NASA Conference Publication 3340, Vol. II

Fifth NASA Goddard Conference on Mass Storage Systems and Technologies

Volume II

Proceedings of a conference held at
the University of Maryland
University College Conference Center
College Park, Maryland
September 17 - 19, 1996

NASA Conference Publication 3340, Vol. II

Fifth NASA Goddard Conference on Mass Storage Systems and Technologies

Volume II

Edited by
Benjamin Kobler
Goddard Space Flight Center
Greenbelt, Maryland

P. C. Hariharan
Systems Engineering and Security, Inc.
Greenbelt, Maryland

Proceedings of a conference held at
the University of Maryland
University College Conference Center
College Park, Maryland
September 17 - 19, 1996

National Aeronautics and
Space Administration

Goddard Space Flight Center
Greenbelt, Maryland

1996

Fifth Goddard Conference on Mass Storage Systems and Technologies

Program Committee

Ben Kobler, *NASA Goddard Space Flight Center (Chair)*
Jean-Jacques Bedet, *Hughes STX Corporation*
John Berbert, *NASA Goddard Space Flight Center*
Jimmy Berry, *Department of Defense*
William A. Callicott, *Consultant*
Sam Coleman, *Lawrence Livermore National Laboratory*
Robert Creecy, *Census Bureau*
Charles Dollar, *University of British Columbia*
Fynnette Eaton, *National Archives and Records Administration*
P C Hariharan, *Systems Engineering and Security, Inc.*
Bernard O'Lear, *National Center for Atmospheric Research*
Sanjay Ranade, *Infotech SA, Inc.*
Bruce Rosen, *National Institute of Standards and Technology*
Don Sawyer, *NASA Goddard Space Flight Center*
Peter Topoly, *National Oceanographic and Atmospneric Administration*

Production, Copy Editing, and Layout

Len Blasso, *Media Specialist Associates*

This publication is available from the NASA Center for AeroSpace Information, 800 Elkridge Landing Road, Linthicum Heights, MD 21090-2934, (301) 621-0390.

Preface

The Fifth Goddard Conference on Mass Storage Systems and Technologies has attracted more than forty papers which are included in these Proceedings. We plan to include audio and video and, if available, text and viewgraphs from the invited papers and the panel discussion in a CD-ROM which will be published before the end of 1996.

A paper on application program: ning interfaces (API) for a physical volume repository (PVR) defined in Version 5 of the IEEE Reference Model (RM) for Open Storage Systems is indicative of ongoing activity to flesh out the RM. However, there still remain a number of other interfaces in the RM which lack APIs. A number of agencies have already deployed petabyte-sized archives with custom FSMS since there are no standards yet, and so there are no COTS software modules which can be combined/integrated to provide file and storage management services. A user panel will discuss the problems and issues associated with available software and, it is hoped, will lay out the desiderata which experience has shown is required for the management of large archives.

Storage architecture, database management and data distribution are covered in three sessions. The future of recording is not necessarily a mix of optical and magnetic technology; as the paper by Stutz and Lamartine shows, microchisels are around the corner, and may provide a solution to the problem of technology obsolescence which has been exacerbated by the ever shorter product development and life cycles. O,tical technology is updated by papers from the Air Force's Rome Laboratory, and from LOTS Technology.

File system performance and modeling are dealt with by a number of authors, and there are progress reports on the definition and use of metadata in archives.

Descriptions of specific archives and storage products have been moved this year to a poster session. Storage vendors will have a special session where they can explain, elaborate and extol their particular solutions.

We are grateful to the members of the Program Committee:

Jean-Jacques Bedet, Hughes STX Corporation
John Berbert, National Aeronautics and Space Administration
Jimmy Berry, National Security Agency
Bill Callicott, consultant
Sam Coleman, Lawrence Livermore National Laboratory
Robert Creecy, Census Bureau
Charles Dollar, University of British Columbia
Fynnette Eaton, National Archives and Records Administration
Bernie O'Lear, National Center for Atmospheric Research
Sanjay Ranade, Infotech SA

Bruce Rosen, National Institute of Standards and Technology
Don Sawyer, National Aeronautics and Space Administration
Peter Topoly, National Oceanic and Atmospheric Administration

for their diligence in identifying the topics and securing the excellent papers for this conference.

We also record our thanks to:

John Otranto, Systems Engineering and Security, Inc for help with some of the figures; Len Blasso, Media Specialist Associates, for editing and layout; Jorge Scientific Corporation for logistics support.

P C Hariharan
Systems Engineering & Security, Inc
Greenbelt MD 20770-3523

Ben Kobler
NASA Goddard Space Flight Center
Greenbelt MD 20771-1000

Table of Contents

Volume I

Table of Contents

Volume II

NEXT
DOCUMENT

S -3205 -

Development of Secondary Archive System at Goddard Space Flight Center Version 0 Distributed Active Archive Center

Mark Sherman, John Kodis, Jean-Jacques Bedet, Chris Wacker
Hughes STX
7701 Greenbelt Road, Suite 400
Greenbelt, MD 20770
{sherman, kodis, bedet, wacker}@daac.gsfc.nasa.gov
301-441-4285 Fax (301) 441-2392

Joanne Woytek, Chris Lynnes
NASA/GSFC
Greenbelt Road
Greenbelt, MD 20771
{joanne,lynnes}@daac.gsfc.nasa.gov
301-286-4418

Abstract

The Goddard Space Flight Center (GSFC) Version 0 (V0) Distributed Active Archive Center (DAAC) has been developed to support existing and pre Earth Observing System (EOS) Earth science datasets, facilitate the scientific research, and test Earth Observing System Data and Information System (EOSDIS) concepts. To ensure that no data is ever lost, each product received at GSFC DAAC is archived on two different media (VHS and Digital Linear Tape (DLT)). The first copy is made on VHS tape and is under the control of UniTree. The second and third copies are made to DLT and VHS media under a custom built software package named "Archer". While Archer provides only a subset of the functions available with commercial software like UniTree, it supports migration between near-line and off-line media and offers much greater performance and flexibility to satisfy the specific needs of a Data Center. Archer is specifically designed to maximize total system throughput, rather than focusing on the turn-around time for individual files. The Commercial Off the Shelf Software (COTS) Hierarchical Storage Management (HSM) products evaluated were mainly concerned with transparent, interactive, file access to the end-user, rather than as a batch-oriented, optimizable (based on known data file characteristics) data archive and retrieval system. This is critical to the distribution requirements of the GSFC DAAC where orders for 5000 or more files at a time are received. Archer has the ability to queue many thousands of file requests and to sort these requests into internal processing schedules that optimize overall throughput. Specifically, mount and dismount, tape load and unload cycles, and tape motion are minimized. This feature did not seem to be available in many COTS packages. Archer also utilizes a generic tar tape format that allows tapes tc be read by many different systems rather than the proprietary format found in most COTS packages. This paper discusses some of the specific requirements at GSFC DAAC, the motivations for implementing the Archer system, and presents a discussion of the Archer design that resulted.

Introduction

One of the critical components within the DAAC's Data Archive and Distributed System (DADS) is the HSM system. Several years ago, UniTree was chosen as the best candidate to satisfy the GSFC DAAC 's requirements providing both the basic HSM functions and the device drivers for the planned robotic devices. After months of integration and customization, UniTree reached some stability but it fell short of the GSFC DAAC throughput requirements [1], and was limited in the configurability of the archive, retrieval, and caching systems based on data-specific characteristics; e.g., size, volume, likely reuse, multiple versions, etc. It also became apparent that this product and other similar commercial products were not fully suited for this domain of application.

Archer is an in-house software package that was developed by the GSFC DAAC to provide management of secondary and tertiary backup copies of all datasets stored in the archive. Archer was developed to remedy some of the major drawbacks of HSMs, such as UniTree, in handling a data (vs. file) archival system In particular its design was kept simple and tailored to handle data requests with large number of files and varying files characteristics. Performance was a key consideration in the design of the system and its highly parallel distributed architecture allows the system to be scaled to much larger archives. This paper starts by presenting an overview of the functionality needed for the GSFC DAAC to be a fully operational Data Center. The overall hardware architecture to meet the needs of the GSFC DAAC is described, followed by a discussion on what led the GSFC DAAC to the development of Archer. The architectural design of Archer is presented with its main features. Finally. the status, lessons learned, and future work are briefly described.

GSFC DAAC functions and architecture

The GSFC DAAC can be viewed as composed of three main components which are a Product Generation System (PGS), an Information Management System (IMS), and a Data Archive and Distribution System (DADS). The PGS and IMS are respectively associated with the production of higher level products and the catalog holdings searched and browsed by researchers. The DADS controls the overall processes of the ingestion of new data and the distribution of data requests. The migration between near-line and on-line devices is handled by both UniTree and Archer, however only Archer has the full capability to migrate media between near-line and off-line. For historical reasons, UniTree is currently responsible for the primary archive. Secondary and a tertiary archives, under the control of Archer, use respectively DLT and VHS as archive media. The Metrum RSS-600 Automated Tape Library (ATL) with 5 RSP-2150 drives and 600 VHS cassettes (for a total capacity of up to 8.7 TB) is shared by UniTree and the tertiary archive. Most tapes in the ATL and four of the five VHS drives are controlled by UniTree. The secondary archive is composed of three DLT 7 cartridge stackers. While

UniTree and the tertiary archive are run on an SG^I ^hallenge L, the secondary archive is executed on an SGI Challenge S.

Two SGI 4D/440 workstations are being used to test new version of the DADS, IMS, Archer software and new releases of UniTree. Having dedicated test machines is very important to avoid affecting the day to day operation at the GSFC DAAC. Several SGI machines are also used to process Pathfinder Advanced Very High Resolution Radiometer (AVHHR) land products and to perform Quality Assessment (QA) on new products generated. Figure 1 and 2 and Table 1 illustrate some of main platforms acquired by GSFC DAAC along with their specific functions.

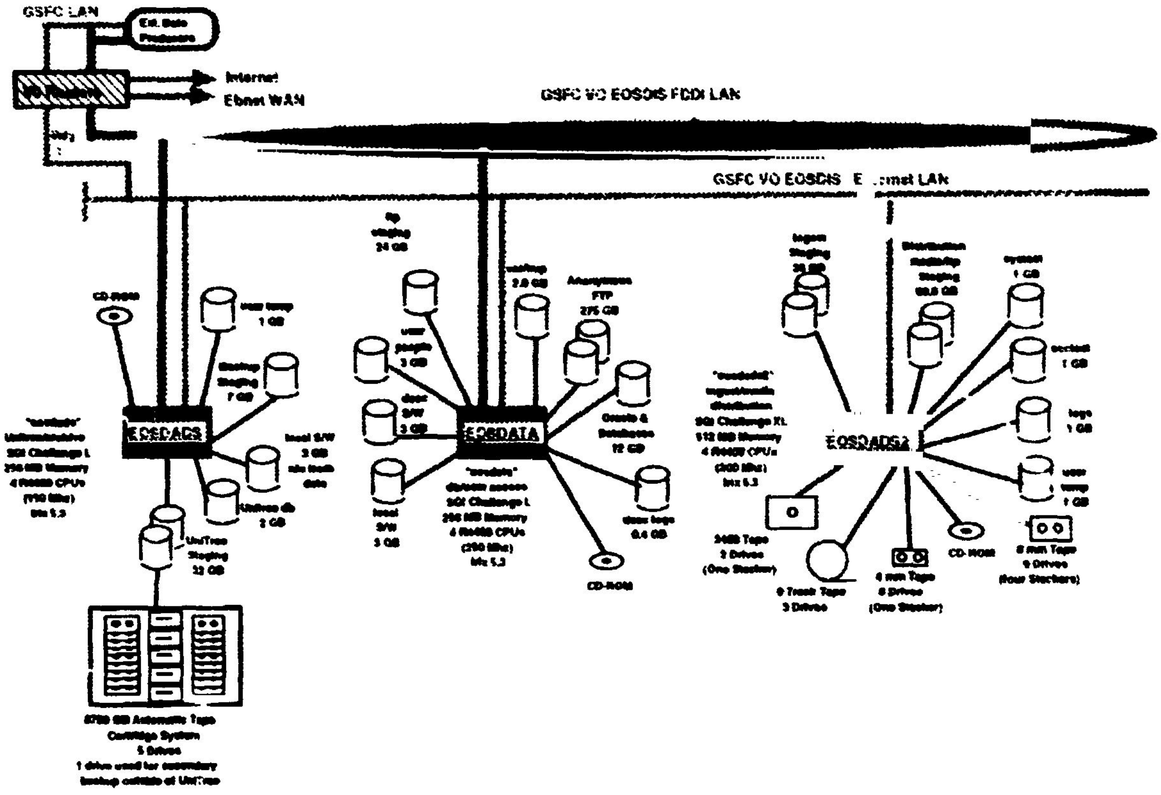


Figure 1 GSFC DAAC 1996 Configuration as of 2/28/96 (1 of 2)

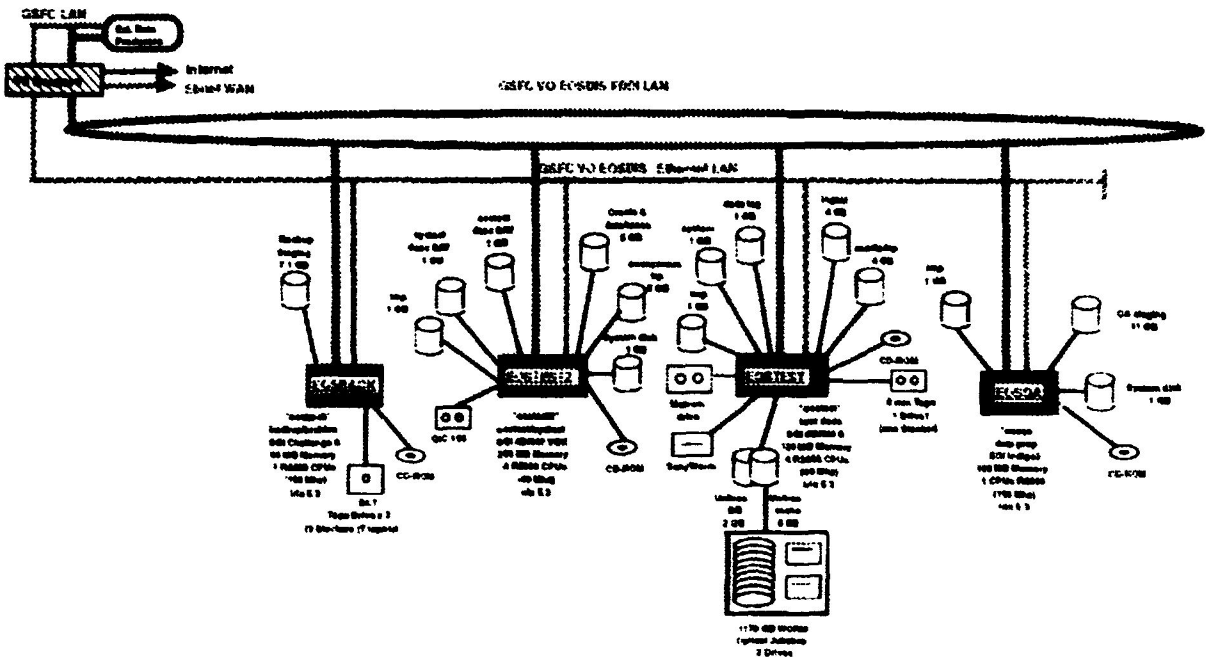


Figure 2 GSFC DAAC 1996 Configuration as of 2/28/96 (2 of 2)

Machine name	Function	Hardware description
EOSDADS	run UniTree & tertiary archive	SGI Challenge L, 256 MB memory 4 R4400 CPUs (150 Mhz) - Metrum RSS600 automatic library - 32 GB UniTree stage disks
EOSBACK	run secondary Archive	SGI Challenge S, 64 MB memory 1 R4400 CPU (150 Mhz) - DLT stackers
EOSDATA	run IMS and Oracle Database	SGI Challenge L, 256 MB memory 4 R4400 CPUs (250 Mhz) - 24 GB ftp stage disks - 275 GB anonymous ftp
EOSDADS2	run ingestion & distribution	SGI Challenge XL, 512 MB memory 4 R4400 CPUs (200 Mhz) - 36 GB ingest staging disks - 61 GB distribution staging disks - 8mm drives - 4mm drives - 3480 drives
EOSTEST2	test software in acctest & systest	SGI 4D/440 VGX, 256 MB memory 4 R3000 CPUs (40 Mhz)

EOSTEST	test dads software & new version of UniTree	SGI 4D/440, 128 MB memory 4 R3000 CPUs (40 Mhz) - 8 GB UniTree cache
EOSQA	run data product QA	SGI indigo 2, 160 MB memory 1 R4400 CPU (150 Mhz)

Table 1. Hardware at the GSFC DAAC

Criteria for the development of a secondary archive

This paper now focuses on issues faced by the GSFC DAAC during the last two years and some of the specific requirements that led to the development of a secondary archive system.

Over the years, the GSFC DAAC has faced problems with the HSM system UniTree and the archive media (VHS tapes and 12" WORM optical platters). In particular, UniTree did not work very well when 12" WORM optical drives were working concurrently with the VHS tape drives. Unitree also did not satisfy the general throughput requirements. and proved difficult to configure based on evolving data characteristics and data request profiles. While some issues have been resolved, others still remain open. Additionally, occasional loss of data due to media failure, UniTree software failures, along with a requirement from the Sea-viewing Wide Field of View Sensor (SeaWiFS) project necessitated the need to keep a second copy of all products. It became apparent that there was an urgent need for a secondary data archive system that would hold a backup copy of all data received at the GSFC DAAC, would take over in case the primary system failed, and if successful in increasing throughput, could be used as a primary retrieval system At the time UniTree was not fully stable and the GSFC DAAC was under increasing need to provide better, more reliable data retrieval and a robust data recovery capability which did not rely on the data provider to re-send lost data. The choices were either to purchase a second COTS product or to develop our own secondary data archival system. The data archive system was intended to mostly store data to archive tapes, track file location and tape utilization, and to handle both near-line and off-line tapes. Most COTS packages evaluated were deemed too sophisticated and expensive for the simple set of requirements that had been identified. Further, many of the COTS HSMs , which were oriented towards transparent, interactive file retrieval functionality, did not seem to fully meet these simple requirements. This was particularly true for automatic migration of media between near-line and off-line storage, and large, batch oriented file/data requests. Our experiences with the UniTree COTS package also pointed out other problems with commercial HSMs, such as performance bottlenecks and maintainability issues. For these reasons, the decision was made that the GSFC DAAC would gain by developing its own secondary data archive system. The remainder of this section focuses on some of the criteria that were factored into the secondary archive design.

As mentioned above, UniTree was designed around limited, interactive file access which imposed limitations that were undesirable for a large scale science data center. For instance, UniTree limits the number of concurrent stage operations (around 100) which causes major problems when large number of files are to be staged. Also, the order of requesting and staging data, along with adequate feedback on both successful and unsuccessful retrievals, are critical, both to achieve good performance, and to simplify the media distribution process. For example, a request may need a set of files staged and then copied to a number of 8mm tapes for distribution in the time order in which the data was initially produced. The request would best be handled by staging in the time order to be distributed, particularly if multiple distribution tapes will be needed. Additionally, in a production environment it is not unusual to have unexpected hardware and software problems or unexpected workloads that must be rectified manually. Therefore, it is important to have full control over the archive, letting the system run by itself, but allowing operators to take control of the system when needed. To provide flexibility and adaptability to facilities with the needed requirements and resources, HSMs should have an Application Program Interface (API), which many commercial products either do not provide or provide with very limited capabilities. It would be highly desirable to have standardized APIs to facilitate transition to a new HSM when needed.

A key element of a typical data retrieval request submitted at the GSFC DAAC is the need to stage, in one request, a large number of small files. Some HSMs tend to perform poorly when several hundred or thousand of files need to be staged, even if the files reside on few tapes. Other products put a limit (e.g. 100) on the number of stages that can be submitted at once, reducing overall performance, requiring substantial software design to properly handle the staging, and having a large impact on the day to day operations. On average, most of the files currently archived at the GSFC DAAC are small (around 1 MB) while data requests range from a single file to several thousand files at a time, resulting in a high penalty when retrieved from tapes. The overhead of the pick, mount, load, search and rewind operations is high compared to the read/write operation which may take only a few seconds for these small files. Consequently, it is critical to minimize the number of mounts and maximize, whenever possible, the amount of files read/written per mount. It is therefore desirable to sort the order in which files are transferred to and from tapes by which tape they are on and their position on the tape. This may be achieved by knowing the physical location of the files on tapes and then writing software to request the files in that order. Unfortunately, this information is not easily available in HSMs such as UniTree. To maximize system throughput, it is also necessary to keep data transfer rates to/from the storage devices at nearly the limits imposed by the hardware. Detailed analyses were done on the performance of the VHS drives under UniTree, and it was shown that data transfer rates were substantially less inside UniTree than those measured outside UniTree, even with just a single drive operating [1].

Performance is a key issue in an archive, but other considerations such as interoperability are equally important. HSM vendors with their own proprietary formats make the transition to another HSM very difficult and expensive. This can have disastrous consequences if a vendor decided to stop marketing their products or to stop support of a given hardware device, as was the case for UniTree and the Cygnet jukeboxes at the GSFC DAAC. The situation worsens as the size of archives increases dramatically (Petabytes). The GSFC DAAC also has a requirement to migrate all of its archived data under the control of the Version 0 system to the next generation system. By storing the data in a non-proprietary, generally used format such as tar, migration can be more easily and quickly accomplished, since all that is required is to physically move the tapes to the new system. The interoperability of the tapes can be resolved by having one or several standardized tape format(s). This is difficult to achieve when vendors disagree on the merits of the formats and have already invested large amount of money in them. Another approach may be to provide a mechanism for HSMs to recognize and read formats from various vendors and do this without sacrificing performance. An important feature that is not always available is the ability to reconstruct the data base from the data itself. For instance, UniTree data is useless without the UniTree data base. These problems have been recognized and an Information and Image Management International (AIIM) File Level Metadata for Portability of Sequential Storage Media group has been formed to address some of these issues. This group met for the first time in April 1996, in Chicago, Illinois.

Faced with storage requirements growing exponentially and limited budget, it may be necessary to store data off-line. This solution is even more attractive in a data center where many tapes are seldom requested. This feature seems to be ignored or is limited at best with some HSMs. It is not sufficient to indicate that the tape is off-line. At a minimum the physical location of each off-line media should be known by the HSM and operators should be prompted to transfer media between near-line and off-line in an efficient manner. This should be viewed as another level of hierarchy with full functionality, and statistics should be made available.

A key issue in any Data Center is the data integrity and the data preservation. To ensure the highest quality for all data ingested and distributed to the users, it is important to capture, report, and react to errors in a usable way. These errors could occur with the media, the drives, the disks, or be related to some software problems. Even soft media errors may need to be monitored to identify archive media degradation. Data corruption needs to be automatically detectable through methods such as computation and comparison of file checksums upon all archival and retrieval requests. In spite of being critical, errors are not always provided with enough information, are often listed in a cryptic form, are difficult to locate in log files, or are simply not reported. Programs requesting the data are often not provided with adequate feedback to respond to both critical (e.g., hard media) failure and non-critical (e.g. soft media) failures. This creates confusion, requires a high level of expertise, and can have a detrimental impact on day to day operations. Error detection is not sufficient in itself and "smart" algorithms should be in place to take appropriate actions after errors are discovered. For example, a configurable limit should be set pertaining to the number of retries to read or search for a file. Another example may be to not automatically mount new media when an unrecoverable write error is detected, since the problem could be due to a bad drive and could result in numerous new tapes being discarded. Similar problems can occur with WORM optical media where a failure due to a bad drive is incorrectly interpreted to be a media failure and a new media is requested. When the request again fails, another new media is requested, and so on, until operators notice the problem and shut-down the operation. This can cause the loss of many platters and requires extensive manual intervention to rectify this situation . These examples illustrate that the hard-coded error handling policies implemented for general, success-oriented operations do not always function well within a large, operational system. These policies are easily correctable and changeable. Changing policies and requirements may be a trivial task to implement with an in-house system, but may be much more difficult to integrate with a commercial packa e.

When dealing with very large science data centers (Petabytes), scalability is a major issue. An HSM should be designed to scale not only with the volume but also with the number of files being archived. This may require distribution of the software as well as the hardware. Implementation of a Unix file system or a virtual disk system is not regarded as a viable solution because of its limitations. There is a limit in the operating system on the number of concurrent open calls. The name server in an HSM can also become a bottleneck with very large number of files and some of the modules composing a data archive system may have to be distributed over several machines to spread the load more evenly.

Purchasing a commercial product such as an HSM provides many advantages. On the other hand, there may be major drawbacks that should be diligently evaluated before making any decision regarding the need for a COTS product versus an in-house product. One major problem experienced at the GSFC DAAC was the integration of UniTree with custom archive and distribution software. The task was difficult, time consuming, expensive to implement, and caused long delays in the delivery of the whole system. One solution was to request the vendor to incorporate the desired functionality in a new release. However, these functions may be too specific to have market value; or when there is interest to other users, it usually takes months, if not years, before design, integration and release. Another approach is to contract the integrator to develop specific functions that are not part of the core commercial product. Besides the length of time to set-up the contract, provide the requirements, and then design, write, test and integrate the functions, there is a high risk involved in tailoring a commercial product to meet specific needs, as each new release of the product may require new customized development resulting in a high cost. All together, the process can be extremely lengthy in time and frustrating in having to write work-around software or procedures to try and handle the situation while waiting for the vendor to react. HSMs are rather complex systems, built for specific, well-defined systems, and are not without flaws. Some of these bugs may seriously limit how the system can be used and it may take weeks or months to obtain a patch to fix the problem. While requiring in-house resources and expertise, there is more control with programs developed in-house. Bugs can usually be rectified more quickly and decisions can be made internally to prioritize them. Moreover, the experience we had with UniTree and the discussion we had with other colleagues tend to confirm that HSMs have not yet reached the stage of maturity found in products such as data base management systems.

Part of the original charter of the GSFC Version 0 DAAC was to test EOSDIS concepts and standards. Experimentation with various HSM strategies and the development of Archer as a possible alternative to commercial HSM products fit within that charter. Having an in-house product would also increase the ability to add new media types, which usually takes place on a longer time scale with COTS. The high cost of commercial HSMs is another consideration that cannot be ignored and contributed heavily in the decision to develop Archer. This is even more important in a distributed environment where a home-grown HSM can be freely redistributed whereas a COTS has to be licensed for multiple platforms and sites. In addition to the expensive purchase price, there is usually a high maintenance cost and some integration development costs that makes commercial HSM solution less attractive. While the preference is to use a commercial product, in some cases no commercial product can satisfy specific and unique needs, and the developer must rely too much on companies whose goals are oriented towards slightly different requirements or functions. A key to the usability of a COTS product is whether its main functionality matches or just resembles one's needs. If just resembling one's needs, as was the case of COTS HSM packages and the science data needs of the GSFC DAAC, then attempting to either fit the COTS package into a slightly different functionality or assuming new releases to include the required functionality can be costly in time, resources, maintainability, and usability . These are some of the arguments and justifications that led to the design and development of a secondary archive system at the GSFC DAAC. One can hope that HSMs will become, in the near future, mature and flexible products that satisfy a vast and varied quantity of customers at a reasonable price.

Design of the secondary archive

Archer is a hierarchical storage management system that was designed to satisfy the requirements specified in the previous section. Files can reside in a cache, be robotically accessible, or be on a tape off-line. Users do not need to know the physical location of the files (data transparency), however, this information is easily and rapidly accessible through an API or by querying the Oracle data base which is used to keep track of file locations. The use of a relational data base facilitated and expedited the development of the system and provided a journal file to insure integrity of the archive database. Migration between cache and tape is automated and data can be stored and organized by families. For instance, a family can represent all files that belong to a specific product and level. The Archer file names are similar to the ones used in Unix, yet there is no implementation of a Unix file system. Consequently, commands such as open/close are not available and others, such as ls must be simulated through database SQL commands (e.g., and "als" command is provided to simulate ls). Files are simply requested to be stored or retrieved to/from the archive via PUT and GET operations. Multiple users can be serviced simultaneously and the client/server architecture has been designed to permit a distribution of the various servers among different machines to make 'he system scalable.

Archer file names have two parts. The first part identifies the directory to which a file belongs. The second part identifies the file. Both the directory and the file part can be any arbitrary string of characters (e.g. "/" are not required) but by convention, the names have been chosen to be consistent with Unix. Each directory is assigned to a family when created and is stored in an Oracle database table. The first part of a file name must completely match one the Archer directories, the part remaining is considered the file name.

The architecture of Archer is illustrated in Fig. 3. The main components of the system are defined as:

client interface (API): This is a series of C-callable entry points through which requests are originated. A request can be made to archive files, retrieve files, list files, delete files, list directories, list families, add tapes, list tapes, delete tapes, and flush families. All client interfaces communicate with a single archive server process.

