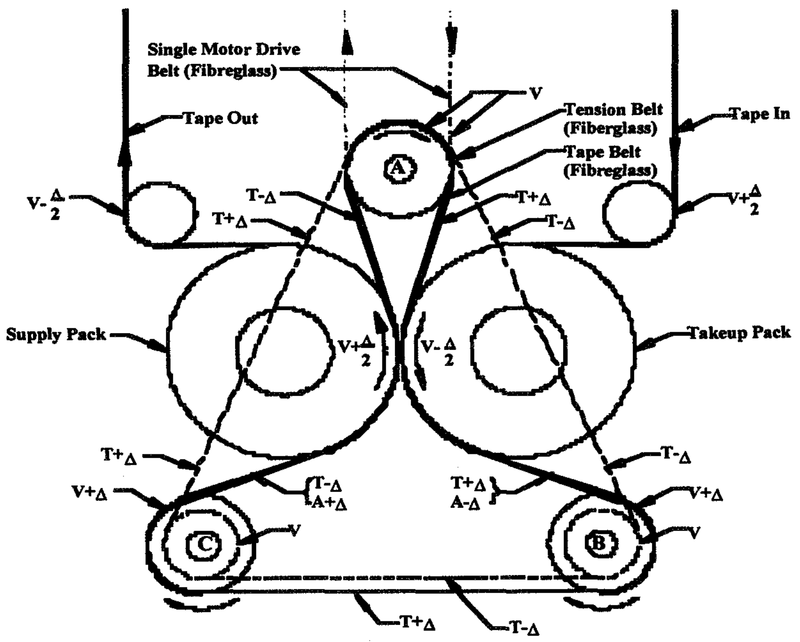
$$\text{time}_{search} = .012 + \frac{218}{25 \times 6} \approx 1.5\ \text{sec.} \tag{5}$$

Figure 3

NII™ OPTICEL TAPE HANDLING PRINCIPLE

Simplifying Assumptions:

- Pulleys A, B, C driven by fiberglass belts at constant angular velocity.
  (B and C diameter ratios exaggerated).
- Path Stiffness: Tension Belt >> tape belt >> tape path



At System Equilibrium:

- $\pm\,\Delta$: Increase / decrease in tension T, Area A, Velocity V.
- Mass flow in belt spans = A x V = Constant (no belt slippage)
  $A \propto 1/T;\ \therefore T \propto V$
- Around pulleys, + ΔT (tension gain) provided by - ΔT (tension loss)
  $\therefore$ Net Torque = 0
- Velocity Differential ΔV at pack rims transfers to tape as tension differential ΔT.
  I.e. Tape Tensioning is lossless.