Files can archived and retrieved in any size batch using either a synchronous or asynchronous method. The client is responsible for copying files out of cache during a file retrieval request. Command-line wrappers exist around all API functions so that the Archer internals can be accessed from the shell.

archive server: Only one archive server exists per archive. The archive server supports multiple file servers, and is responsible for directing message traffic between client processes and file servers or rejecting any requests which contain invalid information. The archive server can run on any machine in the archive.

file servers: Each file server is responsible for managing requests and file tables for a set of families in the archive. The file server manages cache space for all requests and verifies that the requests are satisfied. Each file server can manage multiple cache directories. Each file server supports multiple storage managers. For performance reasons, file servers may run on different machines in the archive.

copy server: A copy server is a small process which receives requests from the file servers to copy files into cache for archive requests. The copy server can copy a configurable number of files into cache in parallel. The copy server exists to minimize the overhead involved with forking processes to copy files in parallel. One copy server runs on each machine in the archive.

storage managers: Each storage manager is assigned a subset of the file server's families. Each may manage a different media type. The storage manager is responsible for managing and ordering the storage/retrieval of requests to/from tape. Each storage manager supports multiple storage servers, all of which must contain the same media type.

storage servers: Each storage server controls an individual storage device whether it is a single drive, a stacker, or a more complex multiple drive robotic system. The storage server is responsible for all activities involved in the storage/retrieval of files to/from tape. These activities include the loading/unloading of tapes to/from drives, tape positioning, tape verification, and the reading/writing of files to/from tape. Each type of storage server has its own type of ACE control display.

Archive Control Environment (ACE):

This is a GUI interface through which the operator and the archive interact. The ACE interface displays the status of the storage server and the device it is monitoring. This status includes whether the device is on-line, off-line, reading, writing, or idle, and the names of the tapes in the slots of the device, if applicable.

Through this interface, an operator may be notified of various events (e.g. system restarts, tape write errors), some of which may require a response. An operator may be prompted to mount a series of tapes in various slots of the device, or they may issue a request to load tapes manually

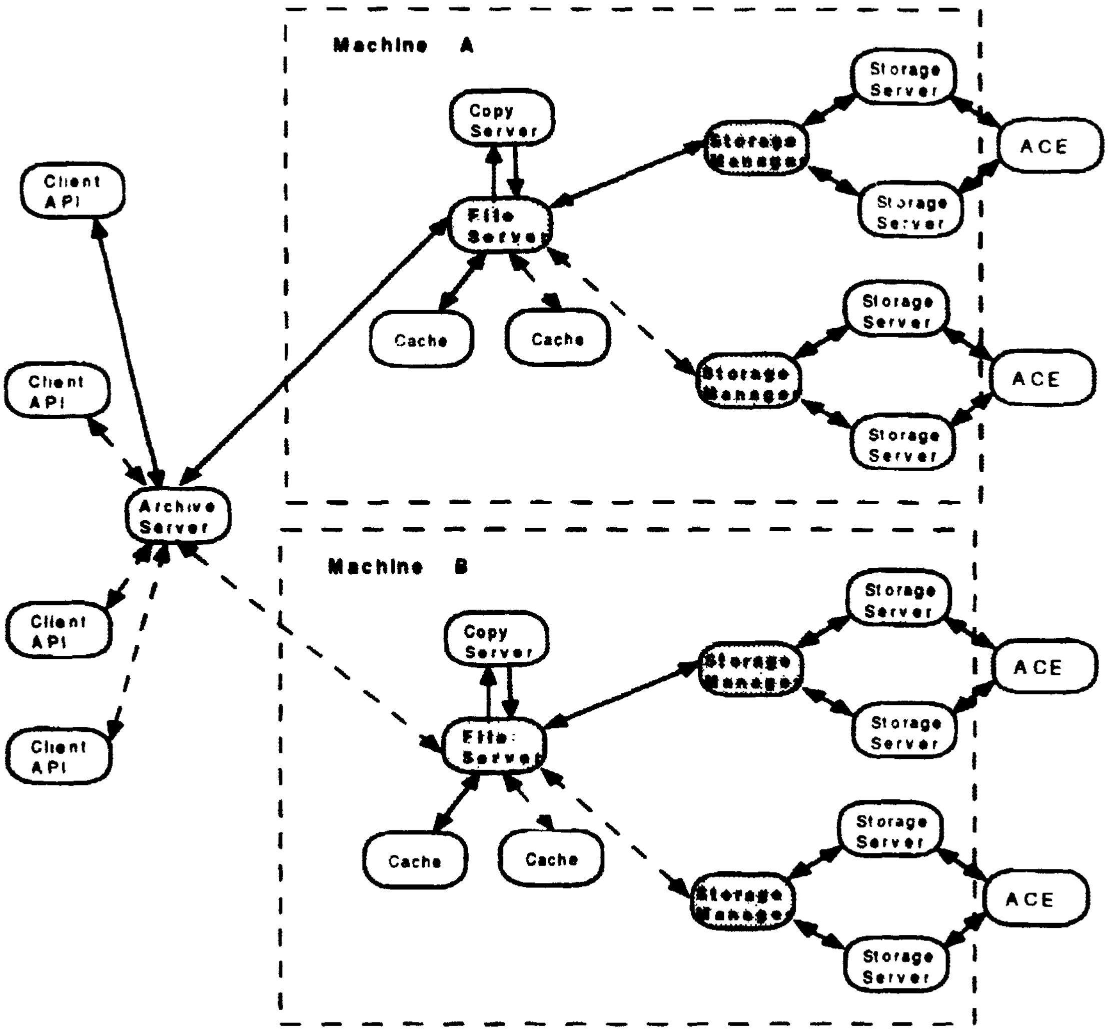


Fig 3 Archer Architecture

PUT and GET scenarios

In a typical PUT scenario, the client sends a request to the archive server to archive a file(s) to a specific family. The archive server directs the request to the appropriate file server. The file server allocates disk space in the cache and sends a message to the copy server to transfer the file(s) into the cache. After the file is copied to cache, a message is sent back through the system, informing the client of the cache transfer status. In a successful cache transfer, a message is sent to the appropriate storage manager. The storage manager receives and queues requests of successful cache transfers and waits for a pre-defined number of files (by family) to be staged in the cache before submitting a request to the storage server to copy the files to tapes. Finally, the storage server mounts the right tape and writes the data to it.

In a typical GET scenario, the client sends the request to the archive server which forwards it to the appropriate file server. The file server identifies whether the file(s) resides in the cache. When the file is not in the cache, the file server allocates disk space in the cache and sends a message to the storage manager to retrieve the file. When the storage manager determines the time is right to fetch the file, a message is sent to the appropriate storage server, the right tape is mounted, and the file is read from tape into the cache. A message is then transmitted back through the system informing the client of this transfer. To avoid authorization problems the client is responsible for copying data from the cache to its location.

Archer storage format

In designing the Archer storage format, the option of using a proprietary format such as the one implemented in UniTree was rejected due to concerns with portability, and flexibility. Another important consideration was the ability to reconstruct the metadata directly from tape without the need of the database. This feature can be useful in the event of a disaster and can also facilitate the migration to another archive system which may not have access to the database system. There is no official standard archive format available but tar is a de-facto standard with Unix and other platforms, and for this reason was selected as the best candidate to satisfy our requirements. As mentioned above, the GSFC DAAC average file size (at the current time) is relatively small (1 MB) and, therefore, saving each file in a separate tar format would result in a heavy performance and space penalty. To alleviate this problem, groups of files are saved in a tar file called a "save set" prior to being migrated to tape. The number of files to tar together is usually selected so that a "save set" is around 50-100 MB for a 1-2 MB/s tape drive. The size of the save set is configurable for different media and data types (i.e., families) in order to best utilize the performance characteristic of the tape drives based on the file characteristics of the data. When a file is requested from an Archer tape, the whole save set where the file resides is read from tape and untared on the fly. Reading a save set takes longer than reading a single file but this penalty is small compared to the high overhead associated with the mount/load/search times. In addition, since the data requests are based on high quantity, batch file retrievals, neither single file access (such as provided by UniTree) or, the even more granular, block oriented access (such as provided in the AMASS HSM system) provide any benefit, and can, in fact ,hurt overall performance for this type of system. The Archer storage format is illustrated in Fig 4.

Error detection and recovery

From the beginning of the design of Archer, special care was given to error detection and recovery. This is critical not only to minimize impact on day to day operations but also to insure the integrity of the data archived and distributed at the GSFC DAAC. The first type of errors to examine is media failure. When a tape write error is detected, several pre-assigned and operator configurable number of attempts are executed. Continued failure will cause an operator prompt to occur with the option to continue retrying the operation, to ignore the requested operation, or to retry the operation on a different tape in the case of a hard write error. If the operator chooses to ignore the requested operation, he/she can then take the suspected dnve off-line to avoid continuous operator prompts resulting from this write error. With a tape read failure, the read operation is retried for an operator configurable number of times, then marked as failed. Operators are notified on their terminals of the media problems.

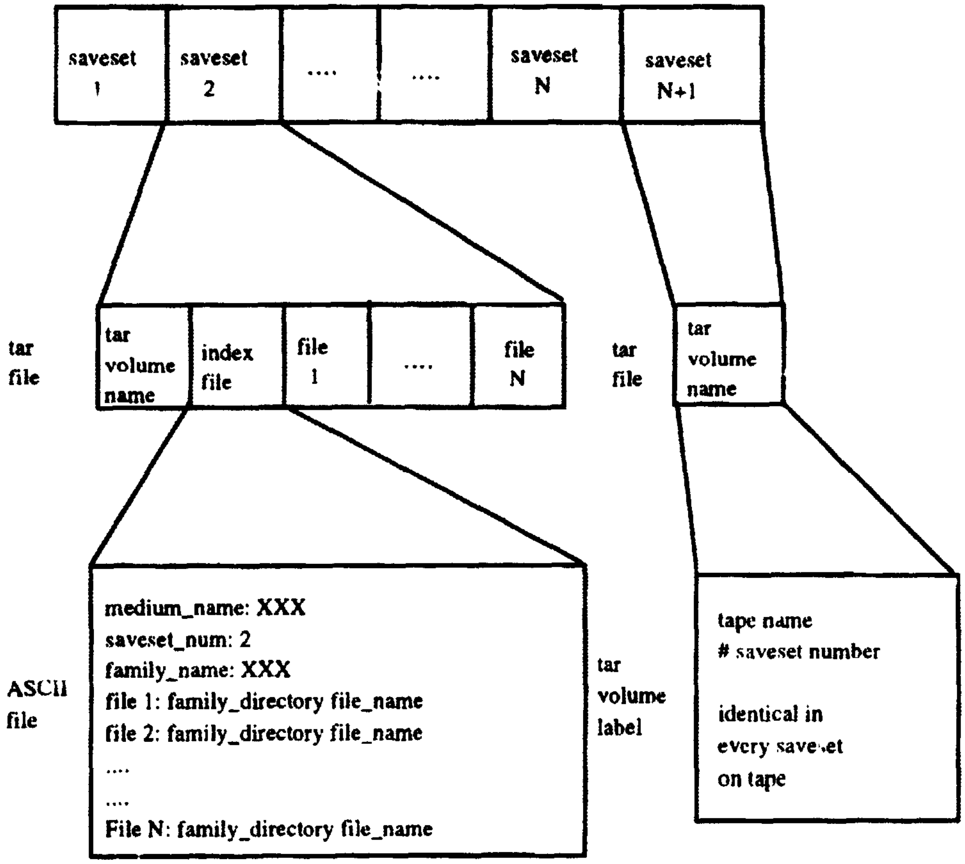


Fig 4 Archer tape format

Performance

One of the main considerations in the design of Archer was to develop a system with good performance. The emphasis was on the gross throughput of groups of related files as opposed to single-file turn around time. In order to achieve this objective seveial key features have been implemented. As mentioned above, files are grouped in save sets, improving the performance of a system with small files. To increase the hit cache ratio, a cacl ɜ management algorithm has been developed on the file server with tne capability to easily include new scheduling algorithms if desired. Improved log messages have also been designed to track the status of each file (examples: staged and purged) in the system and to monitor and generate performance statistics. New files ingested in the system are queued in the cache and copied to tape only after a pre-assigned volume of data is reached. This allows a large volume of data to be copied with a single tape mount. Files requested are first searched for in the cache. When the files are not located in the cache, Archer will sort files in the order they are physically stored on tapes, to minimize the overhead due to file positioning on the tape and the mounting and dismounting of tapes. Archer was developed with a multi-threaded client/server architecture and multi-threaded tape I/O architecture that provides efficient streaming of tape drives. The DLT tape drives have been tested to read/write close to the peak transfer rates advertised by vendors. Having a large database that contains the logical to physical relationship provides easy to utilize information but, due to the size of the files (millions) and the need to continuously access the table, performance is adversely affected. To partly alleviate this problem, the first part of the file name maps to the family name, which allows a quick identification of the table to which the file belongs. As mentioned in the Status and Future Work Section, future versions of Archer will be independent of a relational database system.

Operational concepts

One of the goals of Archer was to facilitate the operational activities at the [illegible] DAAC as well as the jobs performed by operators. One of the features of ACE (utilizing a graphical Tcl/Tk interface) is to provide a message button that highlights problems encountered. For example ACE (see Fig. 3) may list a tape write error . Archer processes are carefully monitored by an overseer process and if a problem arises, a message is displayed to indicate if the processes exited normally, abnormally, or failed due to a signal. In the event of failure, the archive is automatically restarted and the operator is notified.

Table 2 summarizes the issues discussed above.

Table 2: Summary of Archer Features and Functions

Issues	Features
Good performance	-low overhead to sustain operation at near tape speed - minimize number of mounts - maximize number of files requested from tapes - multi-threaded tape I/O - multi-threaded client services - hierarchical storage (disk cache, magnetic tape. off-line) - sort file read order by tape - allow large batch reads for improved so[illegible]
Interoperability	- no proprietary tape format (use tar) - open system - self contained (contains data & metadata) (HDF) - recreate metadata dbms from reading tap
Large requests of small files	- save set
Archive management	- support on-line, near-line, and off-line media
Flexible	- API -configurable parameters (based on data type or families, media, system, etc.).
Capture and monitor errors	- tape drive - media - disk cache failure - ACE display/monitor system
Error recovery	- before file is cached - before migration - during migration
Scalable system	- distributed H/W - distributed S/W - distributed storage devices
Administration	- reliable - archive multiple copies - collect statistics - errors - performance - facilitate migration from V0 to V1 - reduce dependencies on vendors - minimum coupling with DADS software - simplify integration - simplify exportation - integrity - journal file - support operator assisted off-line tape access
Kept simple	- does not implement a Unix file system - file name similar to Unix file system - simple synchronous and asynchronous put/get user interface - retrieval is by family and file identifier - COTS software to handle archive database
Hierarchical storage management	- files can be in cache, on tape, or off-line - identical storage and retrieval operations - automatic migration from cache to tape

Status and Future work

Since its delivery in Fall 1995, Archer Version 4.4 has been used on several occasions to recover lost files. Based on random audits, no file loss from Archer has yet been detected and Archer outperforms UniTree in archive operations, especially with large batches. There have been some operational problems. For example, some unexpected tape errors have occasionally caused the Archer system to hang. Also, only one single cache disk is currently supported and file and tape status is available only through SQL database queries.

The next build of Archer, scheduled to be operational in August 1996, should improve the overall performance through better internal scheduling of database operations. Multiple cache support has been added. Error recovery has been modified to prompt operators when several tape retries failed and to provide a choice of options. A global process monitors Archer and alerts operators to any problem detected.

Several other NASA groups have expressed an interest in Archer and there are plans to enhance Archer to be more like a COTS package with full documentation and its own configuration management (independent of the DADS development). The two main features envisioned are to remove Archer dependency on Oracle by maintaining the needed information internally and in disk files, and to improve the storage manager and storage server to better support new robotic devices and drives.

Conclusion

The GSFC DAAC has successfully designed and implemented a secondary archive system with a staff of one to three programmers over a fifteen month period. The initial release was operating after only seven months of design, development and testing. Though still in its infancy, Archer is satisfying the most pressing needs of the GSFC DAAC.

While Archer provides only a subset of the functions available with COTS software like UniTree, it supports migration between near-line and off-line media and offers good performance and flexibility. By selecting tar as tape format, Archer makes data more portable between Unix systems.

References

[1] Architecture and Evolution of Goddard Space Flight Center Distributed Active Archive Center, Jean-Jacques Bedet, Wayne Rosen, Mark Sherman, Hughes STX; Phil Pease, NASA/Goddard Space Flight Center, NASA Conference Publication 3295, March 28-30, 1995.

NEXT
DOCUMENT

The Global File System[1]

Steven R. Soltis, Thomas M. Ruwart, Matthew T. O'Keefe
Department of Electrical Engineering
and
Laboratory for Computational Science and Engineering
University of Minnesota
4-174 EE/CS Building
Minneapolis, MN 55455
soltis@ee.umn.edu
Tel: 612-625-6306
Fax: 612-625-4583

Abstract

The Global File System (GFS) is a prototype design for a distributed file system in which cluster nodes physically share storage devices connected via a network-like Fibre Channel. Networks and network-attached storage devices have advanced to a level of performance and extensibility so that the previous disadvantages of *shared disk* architectures are no longer valid. This shared storage architecture attempts to exploit the sophistication of storage device technologies whereas a server architecture diminishes a device's role to that of a simple component. GFS distributes the file system responsibilities across processing nodes, storage across the devices, and file system resources across the entire storage pool. GFS caches data on the storage devices instead of the main memories of the machines. Consistency is established by using a locking mechanism maintained by the storage devices to facilitate atomic read-modify-write operations. The locking mechanism is being prototyped on Seagate disk drives and Ciprico disk arrays. GFS is implemented in the Silicon Graphics IRIX operating system and is accessed using standard Unix commands and utilities.

Introduction

Distributed systems can be evaluated by three factors: performance, availability, and extensibility. Performance can be characterized by such measurements as response time and throughput. Distributed systems can achieve availability by allowing their working components to act as replacements for failed components. Extensibility is a combination of portability and scalability. Obvious influences on scalability are such things as addressing limitations and network ports, but subtle bottlenecks in hardware and software may also arise.

These three factors are influenced by the architecture of the distributed and parallel systems. The architectures can be categorized as *message-based* (*shared nothing*) and

[1] This work was supported by the Office of Naval Research under grant no. N00019-95-1-0611, by the National Science Foundation under grant ASC-9523480, and by grant no. 5555-23 from the University Space Research Association which is administered by NASA's Center for Excellence in Space Data and Information Sciences (CESDIS) at the NASA Goddard Space Flight Center.

shared storage (*shared disk*) Message-based architectures share data by communication between machines across a network with the data stored locally on devices within each machine. Machines in the shared storage architecture access all storage devices directly. Figures 1 and 2 show examples of message-based and shared storage architectures [1][2].

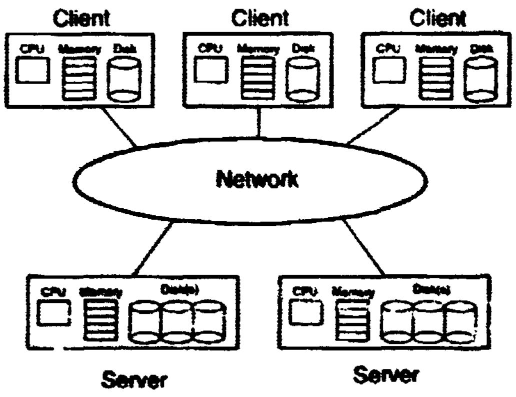


Figure 1: Message-Based

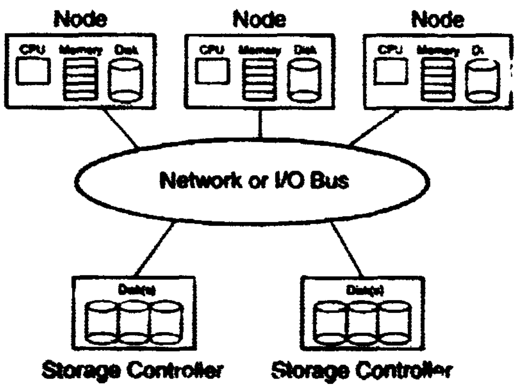


Figure 2: Shared Storage

Advocates of both architectures claim the advantage with respect to these three factors. This is a techno-religious war that will not be resolved any time soon, yet analyzing existing systems gives perspective on the strengths and weaknesses of each architecture. This next section summarizes a number of distributed file systems based on their data sharing approaches.

Message-based Distributed File Systems

Sun Network File System

The Sun Network File System (NFS) was designed by Sun Microsystems in 1985 [3]. It's design goals were system independence, name transparency, and preservation of Unix file system semantics. NFS uses a client-server approach. The server is stateless and writes modified data to stable storage before returning results. The server is able to cache data in its system memory to improve performance. The clients make requests to the server with all information necessary to complete the operation. Clients and servers communicate over a network using remote procedure calls (RPC). The RPC is a high level protocol built upon User Datagram Protocol (UDP) and Internet Protocol (IP).

The statelessness of the server eases crash recovery. A client that goes down does not effect the operations of the server or other clients. A server that fails need only to reboot. The clients will resend requests when the server has not completed their requests in a given time. The clients perceive the server as being slow but they are unaware that it has rebooted.

Sprite File System

Sprite is a distributed operating system for networked workstations developed under the Symbolic Processing Using RISCs (SPUR) research project [4]. Like Sun NFS, Sprite uses remote procedure calls to communicate across its network. Sprite's file system is distributed across multiple servers and clients. It's primary goal was to provide name transparency while still providing adequate performance. Even device special files are accessible to any process on the network.

The Sprite file system maintains cache consistency using a server-initiated approach. The server tracks open files. When files are non-write-shared, the clients may cache the portions of a file within their local memories. When a file moves from non-write-shared to write-shared, the server performs a call-back operation and disables client caching.

Andrew and Coda File Systems

Carnegie Mellon University's Coda is a distributed file system descended from the Andrew file system which was a joint research project between IBM and CMU [5][6]. Both file systems are designed to operate on a distributed network of workstations scaling up to 5000 machines.

Coda (constant data availability) was designed to improve on the availability of Andrew. Each client is able to cache entire files locally in its memory and disks. Furthermore, multiple copies of each file may exist on several servers. A server failure may then have little impact on availability. This approach also allows clients to run in *disconnected operation* using only the files it has cached locally. The client can reconnect to the network and synchronize its cache with the rest of the system.

Like the Sprite file system, Coda servers maintain state concerning file accesses. The servers are responsible for performing call-backs when a clients cached data has been modified by another client. File sharing on a client is guaranteed to have consistency described by Unix file sharing semantics. Files shared across different systems see consistency at the granularity of the entire file.

xFS Serverless Network File System

The xFS file system is part of the Berkeley's Network of Workstations (NOW) project [7]. It is a successor to some of the research from the Sprite project. It uses a log structured approach like Sprite's Log-structured File System (LFS) and Zebra's [8] striping technique to simplify failure recovery and provide high throughput transfers.

In xFS workstations are connected by a fast, switched network. xFS is said to be serverless, since the storage server functionality can be placed on the same machines as a client. Hence, any system can manage control, metadata, and real data. This has advantages of load balancing, scalability, and high availability.

Like Sprite, the system supports data caching on the clients [9]. The client requests data from a manager. This manager tries to satisfy the request from another client's cache; otherwise it directs the client the appropriate storage device. xFS uses a token based cache consistency mechanism. A client must acquire a token for each file system block that it wishes to modify. The managers notify the clients to invalidate their stale copies of the data and forward their requests to the new owner of the token.

Shared Storage Distributed File Systems

Digital's VAXClusters VMS

The VAXcluster is a "closely coupled" structure of VAX computing and storage nodes that operates as a single system. This system had VAX nodes connected by a message-based interconnect. Each processor runs the same copy of the distributed VAX/VMS operating system. The interconnection network had two topologies: the high performance Star Coupler hub that supported a maximum of 16 devices and a low cost Ethernet network [10].

The storage devices are connected to the system through a Hierarchical Storage Controller (HSC). For high-reliability, another HSC could be placed between the dual ported storage devices and the star coupler by adding a redundant path between the CPUs and the storage devices.

The operating system allows files to be shared using the cluster's distributed lock manager. Lock requests are made for a particular access mode: exclusive access, protected read, concurrent read, or concurrent write. Incompatible requests for resources are queued until the resource is unlocked. This system is a shared storage architecture, since all file requests are serviced from the shared HSCs. Each HSC can support up to 32 disks.

VMS allows caching for data ana file system resources. Coherence is maintained between the CPU's local memories by sequence numbers within the files' synchronization locks. When the file system modifies a block, it increments the sequence number in the file's lock. If another system has this block cached and later references it, this system will find that it's sequence number is old. The block will be refreshed from the HSC.

Cray's Serverless File System

Cray Research's Serverless File System (SFS) is a file system incorporated in their UNICOS operating system [11]. The file system uses a HIPPI disk array as the shared storage device. The disk array is connected to four C90 machines through a HIPPI switch. All C90 machines (nodes) act as peers; there is no server machine.

Arbitration of the HIPPI disk array is performed on a Sun SPARC workstation which is also connected to the HIPPI switch. This workstation, call the *HippiSeMaPhore* (HSMP) is responsible for maintaining semaphores used for mutual exclusion of data stored on the disk array. It also has error recovery functions.

Cray's SFS supports two types of file operations: multiple readers and single writer. SFS provides consistency by using the semaphores to facilitate read-modify-write operations as well as limiting the open file states. Nodes are able to cache data like a local file system but with the constraints of more limited parallel file operations.

Message—based Versus Shared Storage

The message-based architecture's strength lies in its extensibility. The approach is as portable as the network protocol connecting the machines and it can potentially scale to large numbers of machines. The best example of message-based portability is NFS. This file system dominates the industry, because it is available on almost every platform. File systems like Coda have shown that the message-based approach scales to thousands of machines.

Message-based systems may perform well if data access is well balanced across all machines. Load balancing is difficult since machine capacities and usage differ dynamically across the system. Locality is also difficult to maintain, since there will always be resources that are shared by many nodes in the system. Redundant copies can be maintained but at the cost of coherence overheads. Furthermore, the performance benefit of high speed devices like disk arrays is negated, since the bandwidth to each machine is limited by the network. To summarize, achieving good performance on message-based systems is not an easy task.

Server and device failures are another challenging problem facing the message-based approach, since a server failure may result in data becoming inaccessible. Fault tolerance may be maintained using disk array devices at each node, but redundancy is not extended across machines. Software redundancy schemes must be built into the file system to maintain any fault tolerance.

The shared storage approach has the advantage that every machine has nearly uniform access to all storage devices and freedom from servicing data requests from other machines. This approach is similar to the traditional uniprocessor model where a machine acts independently. Also, failures of a machine have little effect on other systems except for possible load increases. Storage device availability can be improved using hardware RAID [12].

The shared storage architecture takes advantage of the properties of the underlying storage hardware. Since every node has uniform access to the devices, the bandwidth produced by disk arrays or disk striping can be utilized by all nodes. Also, devices capable of command queuing can optimize head seeks to provide high throughput.

The downfall of traditional shared storage system has been scalability and cost. Systems like the Digital's VAXcluster and Cray's SFS are based on proprietary networks and hardware. Proprietary hardware does not benefit from market competition and often remains costly. Furthermore, these systems do not scale with the number of nodes. In both examples, only one storage device and a few machines can be attached to the system.

So far neither architecture fully satisfies all three factors - performance, availability, and extensibility but new network technologies are changing this. For instance, *Fibre Channel* (FC) is an emerging ANSI and International Standards Organization (ISO) standard for a network architecture [13] that supports network attached storage by including the SCSI-3 channel protocol. Fibre Channel provides two topologies for network attached storage: switched and arbitrated loop [14].

A high speed network like Fibre Channel can improve the performance of both shared storage and message-based architectures, yet it does little to improve the extensibility and availability of the message-based approach. Providing network attachment to storage devices greatly enhances the extensibility of shared storage. With a network like Fibre Channel, a system that is non-propriety and portable can be built using the shared storage architecture.

Existing shared storage systems must be redesigned to exploit the properties of these networks and devices. The assumptions that traditional shared storage file systems made with respect to data caching, coherence, and resource management are ob;olete. For instance, Cray's SFS caches data locally on its nodes to exploit the high bandwidth, low latency memories of the C90s. This caching comes at the price of allowing only non-write-shared file operations. That is, if the file is opened by one or more readers, a writer cannot access it until all readers close the file. This coherence mechanism can lead to large latencies and even starvation.

The Global File System

The Global File System is a prototype design for a distributed file system. Network attached storage devices are physically shared by the cluster nodes. The GFS prototype is implemented in the Silicon Graphics' IRIX operating system [15][16] under the VFS interface and is accessed using standard Unix commands and utilities [17][18][19]. The machines and storage devices are connected via a Fibre Channel network.

GFS views storage as a *Network Storage Pool* (NSP) — a collection of network attached storage devices logically grouped to provide node machines with a unified storage space. These storage pools are not owned or controlled by any one machine but rather act as shared storage to all machines and devices on the network. NSPs are divided into subpools where each *subpool* takes on the attributes of the underlying hardware.

GFS targets environments that require large storage capacities and bandwidth such as multimedia, scientific computing, and visualization [20][21]. These large capacities influence tradeoffs, such as caching and the metadata structure, associated with the design of a file system.

Chip integration has transformed storage devices into sophisticated units capable of replacing many of the functions performed by a server machine in a client-server environment. These devices can schedule accesses to media by queuing multiple requests. They possess caches of one or more Megabytes that can be used for read and write caching and prefetching.

GFS caches data in tue nodes' main memories only during I/O request processing. After each request is satisfied, the data is either released or written back to the storage devices. To exploit locality of reference, GFS caches data on the storage devices. GFS informs the devices on each request what data is appropriate to cache - such as metadata that is accessed repetitively and small files like directories which are frequently accessed. Consistency is established by using a locking mechanism maintained by the storage devices to facilitate atomic read-modify-write operations. This fcrm of locking has the simplicity of a centralized mechanism yet is distributed across a large number of devices.

Figure 3 represents an example of the GFS distributed environment. The nodes are attached to the network at the top of the figure and the storage pool at the bottom. Connecting the nodes and the devices is a Fibre Channel network which may consist of switches, loops, and hubs. In the example, three different subpools exist: */single* is a single disk, */wide* is a striping of several disks, and the */fast* is a disk array.

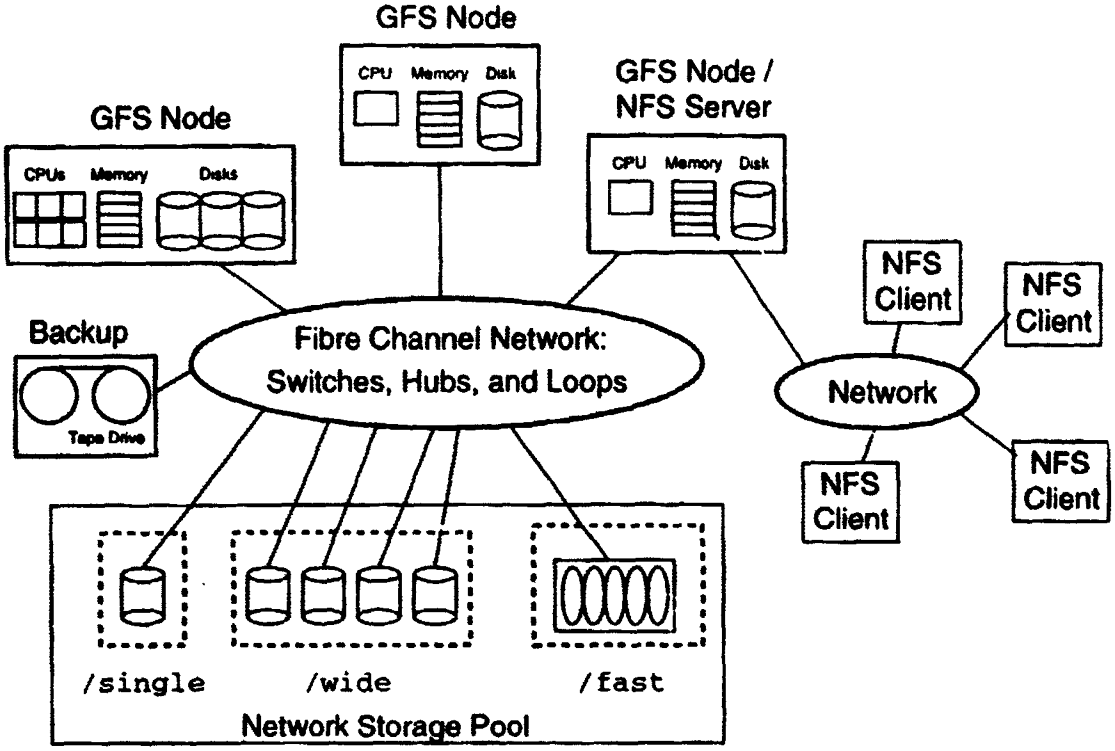


Figure 3: GFS Distributed Environment

To the figure's left is a tape device which is directly connected to the network. Such a tape drive may be used for data backup or hierarchical storage management. A node could initiate third party transfers between the disk devices and the tape drive. The figure also shows how a GFS host can act as a NFS server. This ability allows machines without GFS capabilities to access GFS data via an NFS exported file system. The operating systems VFS interface handles the translation between GFS and NFS.

File System Structure

Each GFS file system is divided into several *Resource Groups* (RG). Resource groups are designed to distribute file system resources across the entire storage subpool. Figure 4 shows the logical structure of a file system. Multiple RGs exist per device and can be striped across several devices.

Resource groups are essentially *mini-file systems*. Each group has a *RG block*, data bitmaps, dinode bitmaps (used as the dinode free list), dinodes, and data blocks. The RG block contains information similar to what traditional superblocks maintain: the number of free dinodes, the number of free data blocks, and the access times of the RG. File data and metadata may span multiple groups.

GFS also has a superblock which contains information that cannot be distributed across the resource groups. This information includes the number of nodes mounted on the file system, bitmaps to calculate unique identifiers for each node, and the file system block size. The superblock also contains a static index of the RGs. This RG *index* describes the location of each group as well as the group attributes and layout.

A GFS dinode takes an entire file system block. Each dinode is divided into a header section which contains standard dinode fields and a section of pointers. The number of pointers in a dinode is the determined by equation 1 and the number of pointers an indirect block has is given in equation 2.

$$\text{\# pointers in dinode} = \frac{\text{file system block size - size of dinode header}}{\text{size of block address}} \tag{1}$$

$$\text{\# pointers in indirect block} = \frac{\text{file system block size}}{\text{size of block address}} \tag{2}$$

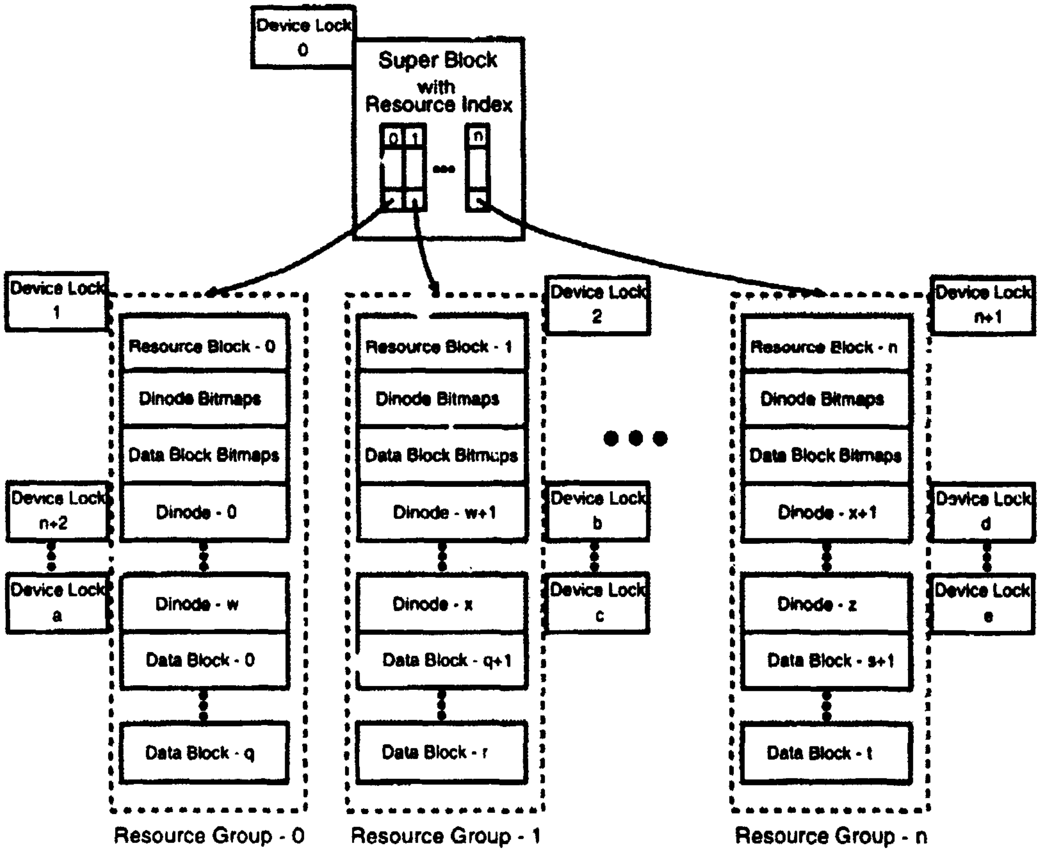


Figure 4: GFS Structure

The pointers form a tree structure in which all data blocks are at the tree's leaves and have the same height. This structure is used so that all its accesses require the same number of indirect accesses to the data blocks regardless of the offset into the file. This structure differs from the traditional Unix file system (UFS) where data blocks might have different heights. The UFS tree is simpler to implement yet can require an additional level of indirection. Furthermore, UFS places multiple dinodes per file system block. By taking an entire block, the GFS can have hundreds of direct pointers in the dinode instead of just ten as in a UFS dinode. Figure 5 illustrates a GFS dinode and one level of indirection for referencing the data blocks.

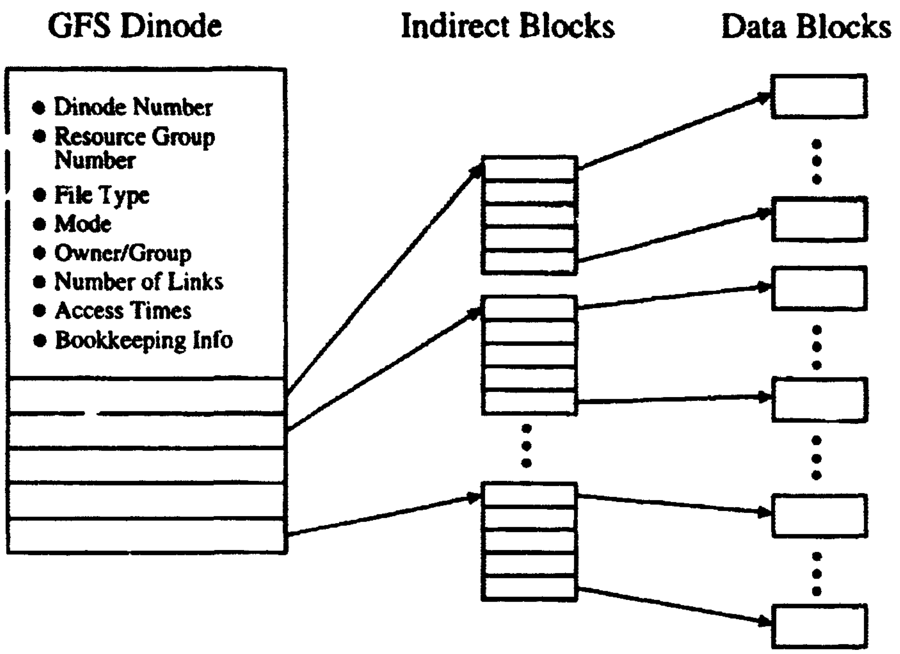


Figure 5: GFS Dinode

Device Locks

Device Locks are mechanisms for node machines to maintain mutual exclusion of file system data. They are implemented on the storage devices and accessed with a single SCSI command. The *Dlock* command instructs the devices to perform primitive operations on the locks - *test and set* and *clear*. The implementation of the device locks on the device are limited by the following constraints:

1. The device lock commands are independent of all other SCSI commands.

2. Devices supporting device locks have no awareness of the nature of data or resource that is locked for mutual exclusion.

3. Each lock requires minimal amounts of device memory - as little as one byte per lock.

Lock States

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 a machine node. If the bit is 0, the lock is available to be acquired by any node. The Dlock command action *test and set* first determines if the lock value is 1. If value is 1, the command returns with a status indicating that the lock has already be acquired. If the value is 0, Dlock sets the lock to 1 and returns a good status to the initiator. The Dlock command *clear* simply sets the lock bit to 0.

Clocks

Associated with each lock is a clock. The clocks are logical clocks in the sense that they do not relate to time but instead keep an ordering of events for each lock. These clocks are incremented when a successful action is performed. The clocks are used to monitor how often a lock is accessed; i.e., how many times the lock has been set and then cleared. Such a clock gives insight into load balancing ***hot-spots***. These occur when some locks are accessed more often than others. More importantly, these clocks are useful for error recovery.

The clocks are implemented using a minimal amount of memory — typically 7 to 16 bits each. The initiators must be aware that the clock values periodically roll-over from their maximum value to zero. This may happen several times a second on a highly accessed lock, so care should be taken by the initiator not to assume that the clock value is slowly growing. The clock value is returned after each Dlock command.

Device Failures

The device locks and their accompanying clocks are stored in volatile memory on the device, although the locks are held across SCSI resets. When a device is powered on or a failure occurs which results in the locks being cleared, the device notifies all nodes by setting *Unit Attention* . Upon finding a unit attention, a node checks to see if its locks are still valid. Before proceeding, it will then re-acquire any locks that may have been lost.

Node Failures

A node that fails could leave device locks in the locked state indefinitely. These locks will remain in this state until some node clears them. A node attempting to acquire a lock that is owned by a failed node can identify that the lock has been untouched by checking the activity of the lock's clock. If the clock has remained unchanged for an extended time period, a node can identify such a case and clear the lock.

Care must be taken by the node clearing a lock that it does not own. The true owner may have failed or it may be in a *hung* state from which it will eventually return still believing it owns the lock. Furthermore, two separate nodes may simultaneously identify the same lock which must be cleared and send resets. It may be possible that the first node clears the lock and sets the lock in the following command. The second node which has already decided to clear the lock sends the command after the lock has been acquired by the first node. This second clear request must be ignored.

When a node wishes to clear a lock as failure recovery, the device compares the current clock with the input clock from the node. This test ensures that the lock will only be cleared if the node can identify the current value of the clock.

Deadlocks and Starvation

Deadlocks are avoided by the file system. The file system only acquires locks in an increasing order. Circular dependencies are avoided. Starvation is handled by the file system and device drivers. The file system does not hold locks for more than a few I/O requests to storage. A node's device drivers test for its own starvation by checking the activity of the lock-based clock values. The node can increase the rate at which lock requests are performed an in attempt to feed its starvation.

Consistency and Caching

Consistency is maintained by using atomic operations guaranteed by the device locks when modifying data. Given the limited number of practical device locks per device - on the order of 1024 - individual locks cannot be assigned to each file. One lock is assigned to the super block, one lock is assigned to each resource group, and the remaining locks are divided among the dinodes. Figure 4 shows how device locks are associated with the superblock, resource groups, and dinodes.

When device locks are not implemented on the storage device, the SCSI commands *Reserve* and *Release* can be used to perform atomic operations on data. These commands provide exclusive access to the entire device for one node by not servicing requests from other nodes. These commands guarantee exclusive access but do not provide much parallelism. With only one reservation per device, many non-conflicting requests have to wait until the storage device is released. In a distributed environment, such limited access decreases system throughput and response times.

The SCSI protocol describes the optional commands *Reserve* and *Release* on *Extents*. These commands allow initiators to reserve for exclusive access only the data blocks that they may need. These commands decrease the granularity of exclusion from the device level to the block level. While potentially increasing the throughput of the distributed system, Reserve and Release on Extent commands require the devices to maintain complicated states of access permissions. For this reason, these commands are generally not implemented by the majority of device manufacturers.

We present device locks as a mutual exclusion mechanism that is highly parallel, has low device overheads, and recovers from failures gracefully. The Dlock command is being prototyped on Seagate disk drives and Ciprico disk arrays.

Preliminary Results

Preliminary measurements have been taken with parallel SCSI hardware instead of Fibre Channel. Fibre Channel hardware is not yet available to us, so we present the parallel SCSI results instead. We hope to have Fibre Channel measurements by the time this paper is presented. These results are based upon tests using SCSI ***Reserve*** and ***Release*** to maintain consistency instead of device locks. The Dlock command is still in development and testing by Ciprico and Seagate.

Test Configuration

These tests were conducted on three SGI Indys running IRIX 5.3 operating system. One Indy has a 100 MHz R4600 processor while the other two have 132 MHz R4600 processors. All machines have 32 Mbytes of main memory.

A Seagate Barracuda 2LP was used as the shared storage device. The 2LP is a 2 Gigabyte Fast SCSI-2 drive. It has a 17 millisecond maximum seek time and a 7200 RPM spindle speed. Our tests measured the time for reserve and release commands to be approximately 2 milliseconds. This disk and all three system disks share the common bus. The disk caching parameters are set at their default values.

The configuration shown in figure 6 is a representation of the test environment. To overcome cabling difficulties, the system disk of node 2 was attached to the internal SCSI bus of node 0 and node 2's controller was directly connected to the internal bus of node 1. All devices and controllers are electrically connected.

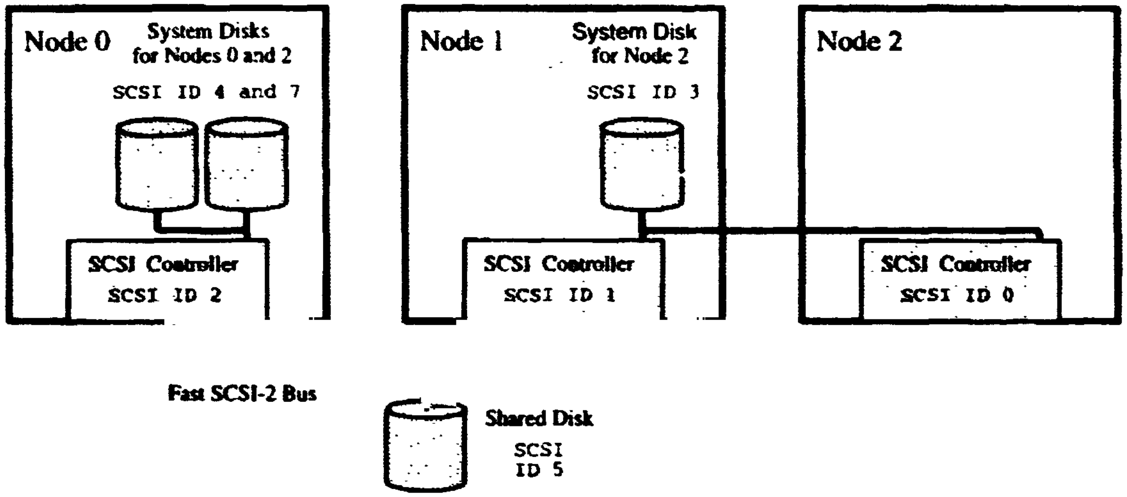


Figure 6: GFS Test Environment

Benchmarks

Small benchmark programs were run on one, two, and all three machines to study how effectively the SCSI bus and disk are shared. The tests were run on the file system running in user space as opposed to running under the VFS interface of the kernel. The user level file system allows for easy tracing and performance measurements. The user level file system adds various overheads to the system, so we believe that the VFS file system will perform even better.

The benchmarks chosen involve tests of creating, writing, and reading files. The file sizes range from 1 Mbyte to 16 Mbytes. Files were written and read by making from 1 to 128 requests per file A delay was placed between each request ranging from zero to one second. This delay represents the time for which a real application would perform computation or wait for some reason other than I/O. All tests were run five times so that the median values could be used tc evaluate the performance. Table 1 summarizes these benchmarks. The benchmarks chosen for these tests attempt to match the performance capabilities of the single shared disk and 10 MB/sec SCSI bus. These benchmarks will be scaled appropriately when performing the tests on Fibre Channel hardware with multiple devices.

Parameter	Range
Number of Nodes	1,2,3
Types	Create and Write, Write, Read
File Sizes	1 MB to 16 MB
Number of Requests per File	1 to 128
Delay Between Requests	0 ms, 100 ms, 500 ms, 1000 ms

Table 1: Benchmark Parameters

Parallel SCSI Performance

Figures 7 and 8 show the speedups for each machine creating and reading a 1 MB file in 128 KB requests, respectively. Figures 9 and 10 show the speedups where machines create and read a 16 MB file in 1 MB requests, respectively. These results are scaled to reflect equal amounts of work of 3 MB and 48 MB respectively. That is, the time for one machine is multiplied by 3 and the time is multiplied by 1.5 for two machines. All these times are then normalized to the one machine test by dividing by the one machine time. Curves are given for no delay and 100 ms, 500 ms, and 1000 ms delays. The write tests show trends similar to the read tests.

Figures 11, 12, 13, and 14 are plots of the 16 MB creates with varying request sizes. Figures 15, 16, 17, and 18 are plots of the same test with read requests. The request axis is the number of requests needed to access the entire file. The three curves for each plot are the time for one machine alone, the time of the slowest machine with three machines running simultaneously, and three times the one machine time. This last curve is given as a constant workload comparison to the three machine case.

Figures 19 and 20 are plots of the number of conflicts encountered by machines when running the 16 MB create and read tests and no delay between requests. Figures 21 and 22 show the same tests with a 1000 ms delay. These conflicts were counted by the device driver whenever a reserve command failed because the device was already reserved.

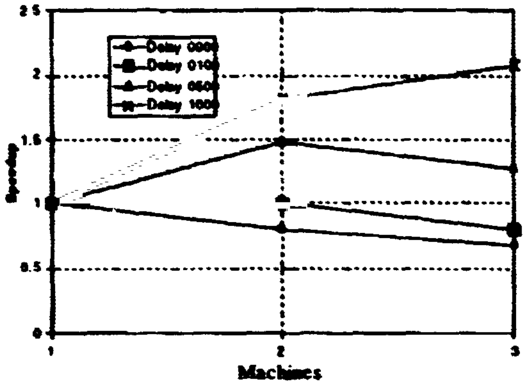


Figure 7: GFS Speedup for 1 MB files Created by 128 KB Requests

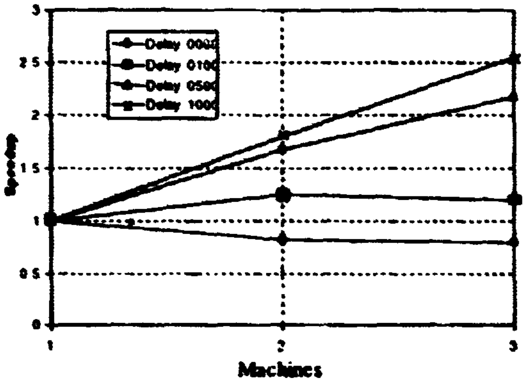


Figure 8: GFS Speedup for 1 MB files Read by 128 KB Requests

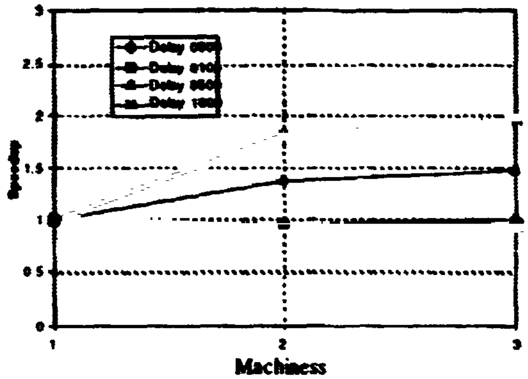


Figure 9: GFS Speedup for 16 MB files Created by 1 MB Requests

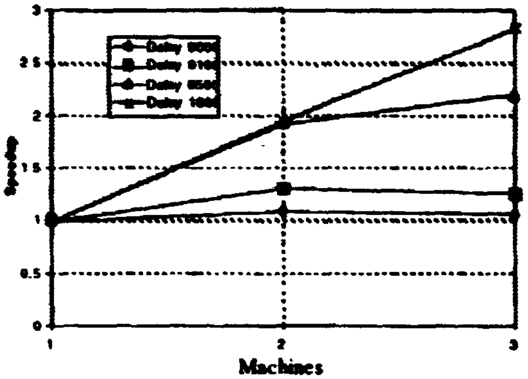


Figure 10: GFS Speedup for 16 MB files Read by 1 MB Requests

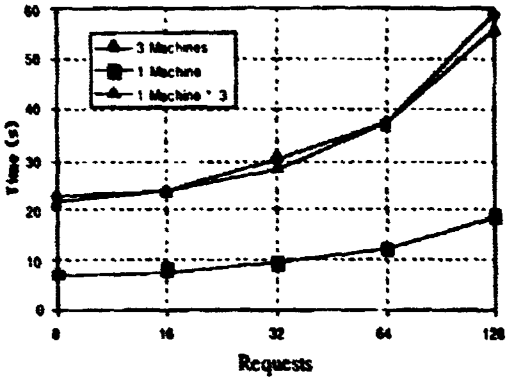


Figure 11: GFS Creates of 16 MB files with no delay

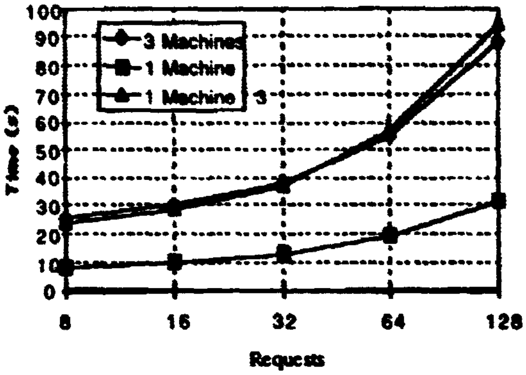


Figure 12: GFS Creates of 16 MB files with 100 ms delay

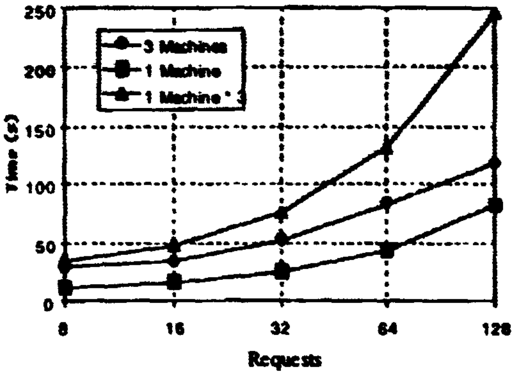


Figure 13: GFS Creates of 16 MB files with 500 ms delay

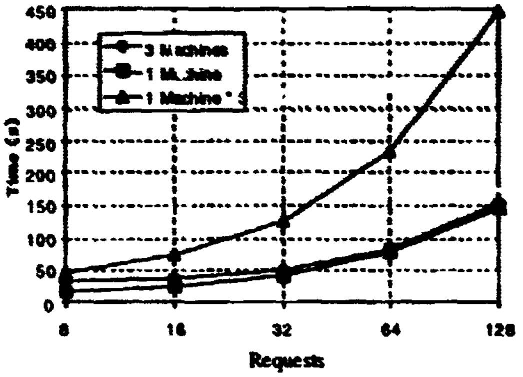


Figure 14: GFS Creates of 16 MB files with 1000 ms delay

The conflict plots show several obvious but interesting trends: the single machine tests had no conflicts; the three machines tests had more conflicts than two machines test; a delay between requests deceased the conflicts; and creates had more conflicts than reads. These trends can be explained by the argument that increasing numbers or the rate of requests increases the chances of having conflicts. The tests that had the most conflicts are those which issued the most requests in a given time period.

The results are promising considering the nature of the tests. The no delay case represents nearly constant access to the device in the single machine case. No parallelism can be exploited by adding one or two machines, since the device is already fully utilized. The slowdown seen in some plots is a result of the contention for the shared device.

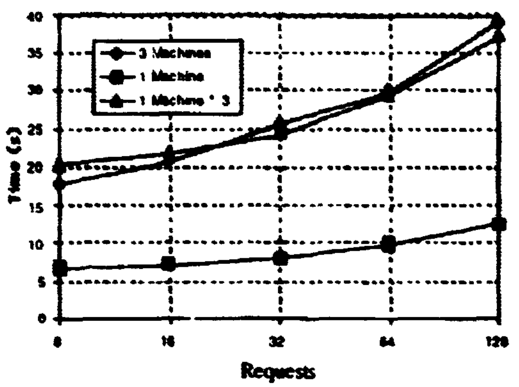


Figure 15: GFS Reads from 16 MB files with no delay

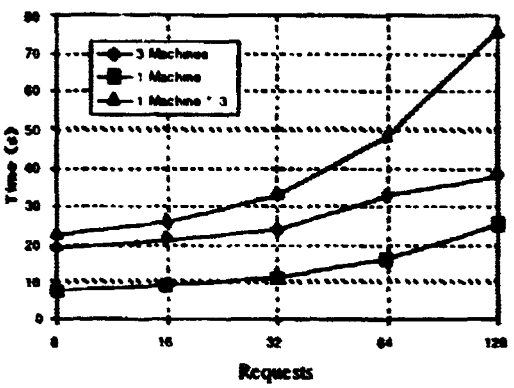


Figure 16: GFS Reads from 16 MB files with 100 ms delay

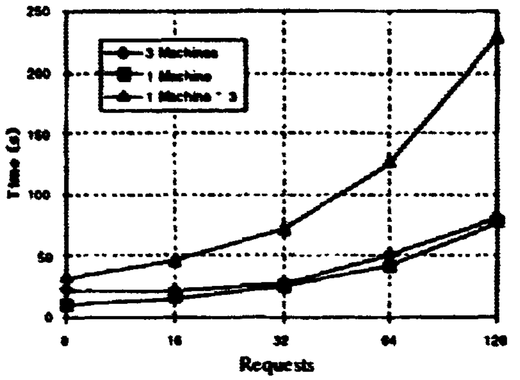


Figure 17: GFS Reads from 16 MB files with 500 ms delay

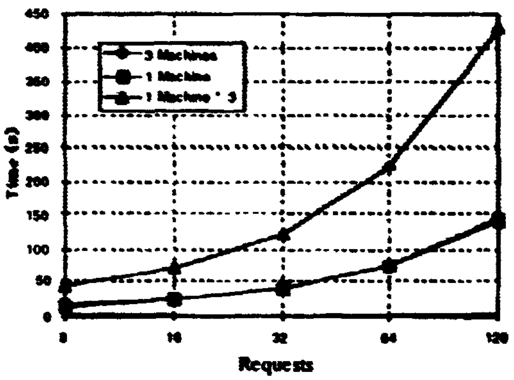


Figure 18: GFS Reads from 16 MB files with 1000 ms delay

File creates are slower than the reads because the creates require additional I/O requests to allocate the dinodes, allocate data blocks, and build the metadata tree. As can be seen in the plots with the number of requests as the X-axis, the number of requests is indirectly proportional to the performance. This is because each request has an overhead of greater that 2.5 milliseconds. Also, with each request there is a possibility of a reservation conflict which slows the request by as many as 100 milliseconds. With Fibre Channel and device locks, both these overheads will be substantially reduced.