Figure 4

NII™ OPTICEL "A" / DRIVE INTERFACE

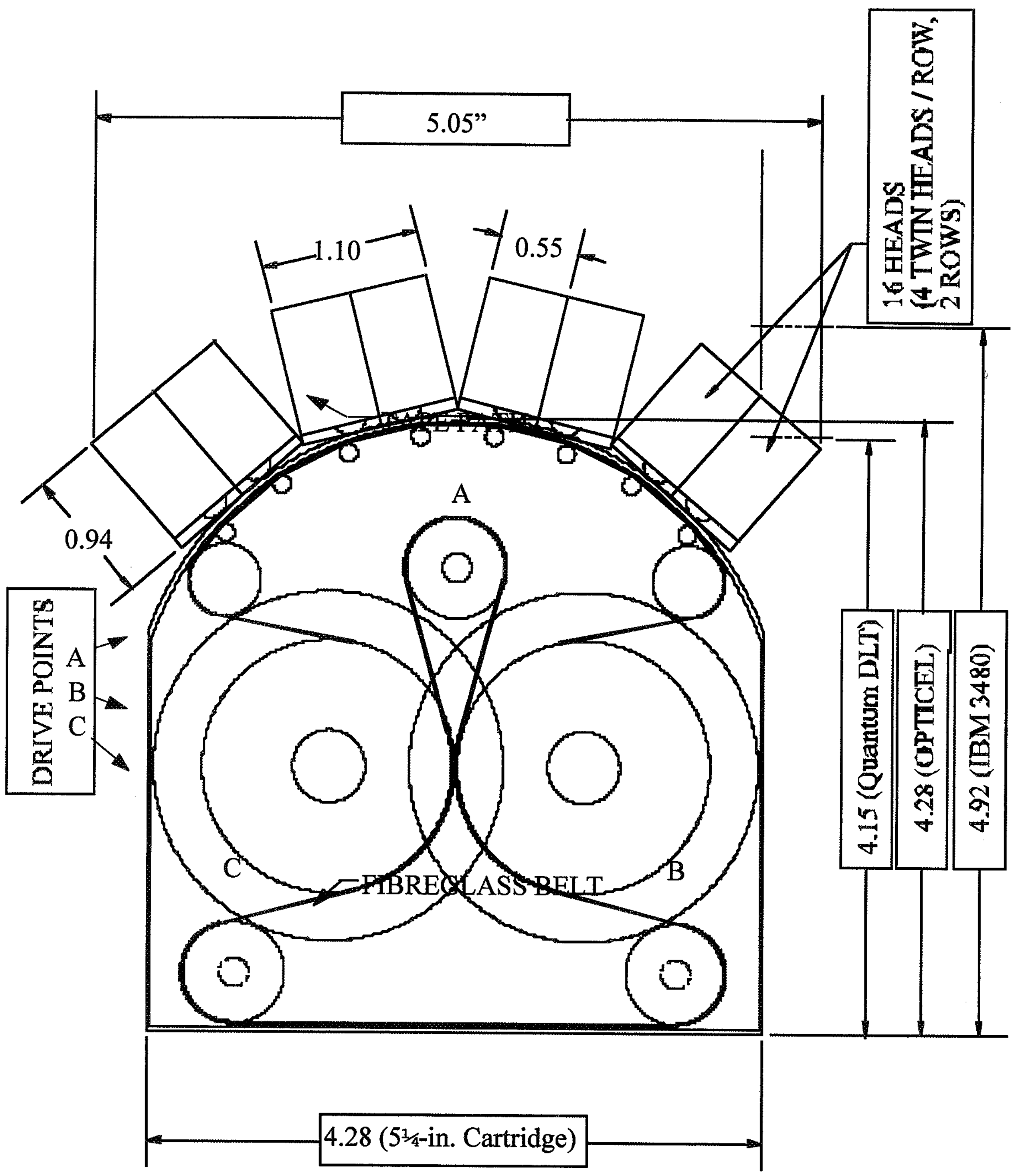


Figure 5

NII™ OPTICEL / DRIVE INTERFACE

SHOWING SINGLE MOTOR DRIVE ENGAGEMENT

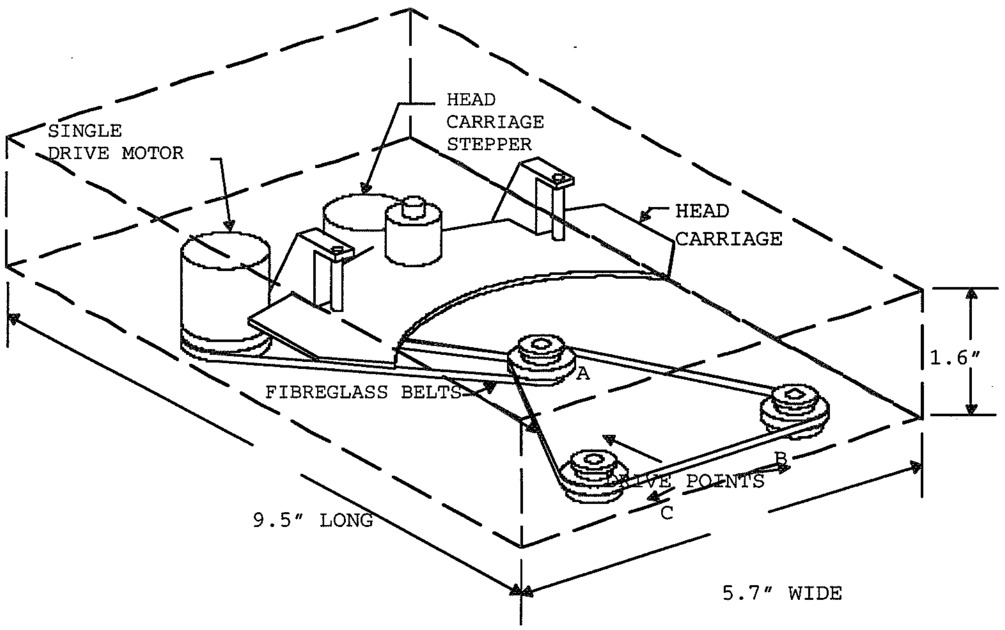
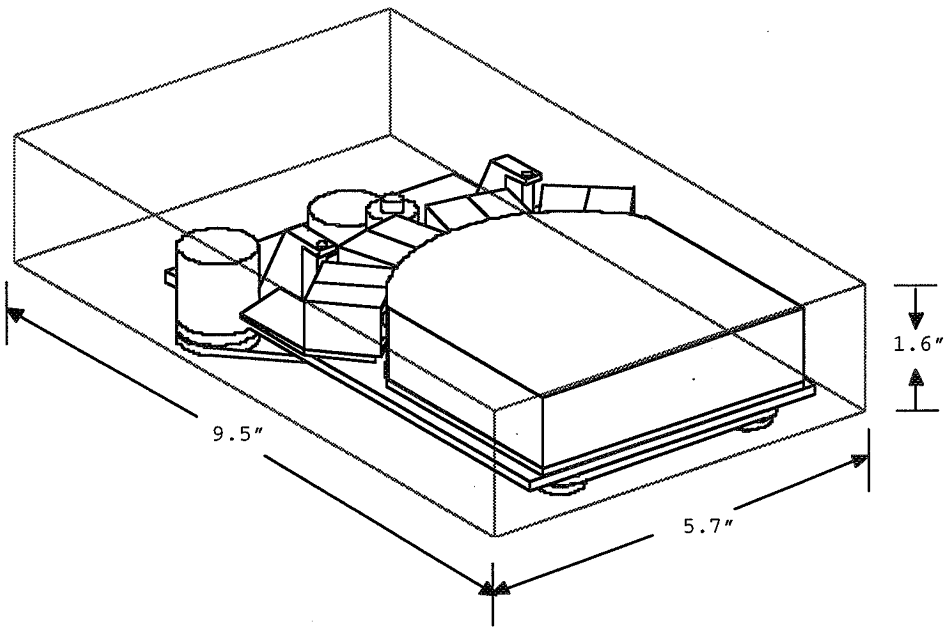


Figure 6

7 TERABYTE OPTICEL™ SYSTEM

P.C. BASED 19mm NII™ OPTICEL "A" CARTRIDGE / 5< - inch HALF HEIGHT DRIVE



Future Developments (Targets for Mass Memory Systems - "Part 3")

Introductory System

The NII™ Opticel will be introduced using the off-shelf WORM optical tape, with phase-change coating on a 1/2-mil substrate. The data transfer rates are limited in the write mode to the power of the write laser, and in the read mode to tape speed and electronics bandwidth.

Target specifications of the introductory Opticel system are:

Size:	Standard 5-1/4-inch half-height drive DLT stackloader-compatible cartridge		
Native Capacity:	7 TB in PC slot		
Search Time:	Statistical average to file - 1.5 sec. Minimum to first bit - 12 ms		
Transfer Rates: (sustained)	Read only:	Single head serial	10 MB/s min.
	Read only:	16-head parallel	150 MB/s min.
	Write only:	16-head parallel	150 MB/s max.
	Read after write:	Parallel write only x 1/2	75 MB/s max.