The 100 ms delay tests allow some parallelism to be realized. However, it is not until the 500 ms delay that the parallelism is exploited for all three machines. The 1000 ms delay may represent the best speedup for this configuration, since according to figure 14 and 18, the 1000 ms delay test does not distinguish between one, two, or three machines running simultaneously.

Striping the file system across multiple devices may have an effect similar to increasing the delay between file system requests. Assuming the 10 MB/sec SCSI bus is not a system bottleneck, adding another device to the configuration may remove as much as 50 percent of the burden from the single disk.

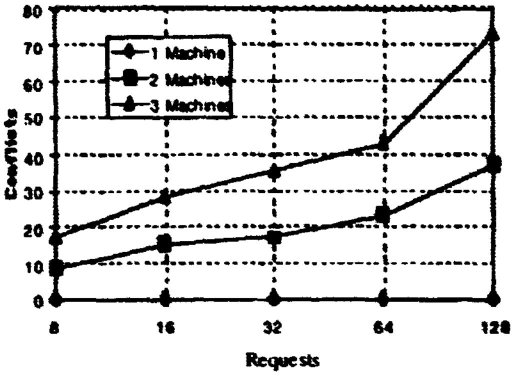


Figure 19: GFS Creates of 16 MB files with no delay

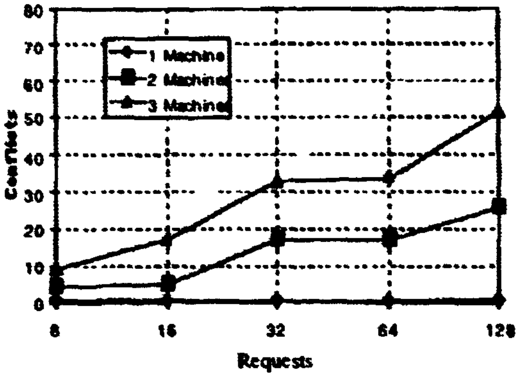


Figure 20: GFS Reads from 16 MB files with no delay

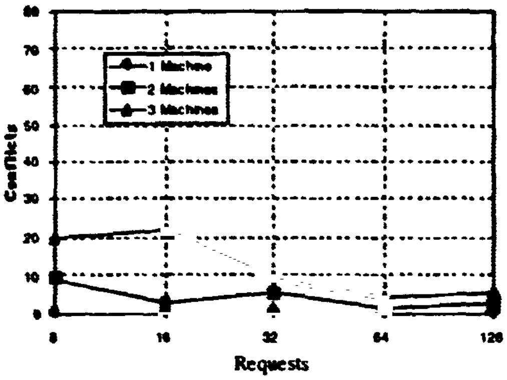


Figure 21: GFS Creates of 16 MB files with 1000 ms delay

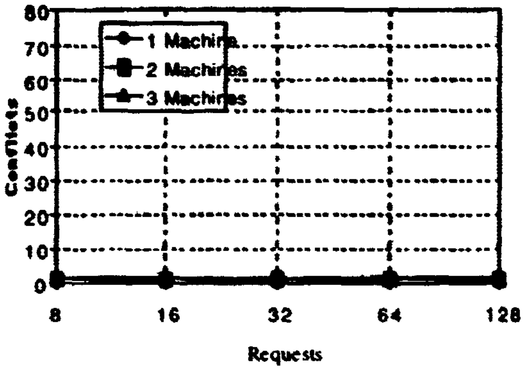


Figure 22: GFS Reads from 16 MB files with 1000 ms delay

NFS Comparison

The benchmarks were also run on a configuration using NFS as a comparison between distributed file systems running on the same hardware. This configuration consists of a dedicated server machine and three clients. The workstations and disk are the same as above; the server machine is a 132 Mhz Indy. Connecting the machines is a 10 Mbit/sec ethernet network. The server's file system was SGI's XFS [22].

Figures 23, 24, 25, and 26 are the NFS equivalent of the speedup curves given above. Figures 27, 28, 29, and 30 are given as a time comparison between the NFS and GFS read tests. A few differences can be noticed between the GFS and NFS tests. First, while both file system have good speedup for larger delays, NFS never has slowdown. However, the NFS tests use a dedicated server which is one more machine than the GFS test configuration. The speedup values do not take this into account. Second, the GFS times are much smaller - between two and ten times faster. In cases where the request sizes were large, GFS exceeded transfer speeds of 2 MB/sec.

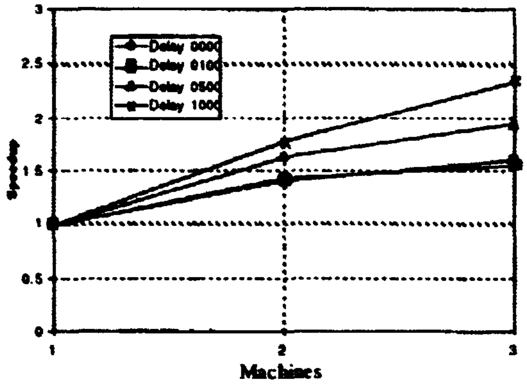


Figure 23: NFS Speedup for 1 MB files Created by 128 KB requests

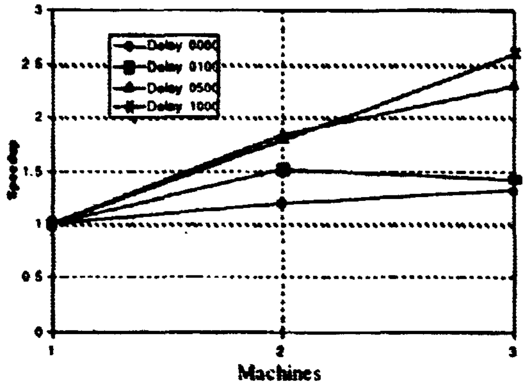


Figure 24: NFS Speedup for 1 MB files Read by 128 KB requests

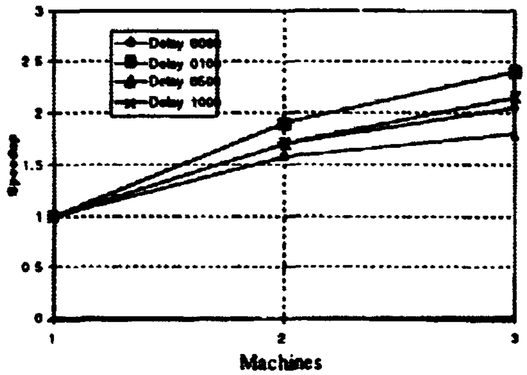


Figure 25: NFS Speedup for 16 MB files Created by 1 MB requests

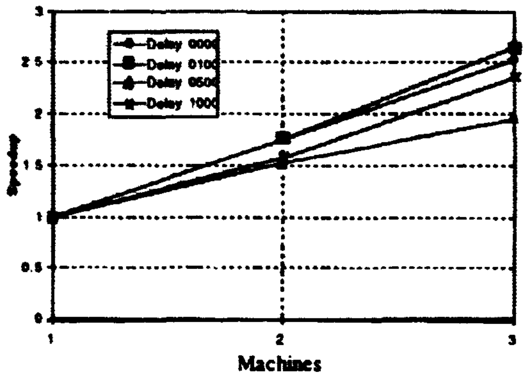


Figure 26: NFS Speedup for 16 MB files Read by 1 MB requests

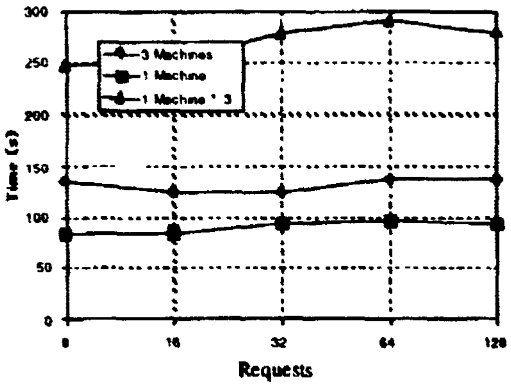


Figure 27: NFS Creates of 16 MB files with no delay

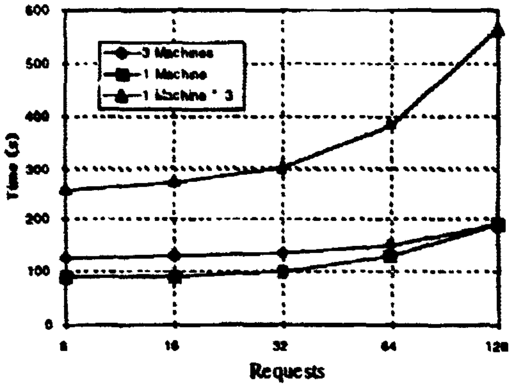


Figure 28: NFS Creates of 16 MB files with 1000 ms delay

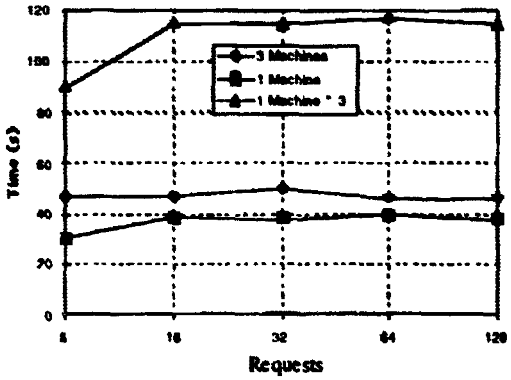


Figure 29: NFS Reads from 16 MB files with no delay

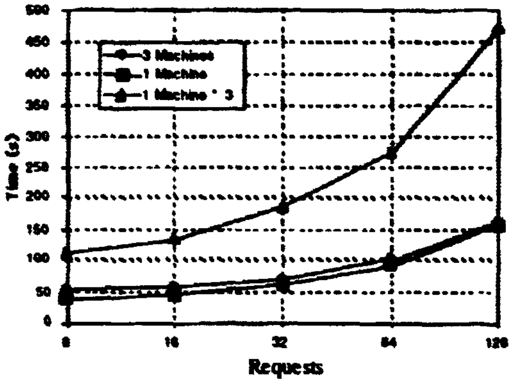


Figure 30: NFS Reads from 16 MB files with 1000 ms delay

Future Work

The GFS project is sti^l in its early phase. When Fibre Channel hardware and device drivers become available, an expanded set of performance tests will be performed. These tests will be performed on file systems with different configurations - file system block sizes, resource group layouts, etc.

The device lock mechanisms are to be studied to accommodate failure recovery. The polling rates between locking retry requests must be tuned to provide a good compromise between low latencies to each node and high system throughput. Associated with these rates is the delay between retry algorithm - constant delay versus variable delay.

The time that a node waits before resetting a device lock owned by another node must also be investigated. This time duration has to accommodate failures in a short period without resetting prematurely. Premature resets cause the previous owner to rebid for locks it once owned. This is acceptable occasionally but should be kept to a minimum.

Currently GFS stores data in a linear layout across the storage devices. This layout allows various forms of striping and data partitioning, but we plan to generalize this layout to a subpool architecture. Each subpool will have attributes reflected by its underlying hardware and configuration. A translation layer will be placed between the file system and device drivers. This translation layer will convert the linear block addresses from the file system to the proper devices and block numbers padding requests when appropriate.

Work has begun to study the caching algorithms and cache configurations of storage devices. Using hardware and simulations, we are attempting to determine the benefit of large caches. Replacement policies are also being studied.

Conclusion

The GFS approach to a distributed file system using shared storage devices seems promising given the high bandwidth natures of new networks and the increasing sophistication of devices. The architecture places more responsibilities on storage devices than message-based architectures. Modern devices are able to cache, perform mutual exclusion, and schedule requests freeing these burdens from the node machines.

The results from the preliminary performance measurements indicate that with sufficient delay between I/O requests to a shared device, device parallelism is exploited within the system. This delay may take the form of machines performing file system requests between lengthy computation or low activity. Striping the file system across multiple devices may have an effect similar to increasing the delay between requests.

We believe that by using 100 MB/sec Fibre Channel and multiple storage devices, this shared storage scheme will scale well to several machines even with large workloads. Furthermore, the fine grain mutual exclusion implemented using the device locks will decrease conflicts to further increase the performance of each node and the system.

Acknowledgments

We thank Ben Gribstad at the University of Minnesota for his help with device driver development and performance evaluations. He contributed a lot to the preliminary results portion of this paper. We also thank Aaron Sawdey from the University of Minnesota for his advice and experience while building the parallel SCSI test configuration used during development and preliminary tests.

The SCSI Device Locks represent input from a number of individuals. Much of the design and definition is a result of dialogues with Ciprico and Seagate. We thank Raymond Gilson, Steven Hansen, and Edward Soltis of Ciprico, and Jim Coomes, Gerald Houlder, and Michael Miller of Seagate. Finally, we thank Carl Rigg and Brad Eacker at Silicon Graphics for granting access to their device driver code which aided in the development of GFS.

References

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

[2] G. F. Pfister, *In Search of Clusters*. Upper Saddle River, NJ 07458: Prentice-Hall, Inc., 1995.

[3] 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.

[4] J. Ousterhout, A. Cherenson, F. Douglis, M. Nelson, and B. Welch, "The Sprite Network Operating System", *IEEE Computer*, pp. 23-25, February 1988.

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

[6] M. Satyanarayanan, "Coda: A Highly Available File System for a Distributed Workstation Environment", *Proceedings of the Second IEEE Workshop on Workstation Operating Systems*, September, 1989.

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

[8] J. Hartman and H. Ousterhout, "Zebra: A striped network file system," *USENIX Workshop on File Systems*, May 1992.

[9] M. Dahlin, C. Mather, R. Wang, T. Anderson, and D. Patterson, "A Quantitative Analysis of Cache Policies for Scalable Network File Systems", *Proceedings of the 1994 SIGMETRICS Conference*, May 1994.

[10] Digital Technical Journal, *VAXcluster Systems*, September 1987. Special Issue - Number 5.

[11] K. Matthews, "Implementing a Shared File System on a HIPPI Disk Array", *Fourteenth IEEE Symposium on Mass Storage Systems*, pp. 77-88, 1995.

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

[13] C. Jurgens, "Fibre Channel: A connection to the future", *IEEE Computer*, pp. 82-90, August 1995.

[14] ANSI, *FC-AL Direct Attach Disk Profile (Private Loop)*, June 1995. Version 1.8.

[15] Silicon Graphics Inc., Mountain View, CA 94029, *Programming on Silicon Graphics Systems: An Overview*, 1996. Document Number 007-2476-002.

[16] Silicon Graphics Inc., Mountain View, CA 94029, *IRIX Device Driver Programming Guide*, 1996. Document Number 007-0911-060.

[17] U. Vahalia, *Unix Internals: The New Frontiers*. Upper Saddle River, NJ 07458: Prentice-Hall, Inc., 1996.

[18] M. Bach, *The Design of the Unix Operating System*, Englewood Cliffs, NJ 07632: Prentice-Hall, Inc., 1986.

[19] B. Goodheart and J. Cox, *The Magic Garden Explained*, Englewood Cliffs,NJ 07632: Prentice-Hall, Inc., 1994.

[20] P. Woodward, "Interactive Scientific Visualization of Fluid Flow", IEEE Computer, pp. 13-25, October 1993.

[21] A. L. Reddy and J. C. Wyllie, "I/O Issues in a Multimedia System", IEEE Computer, pp. 69-74, March 1994.

[22] Silicon Graphics Inc., Mountain View, CA 94029, *IRIX Admin: Disks and Filesystems*, 1996. Document Number 007-2825-001.

NEXT
DOCUMENT

Distributed Large Data-Object Environments: End-to-End Performance Analysis of High Speed Distributed Storage Systems in Wide Area ATM Networks

William Johnston, Brian Tierney, Jason Lee, Gary Hoo, and Mary Thompson

Imaging and Distributed Computing Group,
Information and Computing Sciences Division
Lawrence Berkeley National Laboratory[1],
University of California, Berkeley, CA, 94720

Abstract

We have developed and deployed a Distributed-Parallel Storage System (DPSS) in several high speed ATM WAN testbeds to support several different types of data-intensive applications. Architecturally the DPSS is a network striped disk array, but is fairly unique in that its implementation allows applications complete freedom to determine optimal data layout, replication and/or coding redundancy strategy, security policy, and dynamic reconfiguration.

In conjunction with the DPSS, we have developed a "top-to-bottom, end-to-end" performance monitoring and analysis methodology that has allowed us to characterize all aspects of the DPSS operating in high speed WAN environments. In particular, we have run a variety of performance monitoring experiments involving the DPSS in the MAGIC testbed, which is a large-scale, high-speed, ATM network and we describe our experience using the monitoring methodology to identify and correcting problems that limit the performance of high speed distributed applications.

Finally, the DPSS is part of an overall architecture for using high-speed, wide area networks for enabling the routine, location independent use of large data-objects. Since this is part of the motivation for a distributed storage system, we describe this architecture.

[1] The work described in this paper is supported by ARPA, Computer Systems Technology Office (http://ftp.arpa.mil/ResearchAreas.html) and the U. S. Dept. of Energy, Office of Energy Research, Office of Computational and Technology Research, Mathematical, Information, and Computational Sciences Division (http://www.er.doe.gov/production/octr/mics), under contract DE-AC03-76SF00098 with the University of California. Authors: wejohnston@lbl.gov, tierney@george.lbl.gov, Lawrence Berkeley National Laboratory, mail stop: B50B-2239, Berkeley, CA, 94720, ph: 510-486-5014, fax: 510-486-6363, http://www-itg.lbl.gov). This is report no. LBNL-39064.

1.0 Introduction

We are developing a strategy for using high-speed networks as enablers for storage systems whose components are distributed around wide area networks. The high-level goal is to dramatically increase the location independence for access to "large data-objects". These objects - typically the result of a single operational cycle of an instrument, and of sizes from tens of MBytes to tens of Gbytes - are the staple of modern analytical systems. It is further the case that many of the instrumentation systems that generate such data-objects are used by a diverse and geographically distributed community: examples from the scientific community include physics and nuclear science high energy particle accelerators and detector systems, large electron microscopes, ultra-high brilliance X-ray sources, etc. There are correspondingly complex instrumentation systems in the health care community that generate large data-objects. Our approach is an architecture that uses a collection of highly distributed services to provide flexibility of managing storage resources, reliability of access, and high performance, all in an open environment where the use-conditions for resources and stored information are guaranteed through the use of a strong, but decentralized, security architecture.

In this paper we will discuss some of the aspects of our distributed large data-object architecture, but we focus on the issues for achieving high performance for distributed systems in wide-area ATM networks - a problem that is clearly central to the basic premise of our approach.

As developers of high-speed network-based distributed services, we often observe unexpectedly low network throughput and/or high latency. The reason for the poor performance is frequently not obvious. The bottlenecks can be (and have been) in any of the components: the applications, the operating systems, the device drivers, the network adapters on either the sending or receiving host (or both), the network switches and routers, and so on. It is difficult to track down performance problems because of the complex interaction between the many distributed system components, and the fact that problems in one place may be most apparent somewhere else. Further, these distributed applications are complex, bursty, and have more than one connection in and/or out of a given host at one time and simple tools like ttcp do not adequately simulate these conditions.

We have developed a methodology and tools for monitoring, under realistic operating conditions, the behavior of all the elements of the application-to-application communications path in order to determine exactly what is happening within this complex system. Our approach is to instrument both the applications and the storage systems to do timestamping and logging at every critical point in the data handling system. We have also modified some of the standard Unix network and operating system monitoring tools to log "interesting" events using a common log format that can be correlated with the instantaneous behavior of the application, the storage system, and the transport between them. This allows us to characterize the performance of all aspects of the distributed systems and network in detail, using "real-world" operations. This monitoring functionality is designed to facilitate identifying bottlenecks, performance tuning, and various sorts of network performance research. It also allows us to measure throughput and latency characteristics of our distributed application code.

The goal of the performance characterization work is to produce predictable, high-speed components that can be used as building blocks for high-performance applications, rather than having to "tune" the applications top-to-bottom as is all too common today.

In this paper we describe an architecture for handling large data-objects, the elements, implementation, and applications of that architecture. We also describe in some detail the architecture and performance of a prototype application and a distributed - parallel data server, called the DPSS (Distributed Parallel Storage Server, formerly known as the Image Server System, or ISS) that is used to drive many of the experiments, and is a key element of the large data-object architecture. Finally, we describe some techniques for monitoring and analysis of the elements of the architecture, and some experimental results using these techniques.

2.0 Distributed Large Data-Object Management Architecture

The advent of shared, widely available, high-speed networks is providing the potential for new approaches to the collection, storage, and analysis of large data-objects. In one example, high-volume health care image data used for diagnostic purposes - e.g. X-ray CT, MRI, and cardio-angiography - are increasingly collected at tertiary (centralized) facilities, and may now be routinely stored and used at locations other than the point of collection. In this case, the importance of distributed storage is that a hospital (in fact, almost any instrumentation scenario) may not provide the best environment in which to maintain a large-scale digital storage system, and an affordable, easily accessible, high-bandwidth network can provide location independence for such storage. In the case of health care, the importance of remote end-user access is that the health care professionals at the referring facility (frequently remote from the tertiary imaging facility) will have ready access to not only the image analyst's reports, but the original image data itself.

This general strategy extends to other fields as well. In particular, the same basic infrastructure is required for remote access to large-scale scientific and analytical instruments, both for data handling and for direct, remote-user operation. See [1].

The basic elements of a distributed large data-object architecture include:

- data collection and the instrument-network interface
- on-line storage that is distributed throughout the network (for both performance and reliability)
- processing elements - also distributed throughout the network - for various sorts of data analysis
- data management that provides for the automatic cataloguing (metadata generation) of the data being stored
- data access interfaces, including application-data interfaces

- (transparent) tertiary storage ("mass storage") management
- user access to all relevant aspects (application, data, metadata, data management)
- transparent security that provides access control for all of the systems components based on the resource-owner's policy

Within the network storage system in particular - a "middleware" service - the architectural issues include:

- distributed storage system operation and performance
- user access methodologies
- security architecture

These elements all need to be provided with flexible, location-independent interfaces so that they can be freely moved around the network as required for operational or other logistical convenience.

Figure 1 illustrates the overall architecture. It indicates the central role of a high-speed cache, which is used both for initial data collection, and to provide subsequent high-speed access by applications.

Briefly, the data flow and information generation proceed as follows. The data-objects are first cached on the DPSS (whose components are frequently scattered all over the network). From the cache it is "processed" as required, but typically to produce several pieces of information to be included in the "index". Metadata is generated (by analyzing the object, collecting information forwarded by the object generator, or by associating separate information with the object). This metadata is typically kept in "tagged-file" text files. "Derived" information is generated; in the case of image-like objects, this includes typically "thumbnail" and screen-sized representations, "typical" frames from a video-object, etc. The data-object itself is replicated in a tertiary storage system. All of the information related to the data-object is combined into a Web document that represents a comprehensive index and source of meta-information for the data-object. At this point the data-object has a comprehensive "index", a permanent instance in tertiary storage, and (perhaps) a temporary instance in the network cache. A Web interface can be used for searching, browsing, and accessing the metadata, or the object itself. (See Section 2.3, "Data Management, Mass Storage, and the User Interface" and Figure 3, below.) This same Web interface is used to manage the migration of the data-object in to, and out of, the cache. The user or application never has to deal directly with the tertiary storage system - it is managed in a transparent and location independent manner. Applications that access the data-object can be launched directly from the Web interface, or can use the Web interface to migrate the object to cache, and access it there. Access methods for the data-objects are typically provided as loadable libraries for the application, and provides for application or data specific views of the data-object. (These "objects" are not persistent C++ objects: the "object" consists of access methods (of which there may be several), the metadata (including "derived" objects"), and the data-object itself, all of which typically reside in different locations.) As indicated in Figure 4 (Distributed-Parallel Storage

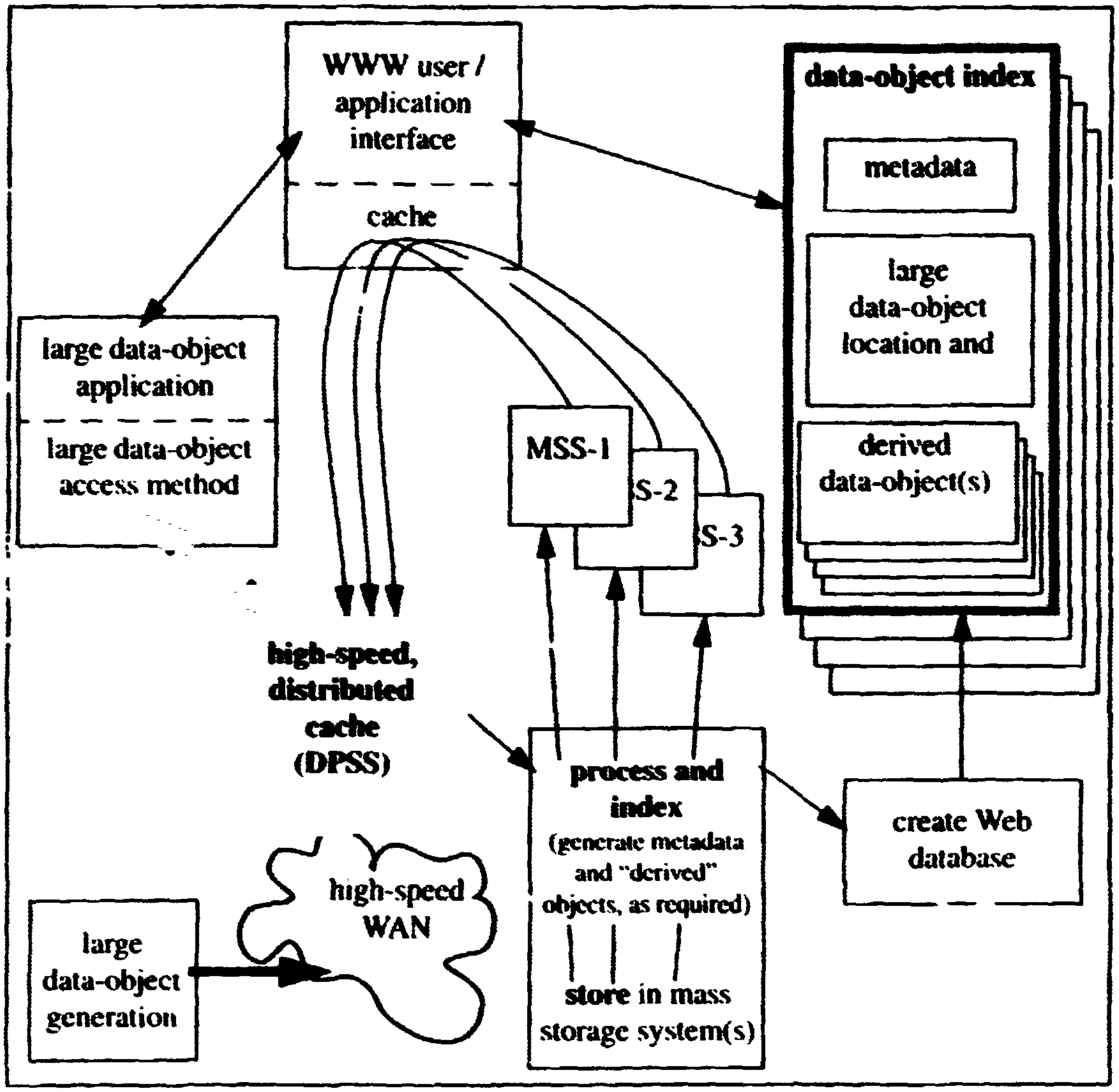

WWW user / application interface
cache
data-object index
metadata
large data-object location and
derived data-object(s)
large data-object application
large data-object access method
MSS-1
S-2
S-3
high-speed, distributed cache (DPSS)
process and index (generate metadata and "derived" objects, as required)
store in mass storage system(s)
create Web database
high-speed WAN
large data-object generation

Figure 1 Overall Architecture for a Distributed, Large Data-Object Environment

System Architecture) the access library translates the application view of the object to DPSS logical block addresses.

Together, these elements of a large data-object management architecture have provided effective management for several classes of data-objects. (See [2] and [3].)

2.1 Data Collection

Instrumentation systems are at the front-end of many distributed iarge data-object environments. Examples incluc̈e particle accele ıtor detectu.., Earth environment monitoring satellites, and medical imaging systems. These sources generate essentially continuous data streams, but ones that have "natural" boundaries that define "objects". For many of the instrumentation systems that we are interested in, one of the primary issues is getting the data out of the instruments and onto the network. One of the circumstances that has led to both the interest in, and practicality of, the architecture being described here is that general purpose workstations now have memory and I/O bus structures that are fast enough to acquire, structure, and send to the network, significant bandwidth data streams. (For example, the newest (mid-1996) DEC Alpha and Sun UltraSparc workstations can deliver 20 MBytes/sec of user data to a network interface.) This is an important capability because it means that the only special hardware that is required to bring instruments on-line is the interface between the workstation and the data source, and even this interface may be provided in "software" using off-the-shelf DSP-based I/O boards.

The frontend workstation acquires the raw data, formats it as "objects" by adding or using metadata from the experiment environment, and then sends the objects into the distributed environment. The data collection workstation frequently also serves as a buffer so that brief interpretations or slow-downs in the network do not result in loss of data

2.2 Network Cache

A high performance widely distributed network storage system is an essential component of a network-based large data-object environment. Distributing the components of a storage system throughout the network increases its capacity, reliability, performance, and security. Usable capacity increases in conjunction with a widely deployed, generalized security infrastructure that can support dynamic construction of systems through brokering and automated acquisition of resources. (See [4].) Reliability increases because storage systems that can be configured from components that have as little as possible in common (e.g., location) provide the resilience that comes from independence (transparent redundancy of data is also possible). Performance is increased by the combined characteristics of parallel operation of many sub-components, and the independent data paths provided by a large network infrastructure. Security is also potentially increased by having many independent components, each of which has local and independent enforcement mechanisms that can limit the scope of a security breach.

The Distributed-Parallel Storage System ("DPSS", also known as the "ISS") is an experimental system in which we are developing, implementing, and testing these ideas. In most configurations, the DPSS is used as a network-striped disk array designed to supply and consume high-speed data streams to and from other processes in the network. (See [5] and [6].)

The DPSS is essentially a "logical block" server whose functional components are distributed across a wide-area network. (See Figure 2) illustrating the DPSS architecture.) The DPSS uses parallel operation of distributed servers to supply image streams fast enough to enable various multi-user, "real-time", virtual reality-like applications in an Internet / ATM environment. There is no inherent organization to the blocks, and in particular, they would never be organized sequentially on a server. The data organization is determined by the application as a function of data type and access patterns, and is implemented so that a large collection of disks and servers can operate in parallel, enabling the DPSS to perform as a high-speed data source or data sink.

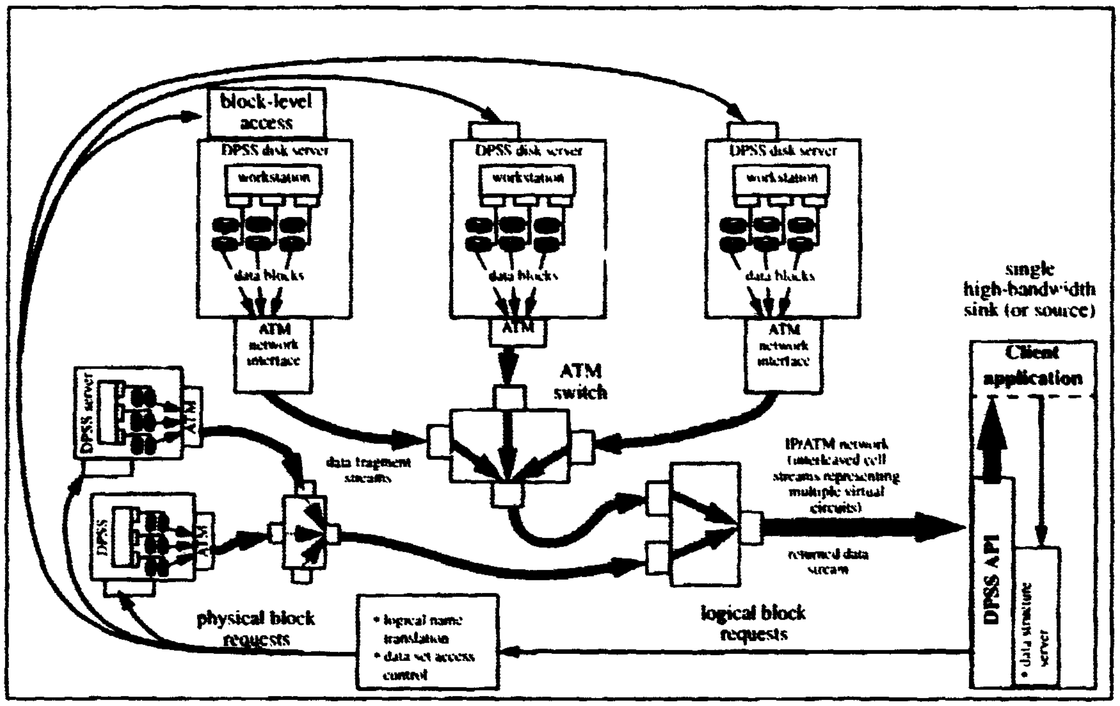


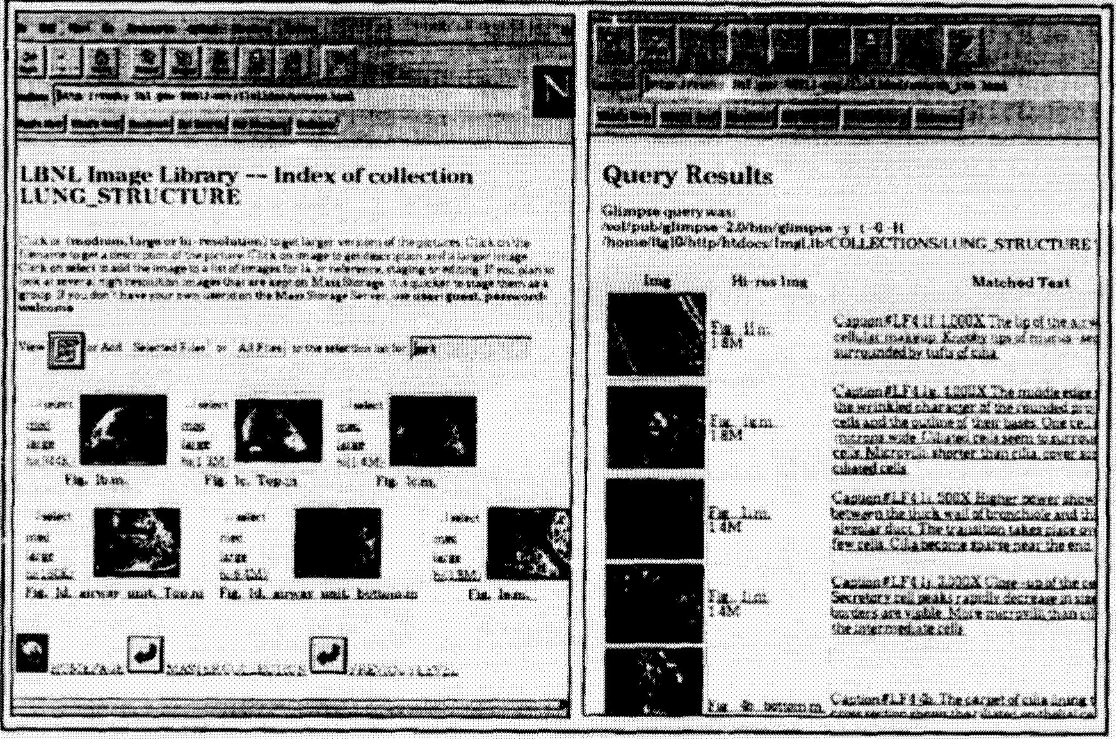
Figure 2 Distributed-Parallel Storage System Implementation

At the application level, the DPSS is a semi-persistent cache of named data-objects, and at the storage level it is a logical block server. Although not strictly part of the DPSS architecture, the system is usually used with an application agent library called a "data set structure access me.hod". This component provi.les an object-like encapsulation of the data, in order to represent complex user-levc' data structures so that the application does not have to retain this information for each ditt. :r' data set. The function and interface of the access methods are left to the application domain, but one simple example is for video data. In this case the access method allows applications to request data by "frame" number. The access method converts the application requests into logical block requests. These logical block requests are then sent to the DPSS Master which serves two functions, request and resource management. The Resource Manager maintains data set definitions, and the Request Manager is responsible for mapping the logical block requests to physical block requests. The Resource Manager also deals with ınteractions with the storage servers to determine available storage (a storage server is an independent entity and may deal with several DPSS Masters) and to establish the "security context" that provides the scope of control for various resources.

A security model and supporting security architecture provides for enforcing "owner" defined management policy for the physical resources and access policy for the data.

2.3 Data Management, Mass Storage, and the User Interface

In any scenario where data is generated in large volumes and with high throughput, and especially in a distributed environment where the people generating the data are geographically separated from the people cataloguing and using the data, there are several important issues: automatic generation of at least minimal metadata; cataloguing of the data and the metadata as the data is received (or as close to real time as possible); transparent management of multiple tertiary storage systems; and facilitation of co-operative research by allowing specified users at local and remote sites immediate access to the data, and incorporation of the data into other databases or documents.



LBNL Image Library -- Index of collection LUNG_STRUCTURE

Query Results

Glimpse query was:
/usr/pub/glimpse-2.0/bin/glimpse -y -i -0 -H
/home/itg10/http/htdocs/ImgLib/COLLECTIONS/LUNG_STRUCTURE

Img	Hi-res Img	Matched Text

Figure 3 **Large Data-Object Management for Video-like Objects**

One example of an approach to this capability is "ImgLib" (see [7]) which uses World Wide Web based tools to provide this library-like functionality. Semi-automatic cataloguing of incoming data is done by extracting associated metadata and converting it into text records, by generating auxiliary metadata and derived data, and by combining these into Web documents. Tertiary storage management is provided by using the remote program execution capability of Web servers to provide transparent access to different kinds of mass storage systems that then return the data-objects to the Web server, or, as is the case in several of our applications (and typically with large data-objects), move the data to a DPSS cache for access by applications. (For an example of a Java applet accessing the DPSS, see http://www-itg.lbl.gov/ISS/browser/iss2d.html.) Where the data-object is stored on tertiary storage, and how to access it, are all part of the Web-based "object-index".

Figure 3 illustrates some of these points. It shows a browser interface on the left, and on the right the results of automatically building a sub-collection of data-objects as the result of a search on the textual metadata. The information about the data-objects that result from the search is shown as a collection of thumbnails, associated pointers to other types of derived data, and a pointer to the original data-object. For the data-objects that reside on tertiary storage (a tape-robot based mass storage system in this case), there is an option for forcing migration of a data-object back to the on-line cache if the data of interest is not already there. The example in Figure 3 is fairly simple, but in the health care information system mentioned below, the original video data-object cannot be compressed and requires a special application to view the original data-objects (directly from the DPSS), or via JPEG or MPEG "movies" that are derived representations (see [2]).

2.4 A Health Care Information System Application

An example of a medical application that uses this distributed large data-object architecture is a system that provides for collection, storage, cataloguing, and playback of video-angiography images using a metropolitan area ATM network.

> *Cardio-angiography[2] is used to monitor and restore coronary blood flow, and though clinically effective, the required imaging systems and associated facilities are expensive. To minimize the cost of such procedures, health care providers are beginning to concentrate these services in a few high-volume tertiary care centers. Patients are typically referred to these centers by cardiologists operating at clinics or other hospitals; the centers then must communicate the results back to the local cardiologists as soon as possible after the procedure. The advantages of providing specialized services at distant tertiary centers are significantly reduced if the medical information obtained during the procedure is not delivered rapidly and accurately to the referring physician at the patient's home facility. The delivery systems currently used to transfer patient information between facilities include interoffice mail, U.S. Mail, fax machine, telephone, and courier. Often these systems are inadequate and potentially could introduce delays in patient care. (See [3].)*

Using a shared, metropolitan area ATM network and a high-speed distributed data handling system, video sequences and still images are collected from the video-angiography imaging systems, stored, and accessed by a remote user. The image data are sent through the network to storage and analysis systems, as well as directly to the users at clinic sites. Thus, data can be stored and catalogued for later use, data can be delivered live from the imaging device to remote clinics in real-time, or these data flows can all be done simultaneously. Whether the storage servers are local or distributed around the network is entirely a function of the optimal logistics. There are arguments in regional

[2] Cardio-angiography imaging involves a two plane, X-ray video imaging system that produces from several to tens of minutes of digital video sequences for each patient study for each patient session. The digital video is organized as tens of data-objects, each of which are of the order of 100 MBytes.

health care information systems for centralized storage facilities away frcm the hospital environment, even though the architecture is that of a distributed system. (See [8].)

This application is in operation in the CalREN, ATM network in the San Francisco Bay Area, and is described in some detail in [2].

3.0 Network Storage: The Distributed-Parallel Storage System

A central issue for the approach of using high-speed networks and distributed systems as the foundation of a large data-object management strategy is the performance of the system components, the transport / OS software, and the underlying network. Problems in any of these regimes will negatively affect our strategy, but such problems can usually be fixed if they can be isolated and characterized. A significant part of our work with high-speed distributed systems is developing a methodology and tools to locate and characterize bottlenecks.

We have designed and implemented the DPSS, as part of an DARPA-funded collaboration known as the MAGIC gigabit testbed[3] (see [9]), and as part of the U.S. Department of Energy's high-speed distributed computing program. This technology has been quite successful in several environments. The DPSS provides an economical, high-performance, widely distributed, and highly scalable architecture for caching large amounts of data that can potentially be used by many different users. Our current implementation provides foı real-time recording of, and access to large, image-like, read-mostly data sets. In the MAGIC testbed, the DPSS is distributed across several sites separated by more than 1000 Km of high speed network that uses IP over ATM as the network protocol, and is used to store very high resolution images of several geographic areas. The first client application of the DPSS was "TerraVision", a terrain visualization application that uses the DPSS to let a user explore / navigate a "real" landscape represented in 3D by using ortho-corrected, one meter per pixel images and digital elevation models (see [10]). *TerraVision* requests from the DPSS, in real time, the sub-images ("tiles") needed to provide a view of a landscape for an autonomously "moving" user. Typical use requires aggregated data streams as high as 100 to 200 Mbits/sec. Even in the current prototype system the DPSS is easily able to supply these data rates from several disk servers distributed across the network.

The combination of the distributed nature of the DPSS, together with the high data rates required by *TerraVision* and various load simulators, makes this a good system with which to test a high-speed network in a much more realistic manner than ttcp-like tools allow.

[3] MAGIC (Multidimensional Applications and Gigabit Internetwork Consortium) is a gigabit network testbed that was established in June 1992 by the U. S. Government's Advanced Research Projects Agency (ARPA). The testbed is a collaboration between LBNL, Minnesota Supercomputer Center, SRI, Univ. of Kansas, Lawrence, KS, USGS - EROS Data Center, CNRI, Sprint, U. S. West, Southwest Bell, and Splitrock Telecom. More information about MAGIC may be found on the WWW at: http://www magic.net/

3.1 DPSS Architecture

As mentioned, the DPSS is essentially a "logical block" server whose functional components are distributed across a wide-area network. The DPSS uses parallel operation of distributed servers to supply high-speed data streams. The data organization is determined by the application as a function of data type and access patterns, and is implemented during the data load process. The usual goal of the data organization is that data is declustered (dispersed in such a way that as many system elements as possible can operate simultaneously to satisfy a given request) across both disks and servers. This strategy allows a large collection of disks to seek in parallel, and all servers to send the resulting data to the application in parallel, enabling the DPSS to perform as a high-speed data server.

The implementation is based on the use of multiple low-cost, medium-speed disk servers which use the network to aggregate multiple server outputs for high performance applications. To achieve high performance all types of parallelism are exploited, including those available at the level of the disks, controllers, processors / memory banks, servers, and the network (see Figure 2).

The security model for the DPSS involves accommodating several different resource owners. The context established between the Data Set Manager (DSM) (see Figure 4) and the disk/storage servers reflects agreements between the owners of physical resources (disks) and an agent that is providing storage to a user community. This context enforces the disk usage agreements. The separate context established between the DSM and the users reflects the use-conditions imposed by the data "owner", and provides for ensuring access control that enforces those use-conditions. For more information on the security architecture see [4].

The overall data flow involves "third-party" transfers from the storage servers directly to the data-consuming application (a model used by most high performance storage systems). Thus, the application requests data, these requests are translated to physical block addresses (server name, disk number, and disk block), and the servers deliver data directly to the application.

3.2 Client Use of the DPSS

The client-side (application) use of the DPSS is provided through a library-based API that handles initialization (for example, an "open" of a data set requires discovering all of the disk servers with which the application will have to communicate) and the basic block request / receive interface. It is the responsibility of the client to maintain information about higher-level organization of the data blocks; to maintain sufficient local buffering so that "smooth playout" requirements may be met locally; and to run predictor algorithms that will pre-request blocks so that application response time requirements can be met. The prediction algorithm enables pipelining of the operation of the disk servers with the goal of overcoming the inherent latency of the disks. (See [5] and [6]).

None of this has to be explicitly visible to the user-level application, but some agent in the client environment must deal with these issues because the DPSS always operates on a best-effort basis: if it did not deliver a requested block in the expected time or order, it was because it was not possible to do so. In fact, a typical mode of operation is that pending block requests are flushed from the disk server read queues when the next set of requests arrive from the application. Fven if the DPSS cannot send all the requested data to the application, it is possible that the data was at least read from disk into the DPSS memory cache, where it will remain available for faster retrieval (for a short time). The application may then routinely re-request some fraction of the data. This deliberate "overloading" of the disk servers ensures that they will be kept busy looking for relevant blocks on disk and caching them in server memory. This approach ensures that the data pipeline stays full, and that disk server resources are never idle.

As mentioned, a DPSS client typically communicates with the DPSS through an application library called a "data structure access method library" (see Figure 4).)

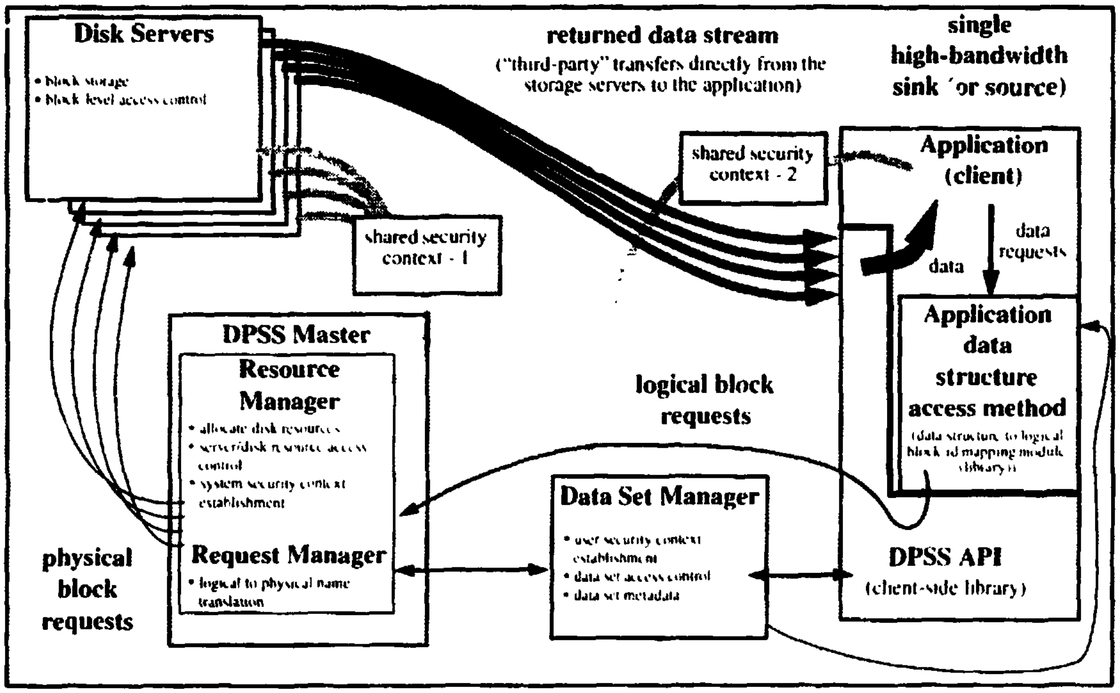

Disk Servers
• block storage
• block level access control
returned data stream
("third-party" transfers directly from the storage servers to the application)
single high-bandwidth sink (or source)
shared security context - 2
Application (client)
data
data requests
shared security context - 1
DPSS Master
Resource Manager
• allocate disk resources
• server/disk resource access control
• system security context establishment
Request Manager
• logical to physical name translation
logical block requests
Application data structure access method
(data structure to logical block id mapping module (library))
Data Set Manager
• user security context establishment
• data set access control
• data set metadata
DPSS API
(client-side library)
physical block requests

Figure 4 **Distributed-Parallel Storage System Architecture**

3.3 DPSS Implementation

In our prototype implementations, a typical DPSS consists of several (four - five) Unix workstations (e.g. Sun SPARCStation, DEC Alpha, SGI Indigo, etc.), each with several (four - six) fast-SCSI disks on multiple (two - three) SCSI host adapters. Each workstation is also equipped with an ATM network interface. A DPSS configuration such as this can deliver an aggregated data stream to an application of about 400 Mbits/s (50 Mbytes/s), using these relatively low-cost, "off the shelf" components, by exploiting the parallelism provided by approximately five disk servers, twenty disks, ten SCSI host adapters, and five network interfaces.

The software implementation is based on Unix interprocess communication mechanisms and a POSIX threads programming paradigm to manage resources on the disk servers (see [11] and [5]). The primary operating systems (Sun's Solaris, DEC's OSF, SGI's IRIX, and FreeBSD) all have slightly different implementations of threads, but they are close enough that maintaining a single source is not too difficult.

The implementation supports a number of transport strategies, including TCP/IP, RTP/IP [12] and UDP/IP. RTP and UDP do not guarantee reliable data delivery and never retransmit. Lost data are handled at the application level. This approach is appropriate when data has an age-determined value: data not received by a certain time is no longer useful, and therefore should not be retransmitted. This is the case for certain visualization scenarios. (This paper, however, focuses on TCP performance issues.)

Other papers describing the DPSS, including a paper that describes the implementation is detail [5], are available at http://www-itg.lbl.gov/DPSS/papers.html.

3.4 *TerraVision* and *tv_sim*: Prototype DPSS Client and Monitoring Tool

TerraVision uses the DPSS client library's logging facilities to log all data movement events associated with an application session. It uses the a standardized log format to monitor a data block's progress from the storage server disks, through the network, and into the application client.

We have also developed a simulator program, *tv_sim*, that can generate data requests and receive data blocks from the DPSS in a manner similar to *TerraVision*'s. Using this program we can generate synthetic request patterns, or repeatedly use actual *TerraVision* session data request traces, and attempt to verify and analyze performance bottlenecks in the DPSS, the application, or in the network in a controlled environment. *TerraVision* is a complex software suite running on complex hardware, and patterns of requested data are complex. *tv_sim* can emulate the *TerraVision* data request patterns through the trace-driven operation facility, but is a "null" application that can be run at much higher overall request rates than real applications, and can eliminate possible effects of data processing or graphics processing on the network throughput.

The *tv_sim* data request sending rate, in terms of block lists per second and blocks per list, can be set by the user, as can the saving of history logs in the DPSS standard format. The sender can also use trace / playback files of actual *TerraVision* sessions instead of generating its own lists of block requests, as mentioned above. Additionally, the user can specify the use of multiple data sets, overall running time, and other runtime characteristics. *tv_sim* and the DPSS thus can be configured to impose almost arbitrary load patterns on a network and to record the results.

The *TerraVision* data request trace is kept in terms of logical block request so that all aspects of the configuration of the DPSS may be changed - the number of storage servers, their location in the network, the data layout, etc. - to facilitate many types of experiments. In other words, real application data request patterns may be applied against different configurations of the distributed storage system, network, etc., because the logical block requests are independent of any aspect of the physical organization of the storage system.

4.0 Performance Monitoring Mechanisms

Network performance and distributed operation characteristics are obviously an important factor in the architecture that we are describing. There are virtually no behavioral aspects of an ATM "network" that can be taken for granted, even in an end-to-end ATM network. By "network" we mean the end-to-end data path from the transport API through the host network protocol (TCP/IP) software, the host network adaptors and their device drivers, the many different kinds of ATM switches and physical link bandwidths, and then up through the corresponding software stack on the receiver. Further, the behavior of different elements at similar places in the network architecture can be quite different because they are implemented in different ways. The combination of these aspects can lead to complex and unpredictable network behavior.

We have built performance and operation monitoring into the storage system and several applications, and have designed tools and methodologies to characterize the distributed operation of the system at many levels. As requests and data enter and leave all parts of the user-level system, synchronized timestamps are logged using a common logging format. At the same time, various operating system and network parameters may be logged in the same format. Several of these instrumented applications and tools are described below.

4.1 DPSS Timing Facility

A request for a data block takes the following path through the DPSS (see Figure 5). A request (a list of blocks) goes from the application to the Request Manager, where the logical block names are translated to physical addresses (server: disk: disk offset), then the individual requests are forwarded to the appropriate disk servers. At the disk servers, the data is read from disk into local cache, and then sent to the application (which has connections to all the relevant servers). Precise timestamps are gathered before and after each major function, such as name translation, disk read, and network send. All timestamps are then logged by the DPSS servers. The timestamps are also sent with the data block to the requesting application, where logging can be performed using the DPSS client library.

Timestamp consistency is provided by the GPS clock-based network time protocol (NTP – described below), which allows us to make precise throughput and latency measurements throughout the DPSS system and underlying network. Instead of trying to analyze the aggregate delay between sending a request and receiving the associated data block, we can pinpoint delays to within narrowly-specified steps in the data path.

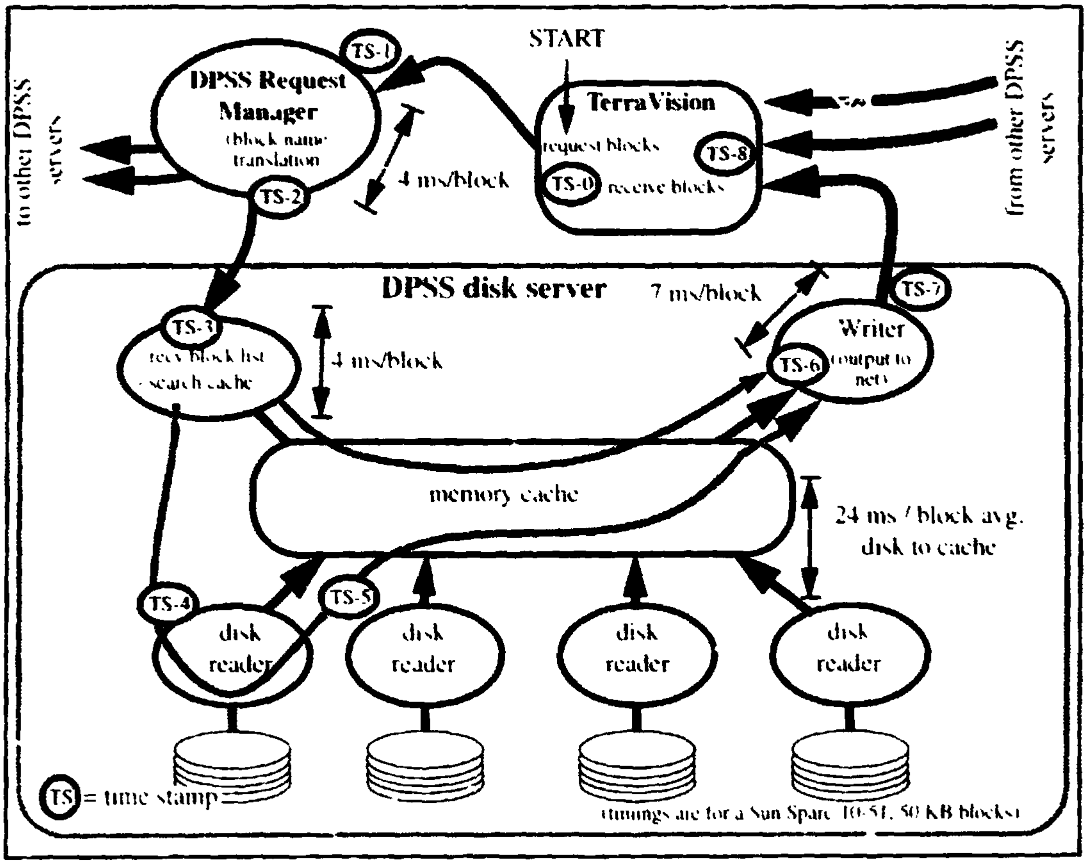


Figure 5 DPSS Performance Characterization Points and Optimal Average Timings

4.2 OS and Network Layer Monitoring

To complement the monitoring at the application level and in the DPSS, we also monitor various operating system and network conditions. We currently collect and log the following types of information:

- TCP retransmits
- CPU usage (user and system)
- CPU interrupts
- AAL 5 information
- ATM switch buffer overflows
- ATM hosts adapter buffer overflow

We have modified "netstat" and "vmstat"[4] to do logging. *netstat* provides the contents of various network-related data structures, while *vmstat* reports statistics on, among other things, virtual memory, disk, and CPU activity. Both programs were modified to present only a relevant subset of their information in the common logging format, and *netstat* was modified to poll and report continuously (it normally provides only a snapshot of current activity). We typically poll at 100 ms intervals, and since the kernel events are not timestamped, the data obtained this way represents all events in this interval.