A desktop rack system comprising 16 Opticels would store:

16 x 7 = 100 TB. Read after write transfer rate > 1 GB/s

Higher Capacity Systems

Although the NII™ Opticel will be introduced with 218 meters of tape, higher-capacity archival systems are contemplated on a compatible migration path, evolving from the Opticel A.

Examples:

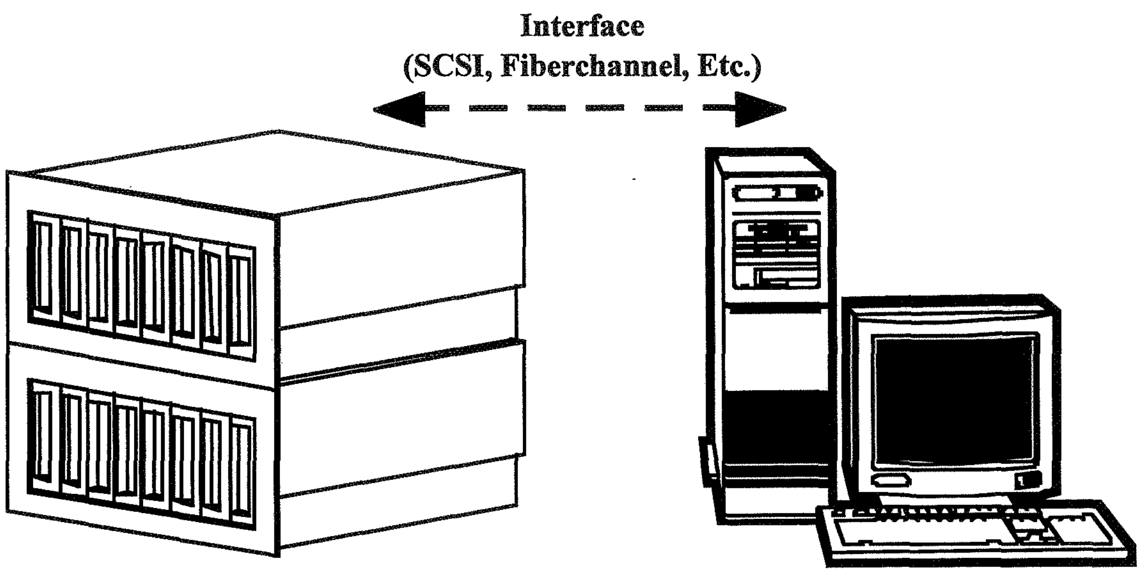
- Use of green lasers with super-resolution would more than double capacity

- A "stretch" NII™ Opticel with ultra-thin tape, and even faster drives, are anticipated to combine to provide capacity in a single Opticel system approaching a Petabyte without a commensurate increase in search time.

Multi-Petabyte Libraries

The capacity of the NII™ Opticel can be expanded by stacking multiple drives in a rack and providing parallel "on line" access (no cartridge exchange time).

The 12ms access time to first bit, average file access time of 1.5 sec (eq. 6) allows formatting of the tape to emulate multi-disk systems while providing storage capacity well beyond disk-based systems.



**7(a) NII™Oticel Image Archive System
100 TB**

**7(b) Workstation
with Robotic Cartridge Changer 2 PB +**

Figure 7

**NII™ Opticel "A" Digital Archive System
100 TB Desktop Unit expandable to Multipetabyte System in Workstation**

Fig. 7A shows a desktop unit with 100TB on-line. Performance is given in Table 1A.

Fig. 7B shows the desktop system expanded in a workstation with a robotic cartridge changer to a near-line capacity of 2 Petabytes. Performance is given in Table 1B.

Not Shown: The 2 Petabyte system can be further expanded open-endedly for near-line access to large archival libraries that can accommodate the additional cartridge exchange time (typically 20 sec.).

- A single ATL StorLink will interface up to 36 Opticel drives for 1/4 PB on line and 528 Opticel cartridges for 3-3/4 PB near line.

- Six libraries can be linked with pass-through capability for an additional 20 PB of near-line storage.