4.3 Common logging format

To easily process the several gigabytes of log files which can be generated from this type of logging, all events are logged using a common format:

keyword: hostname; seconds; nano-sec; data; data; data;......;

The logging format is a semi-colon separated list of fields. The "keyword" is a unique identifier describing what is being logged. By convention, the first part of the keyword is a reference to the program that is doing the logging (e.g.: DPSS_SERV_IN, VMSTAT_SYS_CALLS, NETSTAT_RETRANSSEGS, TV_REQ_TILE). Each log record contains both the hostname of the system on which the event occurred and a timestamp. The timestamp is modeled after the format returned from the Unix "gettimeofday" call, and is logged with a numerical precision of one nanosecond. (We expect to be able to get the NTP synchronized accuracy of the timestamps down to better than one microsecond through a combination of the recently increased available precision of GPS signals and the use of real-time clock boards in the systems under study.)

The end of every log record can contain any number of "data" elements. These can be used to store any information about the logged event that may later prove useful. For example, for the NETSTAT_RETRANSSEGS event, there is one data element, and it contains the number of TCP retransmits since the previous poll time, and the DPSS_START_WRITE event data elements contain the logical block name, the data set ID, a "user session" ID, and an internal DPSS block counter. The log records for a given data block are associated by virtue of being collected and carried in the data block request message as it works its way through the system.

4.4 Log File Analysis Tools

Tools to analyze log files include *perl* scripts[5] to extract information from log files and write data files in a format suitable for using *gnuplot*[6] to graph the results. These tools were used to generate the graphs in Section 5.0.

When trying to identify the source of specific problems (such as those that showed up in the early WAN experiments described below) a good deal of exploratory, interactive analysis of the log data was the key to identifying the important factors, and graphical analysis of individual, exceptional events has proven to be the most important aspect of analysis when one is trying to identify the causes of specific behavior. There are several character-

[4] Both *netstat* (displays network statistics) and *vmstat* (displays virtual memory statistics) are tools available on many Unix systems

[5] For more information see. http://www metronet.com/perlinfo/perl5.html

[6] For more information see: http://www.cs.dartmouth.edu/gnuplot_info.html

istics that have made graphical analysis a powerful technique. What turned out to be the most important was the ability to treat "lifelines" (the temporal trace of a single block from application request all the way through the system to receipt of the data) as identifiable entities that could be individually manipulated and quantitatively analyzed.

In order to enable the quantitative analysis of individual events the graphical tools need to have several characteristics. Probably most important is that the significant features (e.g., all of the time points in a lifeline) must be grouped into graphical objects that can be manipulated as units. Further, it turned out that being able to "sketch", annotate, and create special measurement tools were all important capabilities, and so a versatile graphics drawing tool is very important. (This is illustrated in Figure 7 and Figure 10.)

The *gnuplot* graphics device driver for FrameMaker MIF[7] files groups graphics primitives at two levels: the graphics primitives that result from plotting data from one file are one "object", and at the next level down, each associated set of line segments are sub-objects. Therefore, each of the log file elements, such as block histories, flushed block histories, TCP retransmits, etc., are organized as objects, and the individual block life-lines are kept as sub-objects within these larger objects. The FrameMaker graphics tool can manipulate these objects and sub-objects independently, as well as providing the annotation, measurement, etc., mentioned above, and this proved invaluable in isolating, measuring, and marking significant events.

4.5 Use of NTP

To be able to perform meaningful analysis of a network-based system, precise timestamps, based on the synchronized clocks of all systems is essential. All MAGIC testbed hosts run the 'xntpd' program [13], which synchronizes the clocks of each host both to time servers[8] and to each other. (End-to-end transit times, including speed-of-light and switch delays, are of the order of 10 ms.) The MAGIC backbone segments are used to distribute NTP data, allowing us to synchronize the clocks of all hosts to within about 250 microseconds of each other. The location of the NTP servers in the MAGIC network are shown in Figure 9 (below).

This synchronization between host clocks allows us to characterize the operation of the system in useful and surprising ways. (See Figure 7, below.) For example, the DPSS name server, DPSS disk server, and application are typically on different physical hosts scattered over the network. For the events that characterize the operation of the system, 1 millisec-

[7] FrameMaker (http://www.adobe.com/prodindex/framemaker/main.html) is a multi-platform desk-top publishing program. MIF is its interchange file format that represents both text and graphics.

[8] There is considerable craft and lore in interfacing a precision time source to an NTP server platform, and we readily acknowledge Craig Leres of the LBNL Network Research Group (http://ee.lbl.gov) for working with Dave Mills and his students at Univ. of Delaware to "fine tune" every aspect of the particular GPS clock and server platform OS that we use in our experiments and in the MAGIC testbed. Also see [14] for a description of the characteristics of NTP in the MAGIC environment using this GPS receiver and server combination.

ond resolution is enough to establish the relationship between the impact of an event at one point in the network, and the origin of the event somewhere else in the network. Therefore 250 microseconds clock synchronization of all systems is required.

5.0 Example Analysis

This section presents some of the types of analysis that we have been able to do using the methodology and tools described in the previous section. The specific examples represent a snapshot of the state of our performance measurements during early 1996. As will be illustrated, there are several aspects of the overall system that dramatically affect performance. Two of these aspects that are changing rapidly are workstation ATM interfaces and ATM switch buffer management, and the numbers quoted here are primarily intended to be illustrative rather than an analysis of specific products. For example, over the past two years the throughput of a Fore Systems SBA-200 interface card operating in a Sun SS-20 has gone from 55Mbits/second to 105 Mbits/seconds, due to upgrades in both OS and device driver software, and in the same time frame the Fore ASX-200 ATM switch buffers have increased in size by 50 times. It is therefore certain that specific numbers like these will have changed by the time this paper is published.

The following sections describe performance results and analysis based on our monitoring and logging methodology as applied to the DPSS, *TerraVision*, and *tv_sim* programs (described above) operating together in ATM LANs and the MAGIC WAN.

5.1 End-to-End Performance Experiments

Experiments have been performed to examine the detailed interaction between a DPSS, whose disk servers are distributed over both ATM LANs and a wide-area ATM network, and the *TerraVision* application. Our initial monitoring experiments have focused on issues important to high-performance, highly distributed applications such as the *TerraVision* / DPSS combination. Using the log files described above, we are able to generate graphs (shown in the figures in this section) that have proven to be extremely useful in giving a detailed view of the throughput and latencies at each point in the distributed system: that is, in the application, the DPSS, and in the network.

5.2 LAN Experiments

Figure 6 represents a set of traces, collected by monitoring during application-driven operation, that illustrates the general operational characteristics of the DPSS, and specifically shows the strategy used by the *TerraVision* application in order to keep the overall "pipeline" of the storage system full.

Generally, each line style in the graphs indicates data from a different DPSS disk server, and different line styles are also used for "flushed" data requests (described below). The graphs plot "real time" on the horizontal axis, and the monitoring points on the vertical axis. The timestamps are collected at the monitor points, which represent critical points in the data request-response process from application to distributed storage system and back.

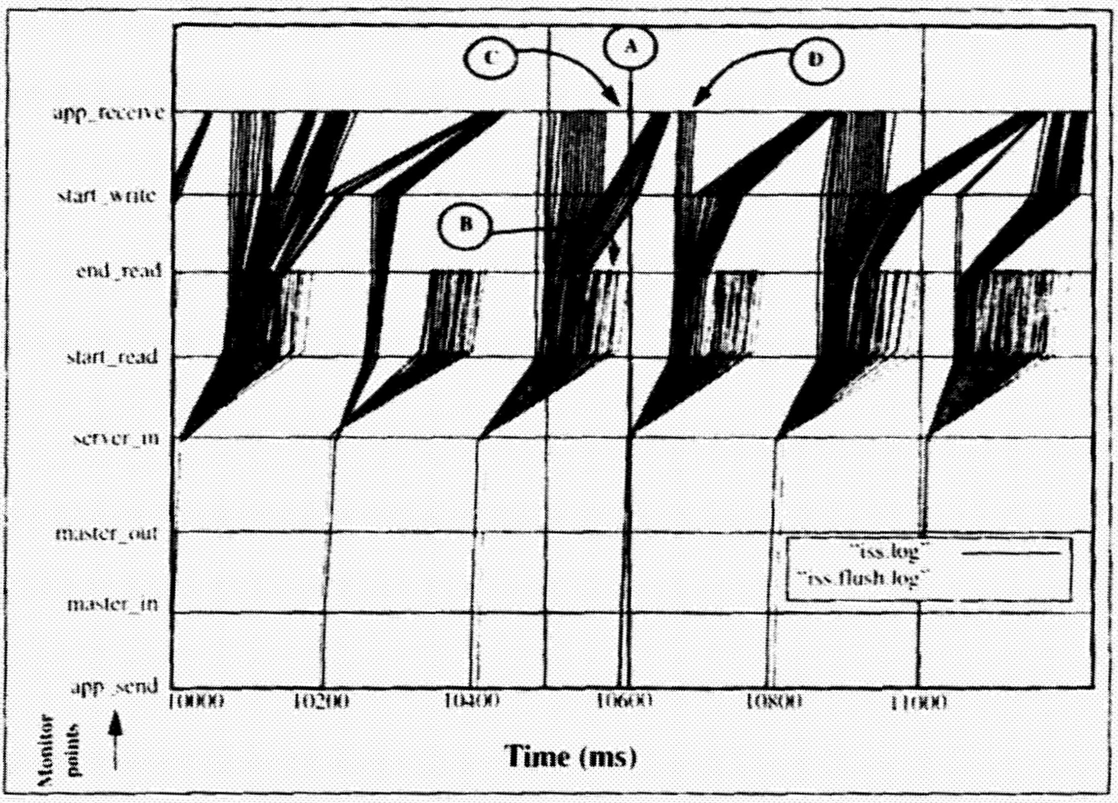


Figure 6 **One Server LAN Test**
(ATM LAN, one SS-20 as server, *tv_sim* on DEC 3000/600)

(See Figure 5.) Each line – a "life-line" – represents the history of a data block as it moves through the end-to-end path.

Referring to Figure 6, *TerraVision* sends a list of data block requests every 200 ms, as shown by the nearly vertical lines starting at the *app_send* monitor points. The initial single life-lines fan out at the *server_in* monitor point as the request lists are resolved into requests for individual data blocks. Each block request is first represented individually in the read queue (*start_read*).

Notice that many life-lines terminate at *end_read*, and that a few also end at *start_read*. Any individual data request that is not satisfied by the disk server before the next request list arrives is flushed (discarded) from all the server queues, but the data is retained in the server memory cache. For example, in Figure 6 the life-lines that started at 10,400 ms that were terminated at Ⓑ did so because the TCP write delay (of unknown cause) at Ⓒ "trapped" a previous set of blocks in the TCP write buffer. Block requests that were in the DPSS write queue when the next request list arrived (at 10,600 ms) are flushed from the queue. However, some of these blocks were re-requested in the 10,600 ms list, and these re-requests are satisfied very quickly because the data is in the disk server memory cache. This is seen in the nearly vertical life-lines at Ⓓ. (The "flush on next request" behavior is necessary to avoid deadlocks in the server, and provides a pre-fetch mechanism for the applications.)

Referring to Figure 7, using these life-line graphs it is also possible to get fairly detailed information on individual operations within the disk servers. For example, when two life-lines cros: in the area between *start_read* and *end_read*, this indicates that a read from one disk was faster that a read from another disk. (This phenomenon is clearly illustrated for the server represented by the crossing solid lines in Figure 7 at Ⓐ.) This faster read might be from disks with faster seek and read times (which is not the case in the experiment represented in Figure 7, as all participating systems used identical disks) or it might be due to two requested blocks being adjacent on disk so that no seek is required for the second block.

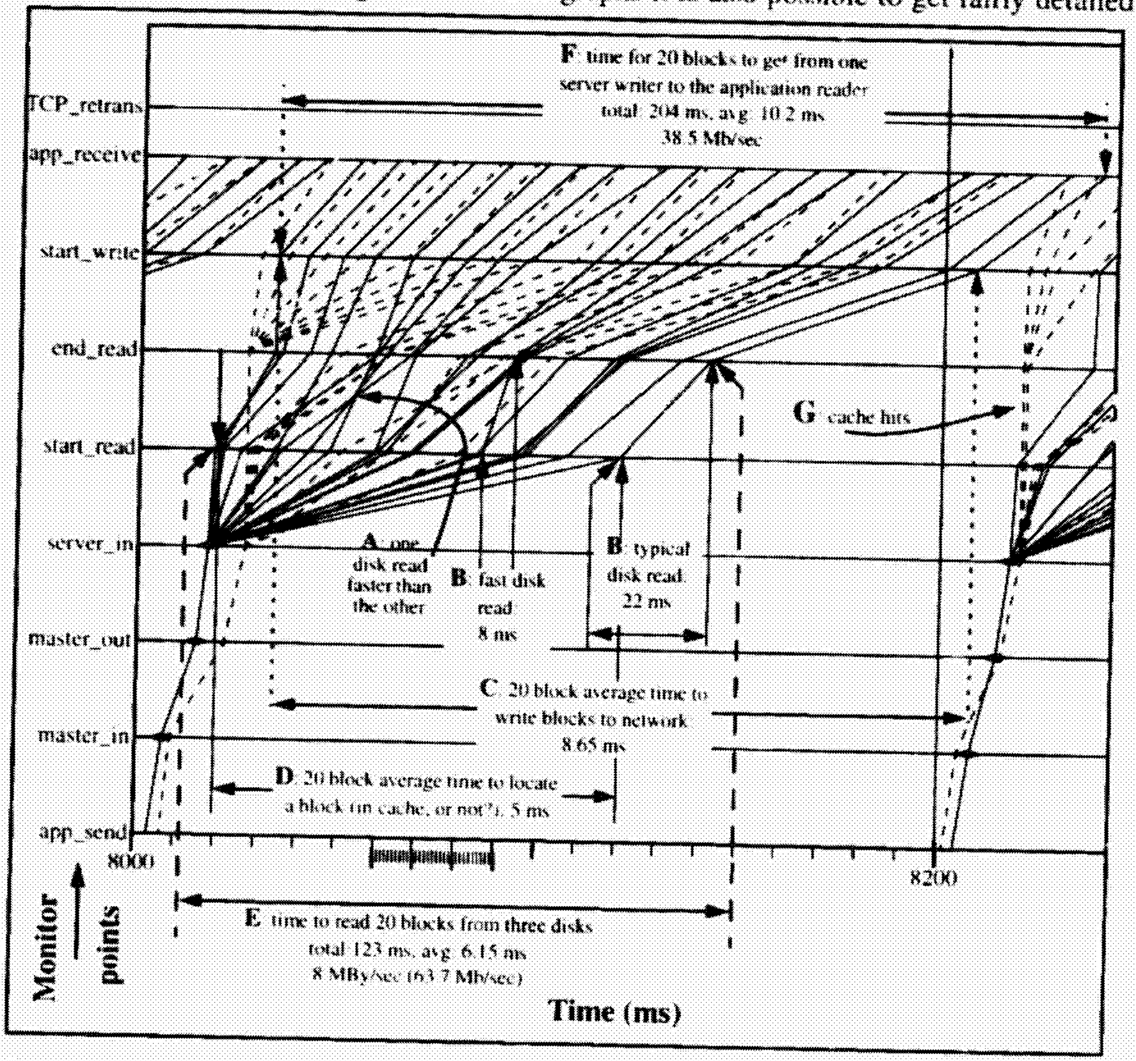


Figure 7 Detail From a Two Server, LAN, Experiment

Further, in Figure 7 we can also see:

- at "B" two different characteristic disk reads (one with an 8 ms read time and one with a 22 ms read time);
- at "C" the average time to cache a block and enter it into the network write queue is about 8.6 ms;
- at "D" the time to parse the incoming request list and see if the block is in the memory cache is about 5 ms;
- at "E" the overall server data read rate (four disks operating in parallel) is about 8 MB/sec;
- at "F" the actual throughput for this server while dealing with a set of real data requests is about 39 Mb/s (this throughput is receiver limited);
- at "G", there are two cache hits (blocks found in memory) as a result from previously requested, but not sent, data being requested. (Flushed requests are not shown in this figure.)

Figure 8 illustrates "correct" operation of multiple servers. This LAN-based two-server experiment shows the interaction of life-lines for blocks from different servers, and a case where the independent servers are behaving almost "perfectly": There are very regular block delivery patterns that alternate almost one-for-one between servers.

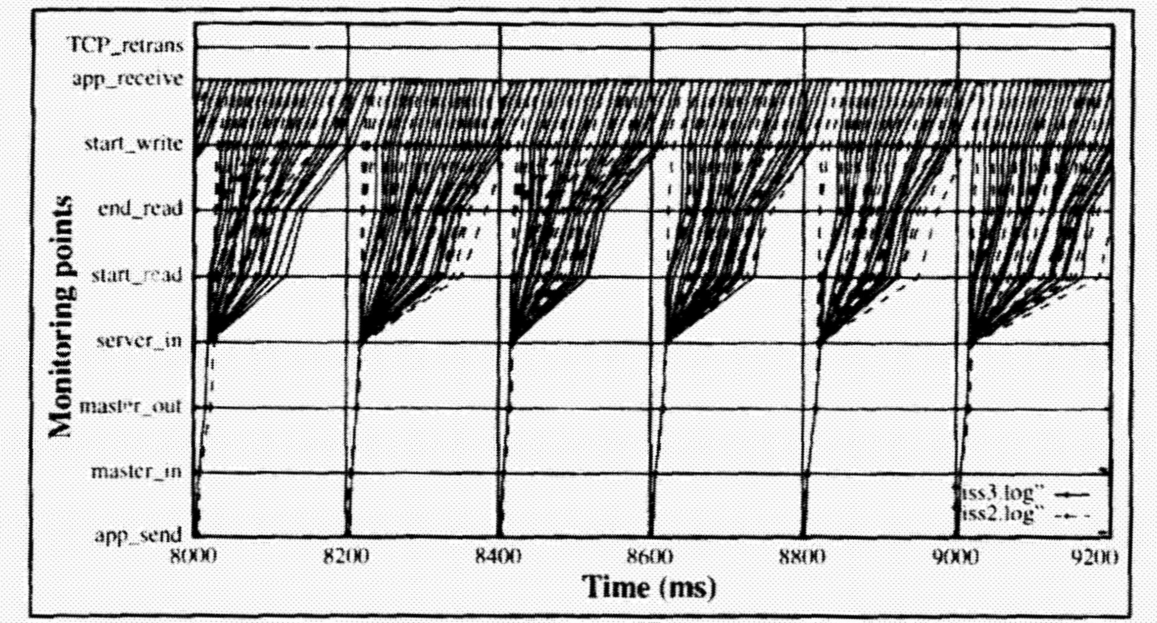


Figure 8 Two Server Test (ATM LAN, two SS-20s as servers, *tv_sim* on DEC 3000/600)

5.3 Wide Area Network Experiments

Of particular interest is the experiment for the three-server configuration operating in the MAGIC WAN testbed (Figure 9). The graphs for the LAN experiments (Section 5.2) show

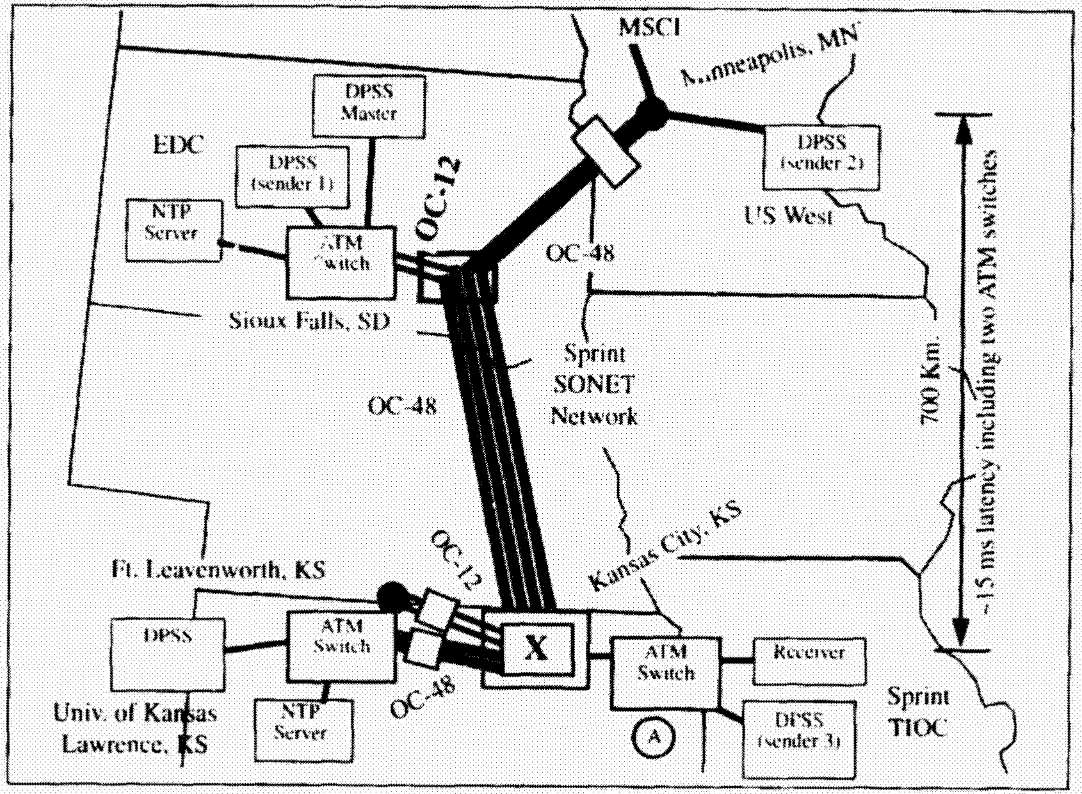


Figure 9 **The MAGIC Network: Test Configuration**

mostly expected behavior: smooth operation, no unexpected latencies, no TCP retransmissions, and so on. However, in the WAN case illustrated in Figure 10, one sees many TCP retransmissions and some extraordinarily long delays (up to 5500 ms).

5.3.1 WAN Experiment Environment

End-to-end performance experiments in the wide area use data block request traces from the *TerraVision* application and then use *tv_sim* to replay the traces. The traces for the application running in the MAGIC WAN environment were obtained with *TerraVision* running on an SGI Onyx with eight 150 MHz MIPS R4400 processors, 256 MB of main memory (4-way interleaved), two RealityEngineII graphics processors, and a single Fore Systems 100 Mb/s TAXI ATM interface. (This configuration is the minimum required to get good interactive visualization of 3D landscape.)

Experiments were run on the MAGIC ATM testbed, using the configuration illustrated in Figure 9. A five-minute *TerraVision* session trace of data block requests was captured, and then using this list of block requests, *tv_sim* was used to repeatedly request and receive those blocks. Experiments were run using a DPSS with one, two, and three disk server configurations. (The number of disk servers is independent of the application data request strategy and transparent to the application, except for establishing the data transfer connections). Log files were collected in the various distributed components for satisfied and unsatisfied block requests, TCP retransmission information, CPU usage, and ATM cell loss in the host adapters and ATM switches (though in this case the switches did not accurately report cell loss).

5.3.2 Analysis of a WAN Problem

In the early operating environment of MAGIC, it was very difficult to get anywhere near the expected throughput with multiple DPSS servers driving a single application. This resulted in a series of cell-pacing experiments done by our collaborators at U. Kansas, Lawrence (see [15] and [16]) that eventually determined that if every source (e.g. DPSS disk server) was paced at 1/N of the final link bandwidth that the total throughput increased significantly. While this solved an immediate problem, it was not a general solution, so we went back and conducted a series of experiments attempting to pinpoint the specific cause of the problem. These experiments and their results are described in [17]; however, here we illustrate some of the analysis.

Referring to Figure 10, first, let us analyze what the performance monitoring shows directly. If we look at the long-delayed block life-lines (emphasized in the figure) we see the characteristic behavior of a data block getting into the write queue (*start_write* monitor point) and then incurring some very long delays getting to the application. These long delays are almost always accompanied by one or more TCP retransmit events. The reason that the server is blocked as a whole (actually just one application is blocked since each application has its own TCP connection to the disk server) is that once a block is written to the TCP socket, the user level flushes have no effect, and TCP will re-send the block until transmission is successful, even though the data is likely no longer needed and is holding up newer data. The server unblocks when the a retransmission is successful, letting the next write proceed. The impact of this is substantial. Following received data lifelines back in time, the time that the data transfers stalled can be identified. These points are labeled (at the top of the graph) with the subscript "b" for blocked. (The three servers are labeled A, B, and C.) The transmission path (TCP circuit) has recovered when the next transmission proceeds at a "reasonable" rate, and the received data event just prior to the first of a group of "normal" receives is labeled with a subscript "u" for unblocked. At the bottom of the graph, the effective transmission from the servers for this application data path is indicated by the horizontal bars. The impact of this blocking and unblocking is that the effective throughput of all three servers combined (on a 100 Mbit/s data path that has no other traffic) is of the order of 1 Mbit/s. Unfortunately, at the time of this experiment we were not able to get accurate reporting from the switch "A" in Figure 9. However, what we surmise happened is the following.

The ATM switch A is where the three server streams come together, and this switch has a per port output buffer of only about 13K bytes. The network MTU (minimum transmission unit) is 9180 Bytes (as is typical for ATM networks). So, the situation is that three sets of 9 KBy IP packets are converging on a link with less than 50% that amount of buffering available, resulting in most of the packets (roughly 65%) being destroyed by cell loss at the switch output port. The TCP congestion window cannot get smaller than the MTU, and therefore TCP's throttle-back strategy is pretty well defeated: on average, every

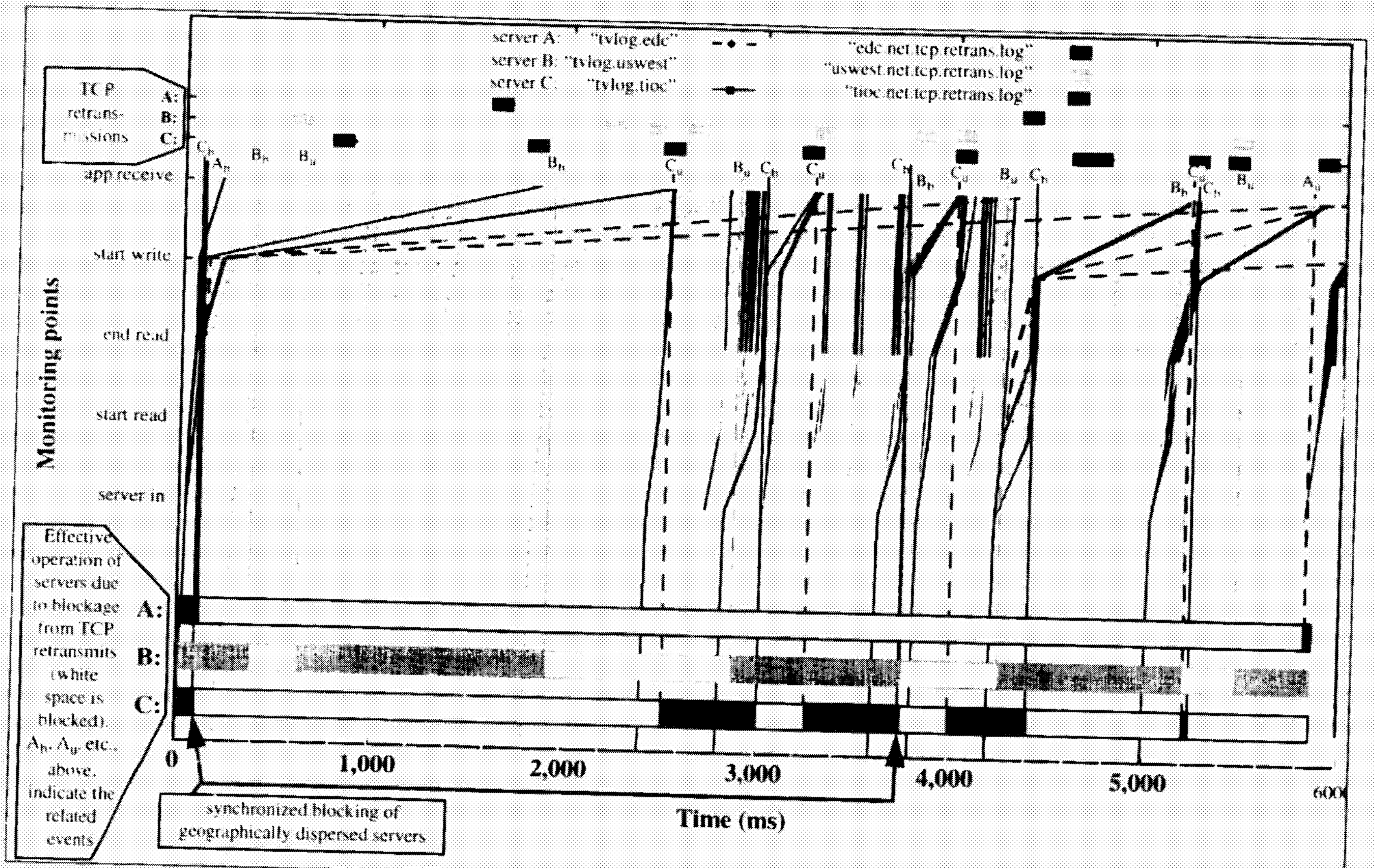


Figure 10 Three Server Test (MAGIC ATM WAN, three SS-10s as servers, tv_sim on SGI Onyx)

retransmit fails, even at TCP's "lowest throughput" setting, because this smallest unit of data is still too large for the network buffers.

Although we could not "prove" these assertions because we could not get accurate switch cell loss information, this analysis of Figure 10 provided enough information that the network operators upgraded the switch at A. The new switch as 600 KBy of buffering, which allows TCP's congestion algorithms to work correctly, and throughput is now up to an "average" of 30 Mbit/s per data path, as should be the case. For a more detailed analysis of this experiment, see [17].

6.0 Conclusions

In order to achieve high end-to-end performance in widely distributed applications, a great deal of analysis and tuning is needed. In the MAGIC testbed we are evolving a methodology that includes network-wide precision time sources and extensive instrumentation for time, latency, and throughput at all levels of the network, operating system, and applications. We monitor a large collection of parameters simultaneously (from the ATM level all the way up through disk performance on the storage servers and the application's use of the delivered data) in order to identify and correct performance bottlenecks. This top-to-bottom, end-to-end approach is proving to be a very useful mechanism for analyzing the performance of distributed applications in high-speed wide-area networks, and the type of graphs presented here are very useful and informative.

Apart from the immediate need for performance in MAGIC, the larger question that we hope to address by this methodology is whether high-performance use of networks, computing platforms, middleware, and applications has to be treated as a "system" problem (that is, all components considered and optimized together) or whether, as we find and correct problems, we will end up with an environment in which widely distributed, high-performance applications can be build by composing "stock" components, both hardware and software.

Some advice for those building distributed applications: timestamp all critical operations using a uniform log format, and run NTP on all hosts, so that the sort of analysis described here is possible.

7.0 Future Work

We are refining the tools and the measurement techniques that capture and log events, and several of the other MAGIC consortium members are doing the same. (For example, a number of the "events" currently collected are the results of watching system variables for some interval, and then using the interval mid-point as the time stamp, when we should be getting the actual event timestamp.) We are exploring the use of the University of Kansas "Data Stream Driver" [18] to improve our timing accuracy for operating system events.

We hope to be able to use the log files from the DPSS client library as "playback" files for 'netspec'[19], which is a distributed network performance measurement tool that is being designed and developed at the Telecommunications and Informations Sciences

Laboratory, University of Kansas. *netspec* supports multiple connections per session, and it will support multiple protocols. This will allow us to easily recreate many different traffic scenarios. This work was presented at the 1996 DARPA Workshop on Wide Area ATM Performance (see http://www.tisl.ukans.edu/Workshops/ATM_Performance/), and one result of this workshop is there will be more work put into working with the University of Kansas to incorporate this logging and graphing methodology into *netspec* to create a general purpose set of tools.

This work is ongoing, and progress reports will be published at http://www-itg.lbl.gov/DPSS.

8.0 References

[1] Johnston, W. and D. Agarwal, "The Virtual Laboratory: Using Networks to Enable Widely Distributed Collaboratory Science" An NSF Workshop, Virtual Laboratory whitepaper. (See http://www-itg.lbl.gov/~johnston/Virtual.Labs.html)

[2] Johnston, W., Jin Guojun, Gary Hoo, Case Larsen, Jason Lee, Brian Tierney, Mary Thompson, "Distributed Environments for Large Data-Objects: The Use of Public ATM Networks for Health Care Imaging Information Systems", (http://www-itg.lbl.gov/~johnston/APII.1.1.fm.html)

[3] Kaiser - LBNL - Philips CalREN project. See http://www-itg.lbl.gov/Kaiser/LKP/homepage.html.

[4] Johnston, W. and C. Larsen, "Security Architectures for Large-Scale Remote Collaboratory Environments: A Use-Condition Centered Approach to Authenticated Global Capabilities" (draft at http://www-itg.lbl.gov/~johnston)

[5] Tierney, B., Johnston, W., Chen, L.T., Herzog, H., Hoo, G., Jin, G., Lee, J., "Using High Speed Networks to Enable Distributed Parallel Image Server Systems", Proceedings of Supercomputing '94, Nov. 1994, LBL-35437. Available from http://www-itg.lbl.gov/DPSS/papers.html.)

[6] Tierney, B., Johnston, W., Herzog, H., Hoo, G., Jin, G., and Lee, J., "System Issues in Implementing High Speed Distributed Parallel Storage Systems", Proceedings of the USENIX Symposium on High Speed Networking, Aug. 1994, LBL-35775. (http://www-itg.lbl.gov/DPSS/papers.html.)

[7] ImgLib: See http://www-itg.lbl.gov/ImgLib/ImgLib_intro.html

[8] Johnston, W. and A. Allen, "Regional Health Care Information Systems: Motivation, Architecture, and Implementation," LBL Technical Report 34770 (draft), Dec. 1993.

[9] Fuller, B., I. Richer "The MAGIC Project: From Vision to Reality," IEEE Network, May, 1996, Vol. 10, no. 3.

[10] Lau, S, and Y. Leclerc, "TerraVision, a Terrain Visualization System,", Technical Note 540, SRI International, Menlo Park, CA, Mar. 1994. (http://www.ai.sri.com/~magic/terravision.html)

[11] Stevens, R. W., *TCP/IP Illustrated*, Volume 1 The Protocols, Addison-Wesley Professional Computing Series, 1994.

[12] Schulzrinne, H., S. Casner, R. Frederick and V. Jacobson "RTP: A Transport Protocol for Real-Time Applications", An Internet Request for Comments (RFC), January 1996. Available from: ftp://ds.internic.net/rfc/rfc1889.txt

[13] Mills, D., "Simple Network Time Protocol (SNTP)", RFC 1769, University of Delaware, March 1995. (http://www.eecis.udel.edu/~ntp/)

[14] Johnston, W., Tierney, B., Herzog, H., Hoo, G., Jin, G., Lee, J. "Time and MAGIC", from the MAGIC Technical Symposium, Minneapolis, Minnesota, 1994. Available from http://www-itg.lbl.gov/DPSS/talks.html.

[15] Evans, Joseph B., Victor S. Frost, Gary J. Minden, "TCP and ATM in Wide Area Networks", CNRI Gigabit Network Workshop '94. (http://www.magic.net/tcp/overview.html)

[16] Ewy, B. J., J.B. Evans, G.J. Minden, and V. S. Frost, "TCP/ATM Experiences in the MAGIC Testbed", Fourth IEEE Symposium of High Performance Distributed Computing, August 1995, pp. 87-93.

[17] Tierney, B., W. Johnston, G. Hoo, J. Lee, "Performance Analysis in High-Speed Wide-Area ATM Networks: Top-to-Bottom End-to-End Monitoring", IEEE Network, May, 1996, Vol. 10, no. 3. LBL Report 38246, 1996. (Also see http://www-itg.lbl.gov/DPSS/papers.html.)

[18] Buchanan, B., Menon, R., Niehaus, D, "The Data Streams Kernel Interface", Telecommunications & Information Sciences Laboratory, University of Kansas, Feb. 1996.

[19] Jonkman, Roelof J.T., "An Overview of NetSpec", Telecommunications & Information Sciences Laboratory, University of Kansas. (http://www.tisl.ukans.edu/Projects/AAI/products/netspec/)

NEXT
DOCUMENT

Understanding Customer Dissatisfaction With Underutilized Distributed File Servers

Erik Riedel
Department of Electrical and Computer Engineering
Carnegie Mellon University
5000 Forbes Avenue
Pittsburgh PA 15213
riedel@cmu.edu
Tel 412-268-3056
Fax 412-268-3010

Garth Gibson
School of Computer Science
Carnegie Mellon University
5000 Forbes Avenue
Pittsburgh PA 15213
garth.gibson@cs.cmu.edu

Abstract

An important trend in the design of storage subsystems is a move toward direct network attachment. Network-attached storage offers the opportunity to off-load distributed file system functionality from dedicated file server machines and execute many requests directly at the storage devices. For this strategy to lead to better performance as perceived by users, the response time of distributed operations must improve. In this paper, we analyze measurements of an Andrew File System (AFS) server that we recently upgraded in an effort to improve client performance in our laboratory. While the original server's overall utilization was only about 3%, we show how burst loads were sufficiently intense to lead to periods of poor response time significant enough to trigger customer dissatisfaction. In particular, we show how, after adjusting for network load and traffic to non-project servers, 50% of the variation in client response time was explained by variation in server CPU utilization. That is, clients saw long response times in large part because the server was often over-utilized when it was used at all. Using these measures, we see that off-loading file server work in a network-attached storage architecture has the potential to benefit user response time. Computational power in such a system scales directly with storage capacity, so the slowdown during burst periods should be reduced.

This research is sponsored by DARPA/ITO through ARPA Order D306, and issued by Indian Head Division, Naval Surface Warfare Center, under contract N00174-96-0002. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing official policies, either expressed or implied, of any sponsoring or supporting agency, including the Defense Advanced Research Projects Agency and the United States Government.

1. Introduction

Recent trends in the computer industry have greatly increased the demands for common, shared information repositories In most cases, these have taken the form of distributed file systems that are shared across a workgroup, organization-wide, or even world-wide. A distributed file system, with a number of machines acting as "servers" and a much larger number of "clients" have become popular due to a number of factors, including separation of administrative concerns, sharing of data, and transparency [Spasojevic96]

Advances in other computing technologies have made possible many novel applications that are placing increasing demands on distributed storage systems. The delivery of video and audio, large-scale parallel applications, and the growth of the Internet have increased demands on distributed information systems both in terms of the resources required by individual applications and the aggregate demands made by a continually increasing number of clients

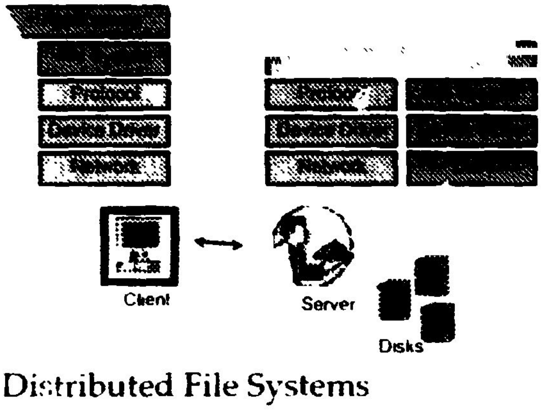
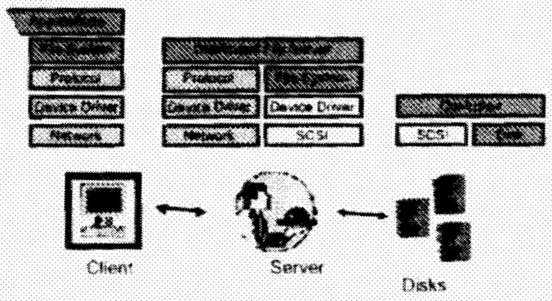


Figure 1 - Traditional Distributed File System

At the core of all distributed information systems lies a set of server resources that are becoming increasingly loaded as the demands increase A traditional distributed file system model, where "storage" is simply embodied in the disk and device driver, is illustrated in Figure 1 This picture explains in part why increasing load on distributed file systems often requires fast file servers - the file server must traverse two protocol stacks for each client request Data must move from attached disk drives, across the SCSI bus, through the server's memory system, back across the system bus, down the network protocol stack and, finally, onto the network wire The server has very little "interest" in the data, yet it must move it through its memory hierarchy - possibly several times - in order to satisfy all the protocol layers involved

In conjunction with this pressure toward using faster machines as file servers, recent years have seen rapid development, both in terms of areal density and in the raw bandwidth that can be provided off the platters of fixed storage devices On top of these trends, perhaps the largest change comes from standardizing storage interfaces The adoption of the SCSI interface for storage devices allowed storage vendors to optimize below a common protocol, and application and file system developers to optimize above it By specifying separate high-level "logical" interface and a physical interface, SCSI made possible numerous optimizations inside disk controllers including RAID, transparent recovery management, dynamic remapping, and storage migration. A common interface to operating system software allowed users to buy drives based on price and performance, rather than on compatibility requirements with other parts of their computer systems. This model has led to typical, high-performance distributed file systems that today look more like Figure 2. There is one interconnect for communication between clients and servers (IP or IPX over ATM or Ethernet), and another for communication between servers and disks (SCSI).



Distributed File Systems (2)

Figure 2 - Actual Distributed File System Architecture Today

The difficulty with this architecture is that a good portion of the overall system power is "dissipated" in the server system that bridges the gap between SCSI and the distributed file system protocol used by clients. With relatively slow storage devices and relatively slow networks, this additional overhead has until now been hidden among other limitations. The continued development of disk technology has made possible products with sustained data rates of up to 12 MB/s shipping today and 40 MB/s does not look unreasonable by the end of the decade. Fibre Channel interconnects also eliminate the traditional SCSI bus a a bottleneck. ATM, Fast Ethernet, and Myrinet provide client network rates of 12 MB/s today and 100 MB/s in the near future. These advances mean that the amount of room to "hide" inefficiencies in distributed file server implementations is shrinking dramatically.

The study described in the rest of this paper examines the requirements placed on file server architectures by studying the behavior of current distributed file system technology. Specifically, we have analyzed the system-level behavior of an AFS (Andrew File System) server in our environment. The following sections will present the behavior we have observed and the pressure on file server performance.

Section 2 provides a brief overview of AFS and presents our measurement methodology, tools, and environment. Section 3 provides a summary of some of the workload characteristics we observed. Section 4 discusses the factors that affect AFS performance as perceived by users. Section 5 discusses the potential available through the use of network-attached storage devices. Finally, we conclude in Section 6 and discuss avenues of future work.

2. Experimental Methodology

2.1. *Andrew File System*

At Carnegie Mellon (and at hundreds of other large institutions around the world) the Andrew File System is used by nearly all computer users. The major contribution of AFS over previous distributed file systems such as the Network File System (NFS), was the focus on scalability of server resources. The goal of AFS was to support a campus-wide network of workstations and users with a relatively small amount of file server resources [Howard88]. The primary way in which AFS addressed this goal is through the use of local disk for extensive client-side caching. Each client workstation in an AFS environment dedicates a portion of its local disk space as a cache for frequently accessed remote data. Data in client caches is kept up-to-date through the use of a strong consistency protocol based on callbacks. When a client accesses a particular file from an AFS server, the server marks a callback for that data and client and promises to inform the client when the data is changed. Rather than having a large number of clients constantly checking in at the file server to see if data has changed, the responsibility for cache invalidation lies with the server.[1]

In the Spring of 1996, our lab upgraded its AFS server in response to our users' complaints about AFS performance. A major motivation in writing this paper is to identify and detail the performance reasons behind the upgrade and determine the implications for AFS distributed file systems built on network-attached storage architectures.

2.2. *Measurement Environment*

The measurements reported here were taken from a single file server over the course of a two month period at the beginning of 1996. This server contained all of the project volumes used for research in the Parallel Data Laboratory (PDL). The server was a Sun SPARCstation 4/60 with 24 MB of memory serving 20 volumes representing a total of 8 GB of data in 4 partitions. The clients were fifteen Alpha AXP machines (Turbochannel models 300, 400, 500 and 600 and PCI models 200 and 400) nine IBM RS/6000 250s located in a single laboratory, and fifteen additional machine of varying types, ranging in power from DECstation 5000s to a SPARCstation 20, in this lab and in the offices of students and faculty. The workload, a diverse set of activities one would expect from a medium-sized research group, included software development, document preparation, data analysis and simulation.

The School of Computer Science network, to which all these machines are connected, consists of an Ethernet segment for each floor of its building, with an additional segment for the central machine room where all AFS servers are housed, all of which are connected to a single bridged backbone. The *cs.cmu.edu* AFS cell, in which our measurements where taken, consists of 25 (primarily SPARCstation) dedicated servers providing home directories, repositories for shared, locally-maintained software

[1] Another goal of AFS is to serve as a wide-area distributed file system that can span the entire globe. In order to facilitate this, AFS provides a single global namespace that is divided at the top level of the hierarchy into a number of *cells*, each of which represents a specific organization or administrative domain. The basic unit of distribution in AFS is a *volume*, a related set of files assigned for a specific purpose and representing a specific allocation of disk space. Each cell contains a set of well-known *database server* machines that maintain a mapping of which volumes reside on which of a number of *file server* machines. The file server machines have disks attached that are divided into logical *partitions*, each of which holds some number of volumes.

collections, and volumes assigned to specific research projects. Larger projects often "own" an entire server which houses all of that group's project volumes. The server under test was running AFS 3.2 with local patches under the Mach 2.6 operating system and the clients were running several different operating systems with AFS versions ranging from locally modified 3.1 to 3.4beta.

2.3. *Analysis Tools*

Traces of file server activity were taken with the aid of a tracing package developed by the Coda group at Carnegie Mellon [Mummert94]. A number of trace points, including most system calls, all accesses into the buffer cache, and all disk requests, within the operating system were annotated with log entries. Logs were collected in a kernel buffer and periodically extracted and shipped over the network to a second machine that gathered the traces on its local disk and periodically transfer them to tape. This facility allowed the collection of very detailed system traces without much effect on performance. The Coda group measured a performance impact of between five and seven percent in their studies. Traces were collected almost continuously over a two month period resulting in over 4 GB of data.

In addition to this data, client and server AFS activity was measured through the use of the AFS *xstat* facility which collected hourly summaries of operations performed, aggregate performance per operation type, as well as details on request sizes.[2] We also used *rxdebug* and *vos* to collect information on active clients and volume use patterns from the server. Statistics of the server and clients over three months represented an additional 400 MB of raw data.

To track performance of the network connecting our machines, we collected statistics derived from a periodic measurement of the round-trip time to the server and client network segments. Our measurement machine (ozone) executed a 30-second ping every 5 minutes noting the average round-trip time and packet loss rate to a selected number of clients (one on each floor with client machines) and to the server.

We developed a set of scripts to process the trace and summary data and used the Matlab numerical computation and visualization system to provide plots and statistical tests. In the following sections we will provide plots of measured data as well as means, variances, and Pearson r correlation coefficients, and r^2 coefficients of determination. We use the Pearson coefficient of determination to quantify how much of the variation in a set of measurements can be accounted for by the characteristics of underlying system factors [Kirk90].

3. Workload Characteristics

In this section, we summarize a number of basic parameters of the workload recorded in our traces. Specifically the effectiveness of client caches, the mix of AFS operations at both the clients and our server, and the transfer size distributions at the server.

[2] Due to the highly distributed nature of AFS and our desire to measure a real workload, it was not possible to track all of the clients that made requests to this particular server, nor can we determine exactly what client activity was directed to this particular server. This introduces some amount of "noise" into our data, making some variations more difficult to explain.

3.1. *Client Caching*

As shown in previous work, the hit ratio for data in the local AFS cache is extremely good [Spasojevic96, Howard88] Table 1 gives the average hourly hit ratio across the twenty clients for which we have the most complete data. This data emphasizes the well-established fact that there is a high degree of temporal locality in user access streams, and that local disk caching in AFS removes a considerable burden from the file server The data shown represents measurements from a single week of traces - specifically the week of January 29, 1996 to February 4, 1996 This representative week-long period will be used throughout the rest of the paper.

	Average	1	2	3	4	5	6	7	8	9	10
data	97.0	99.6	98.9	92.7	94.2	90.5	99.4	81.7	95.8	98.8	96.5
metadata	61.8	98.0	81.5	36.9	15.6	22.1	76.9	16.6	17.5	33.9	20.8
		11	12	13	14	15	16	17	18	19	20
data		99.1	99.4	98.7	99.9	99.6	99.3	99.5	99.3	98.4	99.1
metadata		62.3	22.9	45.0	99.9	98.2	99.3	99.1	96.9	99.0	95.2

Table 1 - Client Cache Hit Ratio

3.2. *Operation Distribution*

Table 2 shows a breakdown of the most frequently used AFS operations and their relative popularity. The Clients column shows the total for the 20 clients reported above over the course of the same week. Note that the number of client and server requests does not match up because this is not a closed system - there were additional clients making requests of the PDL server, and the PDL clients made use of other AFS servers (as we will discuss in more detail later) The total amount of data transferred by clients was 993 MB in FetchData requests and 520 MB in StoreData requests The server provided a total of 750 MB of data via FetchData and accepted 955 MB via StoreData requests.

AFS Operation	Clients		Server	
	total	fraction	total	fraction
FetchStatus	748,620	68.0%	412,695	43.4%
StoreStatus	20,085	1.8%	22,642	2.4%
FetchData	174,717	15.9%	62,288	6.5%
StoreData	46,630	4.2%	32,414	3.4%
CreateFile	15,407	1.4%	17,089	1.8%
RemoveFile	17,242	1.6%	20,422	2.1%
BulkStatus	0	0.0%	244,636	25.7%
GetTime	50,568	4.6%	122,393	12.9%
GiveUpCallbacks	28,343	2.6%	17,298	1.8%
total	1,101,612		951,877	

Table 2 - Distribution of AFS Operations

3.3. *Request Sizes*

Table 3 shows the distribution of request sizes over the course of a week. As seen in previous studies, small requests dominate the mix, while most of the bytes are moved in large requests [Spasojevic96, Baker91]. 80% of reads and 65% of writes are for less than 8 kilobytes However, for StoreData requests, more than two-thirds of the bytes are moved at the largest request size This means that system designers must consider optimizations that maximize the bandwidth of the largest requests without adversely affecting the latency of the majority of small operations.

Request Size	FetchData		StoreData	
up to 128 bytes	19,503	31.3%	7,607	23.5%
129 bytes to 1 K	3,663	5.9%	3,196	9.9%
1 K to 8 K	24,858	39.9%	10,035	31.0%
8 K to 16 K	2,127	3.4%	2,244	6.9%
16 K to 32 K	1,889	3.0%	2,510	7.8%
more than 32 K	10,245	16.4%	6,789	21.0%
total	62,285		32,381	

Table 3 - Distribution of Request Sizes

4. Impacts on User-Perceived Performance

4.1. *Server Utilization*

These statistics provide some idea of the typical work being performed by an AFS file server, but how does the performance of the server figure into customer purchasing and system sizing decisions? The Parallel Data Laboratory recently upgraded its AFS server from a dedicated SPARCstation 1 to a brand-new dedicated SPARCstation 20 with about 5 times the rated performance This upgrade was done to a large extent in response to the increasingly vocal complaints of slow performance by our users. In fact, little data was consulted in the decision to upgrade this server. In an attempt to understand what effect the resources available on our server has on user performance, we took a look at the load on the original server after the upgrade. Given the traces described above, we can in hindsight attempt to better understand how server load relates to file system performance and customer satisfaction

The top chart of Figure 3 shows the fraction of the server CPU spent in the AFS *fileserver* process over the course of a week, averaged over ten minute intervals. As we can see, the CPU on the server is mostly idle. Although we do see a number of peak periods in which the utilization reaches as high as 65%, the mean CPU utilization is less than 3% This is a disturbing result Were we wrong to spend about $10,000 for a new, fast file server to replace a slow, inexpensive server that is only 3% utilized?

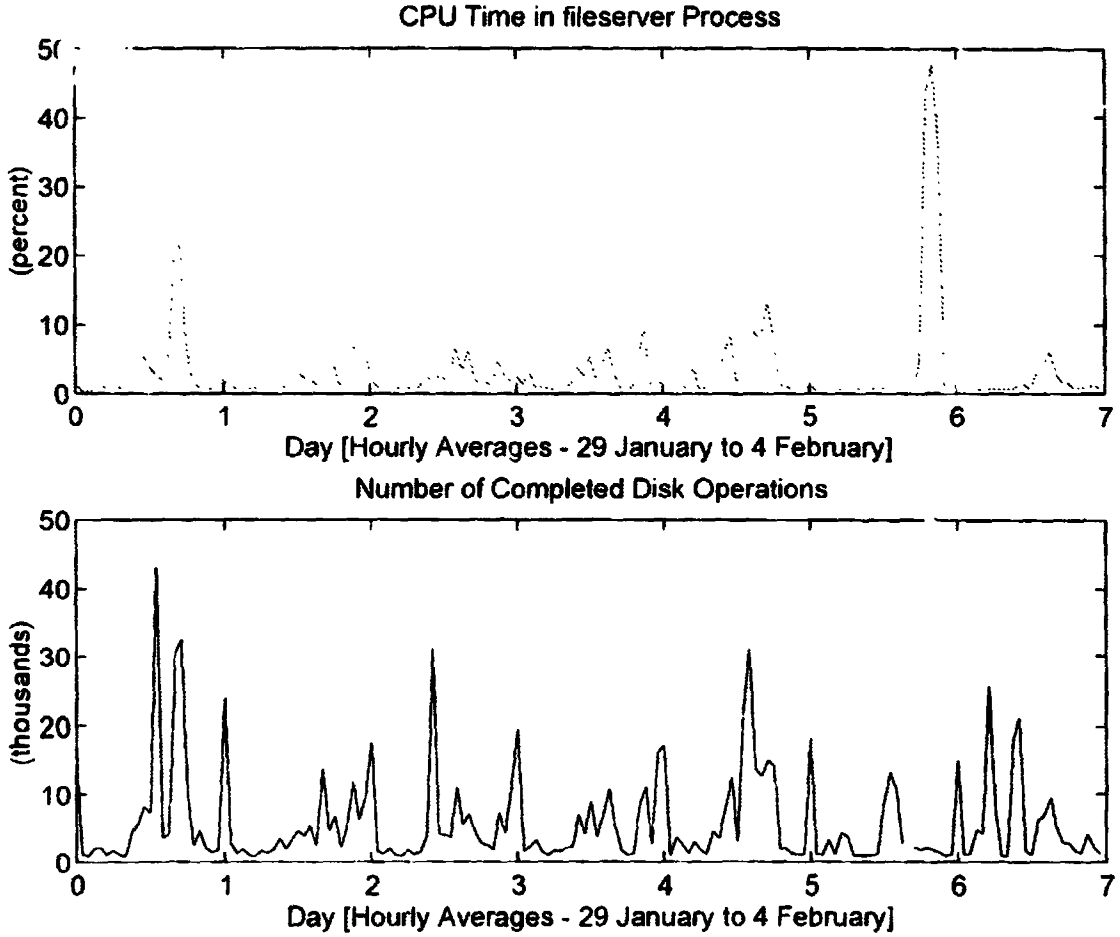


Figure 3 - CPU and Disk Utilization

A similar effect is seen in the plot of disk activity in the lower chart of Figure 3. This chart shows the total number of physical disk accesses completed in each of the same 10 minute intervals It is harder to talk about percentage utilization in this case, but the three drives on this server should be able to sustain considerably more than the 50,000 accesses/hour (14 accesses/second) that correspond to the highest point on the chart The average is less than one access/second over three disks Again, a negligible total average load

Simply looking at these numbers, we might be tempted to conclude that this five year old machine is performing adequately and there is no need for an upgrade at all.[3] So how do we explain our users' complaints? We clearly needed some other measure that we could use to gauge users' perception of the performance of the system Since overall utilization is not the problem, we surmised that looking at response time might prove more enlightening

[3] In fact, the upgrade policy at large AFS sites is rumored to be generally insensitive to utilization as well The algorithm used can roughly be paraphrased as, when customers complain, begin with the oldest component of the system and continue to replace equipment with newer models until complaints subside

4.2 Client Response Time

The client data that we collected provided hourly samples of the number and total elapsed time of all AFS operations of each type completed by that client in that hour We chose to use the average response time for FetchStatus operations as our measure of user-visible performance because 1) it is the most frequently-called operation, 2) in the absence of outside influences, it does an approximately constant amount of work on each call (since data fetches in AFS may be as large as several hundred kilobytes, but most files are much smaller than this, FetchData delays are expected to be much more variable) and 3) we found an r^2 coefficient of determination suggesting that 50% of the variation in the response times of FetchStatus and the per-kilobyte latencies of FetchData are correlated, as shown in Figure 4

If we again look at the average response time in Figure 4, we see significant variation - ranging over an order of magnitude. We hypothesize that users of AFS, accustomed to local disk access times (due to high local cache hit ratios described above) will be significantly affected by high variance in response times, particularly when the effect lasts for significant lengths of time, such as the hourly intervals shown in this chart Based on this, we began searching for the causes of high variance in user response time

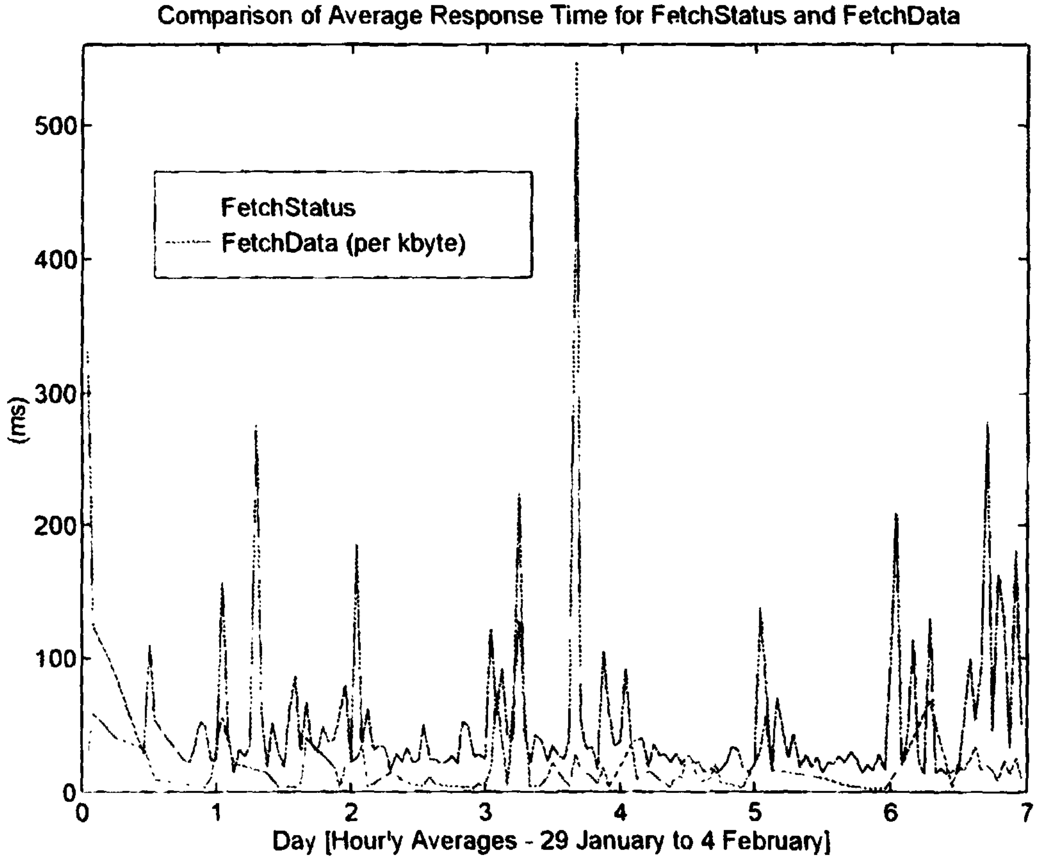


Figure 4 - FetchStatus and FetchData Performance

In order to convince ourselves that our AFS server upgrade had indeed been worthwhile, we performed an experiment to compare the performance of our old server and our new server under the same workload. The numbers in Table 4 show the results of this controlled experiment. One test client was constantly performing `stat()` calls at random into a directory of 2,000 files. At the same time, a second client was running a "competing" workload by continuously reading a large file from the same partition on the same server. Both clients flushed their caches at the end of a cycle so that all operations wǝre handled at the server The table shows the average response time of the FetchStatus operations that resulted from the `stat()` calls, the number of FetchStatus operations completed in the five minute measuring interval, and the average throughput of the competing process

Machine	SPECint92	Average FetchStatus Response Time (ms)	Number of Operations	Competing Read Transfer (KB/s)
SPARCstation 1+	14 0	25.9	8,486	212.7
SPARCstation 20	69 0	16.7	15,291	343.8

Table 4 - Direct Comparison of Server Platforms

From this experiment, we see that the increased CPU performance of the newer machine reduces average FetchStatus response time by 35% at periods of high server load. At the same time, the faster machine can complete almost twice as many FetchStatus operations in the same time interval while also providing 62% higher data throughput Since more server processing power is clearly effective for improving client performance, we expect to be able to find a dependence between server CPU utilization and client response time in our trace data.