Erasable Systems

It is expected that an erasable NII™ Magnacel can be developed which is "plug transparent" to its archival counterpart.

It will employ proprietary new technology in the following areas:

- ME coated ultra-thin tape
- MR thin-film read heads
- Thin-film record heads

- Embedded formatting
- Encoding and ECC for very high magnetic areal densities and ultra-high transfer rates

International Operations

Primelink (Canada) is now forming strategic alliances and contracts with leaders in the industry, and arranging major financing.

It will continue Part 3 of the program as a U.S. corporation under the name "Kyber Corporation."

References

1. Chester W. Newell, "Petabyte Mass Memory System Using the Newell Opticel," Part 1, presented at the Fourth NASA Goddard Conference on Mass Storage Systems and Technologies, March 28-30, 1995.

2. Issued U.S. Patent No. 4,172,569 to C. W. Newell.

Portfolio of other patents and patents pending, available upon request under Non-Disclosure Agreement.

Table 1

COMPARISON: DLT TO NII™ OPTICEL

A. Desktop Mass Memory Systems	**DLT**	**Opticel "A"**	**Advantage**
Drive Size 5-1/4 std. footprint	Full height	1/2 Height	2 X
Native Capacity			
(a) PC-based stand-alone Cart/drive, Max:	35 GB	7 TB	200 X
(b) 19" rack desktop "near line" cart/drives	4 drives 140 GB	16 drives 100 TB	800 X
Search Time:			
Statistical average to file	50 sec.	1.5 sec.	30 X
Minimum to block	N/A	12 ms	--
Transfer Rates: (MB/s)			
Read only - serial:	1.25	10	8 X
Read only - parallel	2.5	150	60 X
Write only - parallel	2.5	150	60 X
Read after write	N/A	75	--

B. Automated Tape Library Systems	**DLT**	**Opticel "C"**	**Advantage**
Typical Changer (carts "near line")	274*	290*	
Native Capacity	10 Terabyte	2 Petabyte	200 X
Average Access Time			
To cartridge	20 sec.	20 sec.	1.0 X
To file (statistical)	+98 sec.	+1.5 sec.	65 X
To block (min.)	N/A	12 ms	--
Transfer Rate (MB/s)			
No. of drives "near line"	10	16	
Read/write	25 MB/s	1.2×10^3	48 X
Write only	25 MB/s	2.4×10^3	96 X
Read only	25 MB/s	2.4×10^3	96 X
Footprint: ft.^{2} per Petabyte	37	4.6	8X

* e.g. ATL 2640

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	**2. REPORT DATE** March 1998	**3. REPORT TYPE AND DATES COVERED** Conference Publication

4. TITLE AND SUBTITLE
Sixth Goddard Conference on Mass Storage Systems and Technologies held in cooperation with the Fifteenth IEEE Symposium on Mass Storage Systems

5. FUNDING NUMBERS
Code 423

6. AUTHOR(S)
Benjamin Kobler and P C Hariharan

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS (ES)
Goddard Space Flight Center
Greenbelt, Maryland 20771

8. PEFORMING ORGANIZATION REPORT NUMBER
98B00027

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS (ES)
National Aeronautics and Space Administration
Washington, DC 20546-0001

10. SPONSORING / MONITORING AGENCY REPORT NUMBER
GSFC/CP–1998–206850

11. SUPPLEMENTARY NOTES
Benjamin Kobler: NASA Goddard Space Flight Center, Greenbelt, Maryland
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,
800 Elkridge Landing Road, Linthicum Heights, MD 21090; (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 Sixth Goddard Conference on Mass Storage Systems and Technologies which is being held in cooperation with the Fifteenth IEEE Symposium on Mass Storage Systems at the University of Maryland-University College Inn and Conference Center March 23–26, 1998. 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, tape optimization, new technology, performance, standards, site reports, vendor solutions. Tutorials will be available on shared file systems, file system backups, data mining, and the dynamics of obsolescence.

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

15. NUMBER OF PAGES
440

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