4.3. *Impact of the Network*

When we first compared the CPU and disk utilization trace to the FetchStatı , response time trace, we were unable to find a significant correlation between times of slow user response and times of high server utilization This unintuitive result led us to look for other factors that might explain performance at the clients The most obvious factor in a distributed system is the network between machines, so this is the parameter \.e examined next

The top chart of Figure 5 shows the average network round-trip time of pings on the lab and machine room Ethernet segments c er one hour periods We see a mean of 9 0 ms and a standard deviation of 7 2 ms on the server network, and 16 9 ms 15.8 ms on the lab segment, where most of ı.ıe clients were located. The lower left portion of Figure 5 shows the graphical correlation between the response time of the network and FetchStatus response time[4] Although not a strictly linear relationship, the Pearson r^2 coefficient suggests that 35% of the variation ir. the response times can be attributed to variation in network performance To focus on this relationship, the correlation graph in the lower right of Figure 5 reports only those hours where average ping time was larger than 20 ms In this figure, a linear relationship between server response time and network response time is more plausible This matches our expectations that the network connecting the machines in a distributed system is a considerable factor in overall performance It is for this reason that the new server .nd many of our clients are being outfitted with switched ATM networking dedicated to the PDL in addition to the existi..g Ethernet However, we

[4] Directly correlated data, with 100% of the variation explained, would appear as a straight lır ın these graphs.

also see that network response time is not a complete explanation of client response time variance.

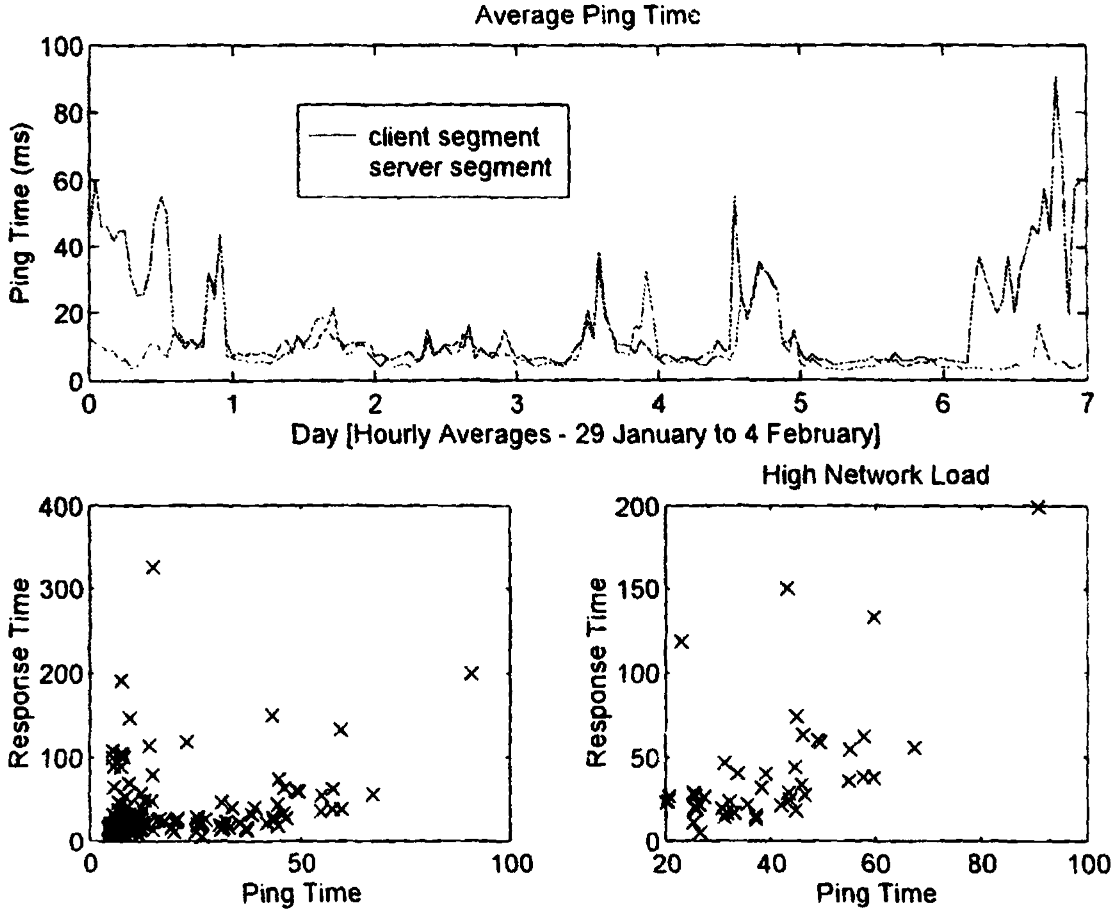


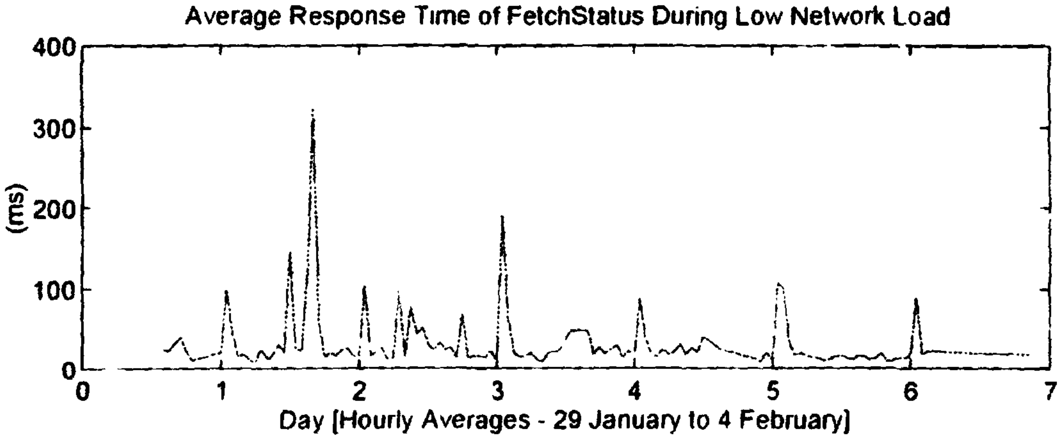
Figure 5 - Correlation of Response Time with Network Behavior

4.4. *Impact of Shared Resources*

Our next step was to again compare server utilization (Figure 3) to average client response time after the periods of high network load are eliminated from the response time trace (see the top chart of Figure 6) Again, we were not able to explain as much of the remaining variance as we expected Seeking an explanation for this disappointment, we did notice an effect that we had not considered in our initial analysis. Although all of the project volumes for the target group were on the server we were tracing, home directories and shared binaries were being accessed on servers shared across the department [5] Since we were looking at all FetchStatus operations performed in hour-long intervals, load on these shared servers could have a significant impact on user response time We see a significant r^2 coefficient of 65% between clients of the same system type, suggesting that about 65% of the variation in a single client's response time trace is explained by the variation on the average response time trace of machines of the same system type At the same time, we see a strong anti-correlation (r^2 coefficient of essentially zero) with clients of different system types. The plots at the bottom of Figure 8 show the correlation between the response time seen at millburn (an RS/6000) and

[5] Instrumenting all of the servers and clients in our environment would have been impractical due to the system changes necessary and the sheer volume of users we would have had to persuade to participate

response time at other RS/6000s and, in the right plot, the correlation between millburn and some of the Alpha AXP machines in the study. Not surprisingly in hindsight, our mistake was to overestimate the effectiveness of the replication of commonly used binaries and underestimate the frequency with which users' home directories are used in the course of project work Although most of the user data may be stored on a fast server, binaries and home directories stored on shared, slow servers may be a considerable drag on user-visible performance [6]



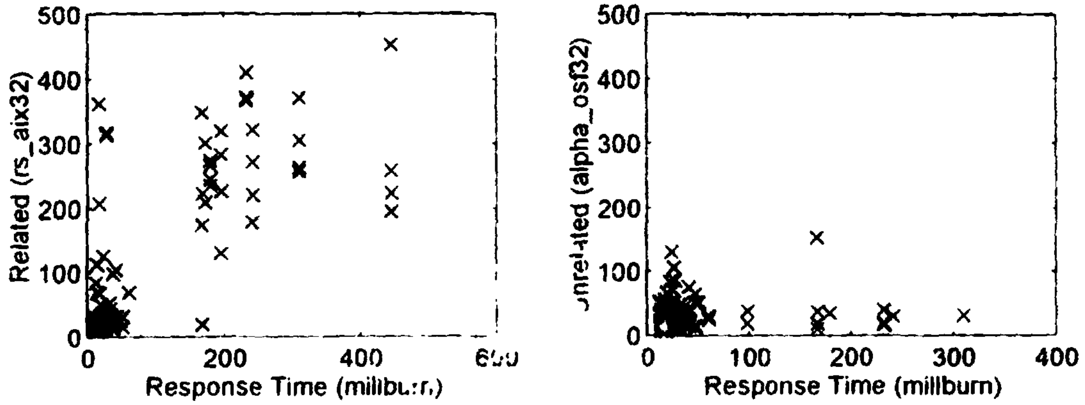


Figure 6 - Correlation of Response Time by Client System Type

4.5. *Impact of Server Utilization*

In order to minimize the effect of interaction with servers other than the one we are tracing, we filtered the response time data to include only those periods when a host was active on our server [7] Figure 7 shows the graphical correlation between average FetchStatus response time and server disk activity and average FetchStatus response time and server CPU activity It is apparent from the leftmost correlation chart of Figure 7 that much of the response time is not correlated with server activity, but as we could not

[6] We will be taking a closer look at this effect and will be placing read-only replication sites of the most-used shared files on our upgraded server to improve our overall performance

[7] A host was classified as being active on the server in a particular hour if it appeared in the . *rxdebug* output at the end of the hour Since *rxdebug* provides information only for those clients the server has recently interacted with, this does not completely eliminate, but should significantly reduce, the fraction of "foreign" FetchStatus requests in the averages

extract the delays associated with central AFS servers, we expect some amount of uncorrelated points Neglecting data points with less than 500 disk accesses per hour in the center plot, we see an r^2 correlation of 25%, as response times are impacted by the amount of disk work (dominated by FetchData operations) the server is already processing when new requests arrive. In the rightmost correlation plot, we see an even closer correlation with CPU utilization (for the same set of points as in the center plot where the disks are busy) which explains just over 50% of the variation in response time. This suggests that poor response times occur when the server CPU and disk are busy (after network and "foreign" server effects have been accounted for). This result fits well with our prior observations that a considerable number of cycles are required to move data from a disk, through the user-level *fileserver* process, back into the kernel, and onto the network and that these numbers scale with the amount of data being moved [Gibson90]

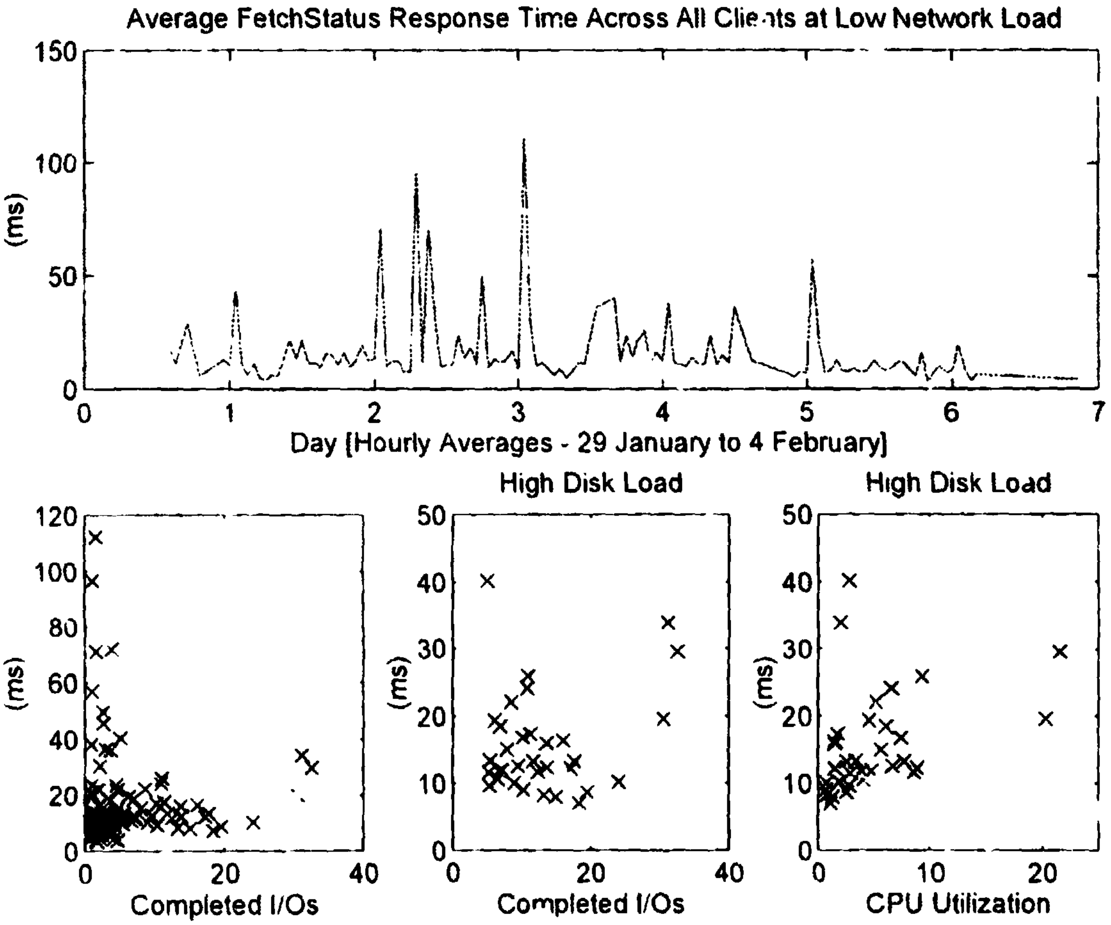


Figure 7 - Correlation of Response Time with Disk Activity and CPU Utilization

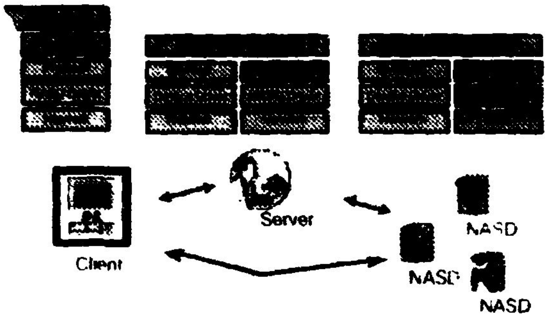
We have finally discovered the correlation we have been seeking - a faster server CPU benefits AFS users because there are bursts of CPU activity, specifically when data is being transferred, during which server load leads to poor client response times.

5. Network-Attached Storage

5.1. *Opportunities for Network-Attached Storage*

Recalling Figure 2, which shows how the distributed file server machine acts as an intermediary, copying data between the client network and the storage interconnect, we would like to develop techniques for reducing server utilization during periods of intense transfer workloads In fact, because of the speed addressability, and distance limitations of SCSI cabling, new storage interconnects such as Fibre Channel are increasingly similar to client network fabrics With this convergence in mind, we propose that the client and storage networks discussed in Section 1 be combined into a single fabric As illustrated in Figure 8, this creates the opportunity for disks with sufficient intelligence to perform a significant fraction of the clients' file operations without the need for intervention from the distributed file server [Gibson96]

Eliminating the server machine as a bottleneck for data transfers between storage and applications provides a significant opportunity for improving overall performance. By not involving a third party, common case transfers are considerably faster and the number of requests that can be serviced at any given time should be increased Data transfer functions are off-loaded to the network-attached devices and the server would be responsible only for "higher-level" distributed file system functionality



Network-Attached Storage

Figure 8 - Network-Attached Storage Architecture

There is a range of possible configurations for such a system At one end of the spectrum, Network SCSI is being promoted by several vendors as a means of providing third-party transfer between clients and drives attached directly to the network [Seagate96] All commands are processed by a server which uses the SCSI third-party transfer interface to instruct drives to transfer data directly to clients At the other end of the spectrum, dedicated Network File System (NFS) or Netware servers [NetApp96, NetFrame96] are storage systems that directly implement these distributed file system protocols, backed by specially optimized hardware configurations Network-attached storage proposes to provide an intermediate point The distributed file system server would continue to be responsible for operations such a file opens and metadata management, but drives would have sufficient intelligence to handle data transfer requests without server intervention for each individual request In order to achieve the desired scalability and performance, it may also be necessary to have file status and inquiry functions handled at the drives [Gibson96]

This direct transfer concept is not a new one In 1991, Randy Katz described the basic advances that make network-attached devices feasible [Katz91] The High Performance Storage Systems project [Watson95] is exploring these technologies in the context of large MPP and SMP systems based on the framework of the Mass Storage Systems Reference Model [Miller88] Van Meter provides a survey of current products and major research issues, including security, network protocols, and the changes in operating system paradigms necessarv to efficiently support network-attached devices [Van Meter96]

Such an architecture raises several important issues Can the drive be made sufficiently intelligent at a reasonable cost? How do we ensure the security and integrity of the data being stored? Can enough of the server functionality be off-loaded to significantly improve both throughput and scalability? How effective will this architecture be for meeting the needs of the clients in a distributed system?

5.2 *Implications of this Study for Network-Attached Storage*

The biggest lesson that we take away from the preceding analysis is that the mean behavior of the system is essentially irrelevant. Even though the system is 97% idle when measured in total, it is the high load periods that matter to customer satisfaction. As Table 5 shows, peak loads, even at the granularity of an hour, are much higher than average loads. Moreover, the distribution of operations measured over the long term, shown on the left of Table 5 and similar to previous studies [Spasojevic96] is not preservea in these peak periods - data activity is nearly twice as common in these peaks With customer satisfaction sensitive to response time variation, the server performance during peak loads is likely to be more important than at other times

Server Operations	Weekly Total			Peak Hour	
	total	fraction	hourly	total	fraction
FetchStatus	412,695	70 6%	1,247	6,209	45 3%
StoreStatus	22,642	3 9%	134	175	1.3%
FetchData	62,288	10 7%	370	4,219	30.8%
StoreData	32,414	5.5%	192	147	1.1%
CreateFile	17,089	2 9%	101	52	0 4%
RemoveFile	20,422	3.5%	122	2,587	18 9%
GiveUpCallbacks	17,298	3 0%	103	326	2 4%
total	584,848		2,269	13,715	

Table 5 - Distribution of Server Operations

Given a high emphasis on the server performance during peak loads, off-loading the high-cost data movement operations, as proposed by the network-attached storage architecture, should decrease the variance in user response time significantly, even though overall averages will simply be reduced from a small number to an even smaller number The appropriate analogy is not to system throughput, but something closer to the way reliability is measured. Changing the mean time to data loss (MTTDL) of a system from 10 years to 100 years does not mean that one expects the system to last ten times as long, but that the probability of a failure occurring within the next hour is reduced by an order of magnitude We suggest that there is an analogous measure for distributed file systems, the mean time until burst (bad) performance (MTTBP) which should be increased so that the probability of poor response times in any given hour of work is decreased We would expect users to be pleased if the occurrence of a period of bad response time were reduced from once a week to once every 3 months

6. Conclusions

Modern distributed file systems such as AFS very successfully cache file data on client machines While this ensures that average response time is low, it also ensures large variance in response time because operations that must contact remote servers are much slower Direct measurement of these remote servers show that their overall utilization can be quite low, 3% in our data, while users are simultaneously sufficiently dissatisfied with performance to pay for a faster server. This study shows that the faster server is in fact needed because, although 97% idle overall, these file servers can be intensely overloaded during bursts of activity, leading to periods of poor response time long enough to disgruntle users

In addition to focusing our attention on burst server loads, our analysis shows that the distribution of operation types during bursts is different from overall distributions. Servers should be optimized for workloads with much more data transfer than the overall distribution suggests.

These results confirm our intuition that network-attached storage, if it can re-route most data transfer directly to storage devices, has the potential to reduce customer response time in two ways - 1) it avoids the copying steps at the server and 2) it off-loads the work of data transfer from the server, reducing the chance of a bust of overutilization

Out future work, then, is to evaluate the client performance on such network-attached storage architectures and demonstrate the implications on distributed file system design

7. Acknowledgements

This research is sponsored by DARPA/ITO through ARPA Order D306, and issued by Indian Head Division, Naval Surface Warfare Center, under contract N00174-96-0002 The views and conclusions contained in this document are those of the authors and should not be interpreted as representing official policies, either expressed or implied, of any sponsoring or supporting agency, including the Defense Advanced Research Projects Agency and the United States Government

We would also like to thank Chris Demetriou and Jim Zelenka for their help is collecting the trace data In addition, this work was complemented by discussions among all the members of our research group, including David Nagle, Khalil Amiri, Fay Chang, Eugene Feinberg, Howard Gobioff, Chen Lee, Berend Ozceri, David Rochberg, and Hugo Patterson

8. References

[Spasojevic96] Spasojevic, M. and Satyanarayanan, M. "An Empirical Study of a Wide-Area Distributed File System" *ACM Transactions on Computer Systems*, May 1996.

[Howard88] Howard, J. et. al. "Scale and Performance in a Distributed File System" *ACM Transactions on Computer Systems*. Volume 6, Number 1. February 1988, pp. 51-81.

[Mummert94] Mummert, L. and Satyanarayanan, M. "Long Term Distributed File Reference Tracing: Implementation and Experience" *Technical Report CMU-CS-94-213*. November 1994.

[Kirk90] Kirk, R. Statistics: An Introduction. Holt, Rinehart and Winston, Inc. 1990, pp. 155-190.

[Ruemmler93] Ruemmler, C. and Wilkes, J. "UNIX disk access patterns" *Proceedings of the USENIX Winter 1995 Technical Conference*. January 1993, pp. 405-420.

[Baker91] Baker, M. et. al. "Measurements of a Distributed File System" *Proceedings of the 13th Symposium on Operating Systems Principles*. October 1991.

[Gibson96] Gibson, G. et. al. "A Case for Network-Attached Secure Disks" *Technical Report CMU-CS-96-142*. June 1996.

[Seagate96] Seagate Corporation "Barricuda Family Product Brief (ST19171)". June 1996.

[NetApp96] "Network Appliance Advantage" *http://www.netapp.com/products*. July 1996.

[Netframe96] "The ClusterServer 8500 Series" *http://www.netframe.com/products*. July 1996.

[Katz91] Katz, R., "High-Performance Network- and Channel-Attached Storage" *Proceedings of the IEEE*. Volume 80. August 1992.

[Watson95] Watson, R. and Coyne, R. "The Parallel I/O Architecture of the High-Performance Storage System (HPSS)" Fourteenth IEEE Symposium on Mass Storage Systems. September 1995, pp. 27-44.

[Miller88] Miller, S. "A Reference Model for Mass Storage Systems" *Advances in Computers*. Volume 27. 1988, pp. 157-210.

[Van Meter96] Van Meter, R. "A Brief Survey of Current Work on Network Attached Peripherals" *Operating Systems Review*. Volume 30, Number 1. January 1996.

NEXT
DOCUMENT

Mass Storage and Retrieval at Rome Laboratory

Joshua L. Kann, Brady W. Canfield, Capt, USAF, Albert A. Jamberdino, Bernard J. Clarke, Capt, USAF, Ed Daniszewski, and Gary Sunada, Lt, USAF

Rome Laboratory
IRAP
32 Hangar Rd.
Rome NY 13441-4114
kannj@rl.af.mil
315- 330-4581

Abstract

As the speed and power of modern digital computers continues to advance, the demands on secondary mass storage systems grow. In many cases, the limitations of existing mass storage reduce the overall effectiveness of the computing system. Image storage and retrieval is one important area where improved storage technologies are required. Three-dimensional optical memories offer the advantage of large data density (on the order of 1 Tb/cm^3) and faster transfer rates because of the parallel nature of optical recording. Such a system allows for the storage of multiple-Gbit sized images, which can be recorded and accessed at reasonable rates. Rome Laboratory is currently investigating several techniques to perform three-dimensional optical storage including holographic recording, two-photon recording, persistent spectral-hole burning, multi-wavelength DNA recording, and the use of bacteriorhodopsin as a recording material. In this paper, the current status of each of these on-going efforts is discussed. In particular, the potential payoffs as well as possible limitations are addressed.

1.0 Introduction

The national security requirements of the United States have undergone fundamental changes in just a few short years. The cold war and incumbent strategic threats have given way to new Third World threats and regional conflicts. In order to achieve Global Awareness and, if necessary, implement Dynamic Planning and Execution, vast amounts of information must be collected, stored, processed, and disseminated through an interoperable Command, Control, Communications, Computing, and Intelligence (C4I) architecture.

Air Force C4I systems must effectively store, retrieve, and manage massive amounts of digital data. Optoelectronic and massively parallel computing demands multi-terabit memories and near real-time write and retrieval rates. Current Air Force systems range from centralized Terabyte and Petabyte storage comprised of large objects (images) to distributed heterogeneous databases that contain many small and large objects (open source databases). Although technologies for storage, processing, and transmission are rapidly advancing to support centralized and distributed database applications, more research is needed to handle massive databases efficiently and achieve the Air Force goal of "Information Dominance".

Our goal is to retain data for potential future analysis in a cost-effective manner. The more relevant data would remain on-line, say for five years, organized with the most relevant data accessible in the least amount of time. It is expected that 2 to 5 Terabytes of new data will be processed each day. Thus the total size of the database (both on-line and off-line) could be as large as 20 Petabytes with about 300 Terabytes of data stored on-line. It becomes apparent that new storage devices (primary, secondary and, even, tertiary) for large multimedia databases, as well as data pathways with the required capacity, must be developed. Access time is about 5 seconds for the data less than a week old, about 30 seconds for data under two months old, and on the order of minutes for the data up to 10 years old.

Over the years, Rome Laboratory has nurtured a comprehensive program, developing revolutionary new storage techniques that meet the various demands for data storage and retrieval. Our group is currently investigating various approaches to ultra dense, highly parallel three-dimensional optical memory storage systems. This article traces the recent history of optical data storage and updates the status of Rome Laboratory's research efforts in this field.

In the mid-1970s and early 1980s optical storage reached the consumer market. Industry giants like RCA and Philips developed and marketed playback devices and large format "laser disks" for home movie viewing. While laser disks never generated a large, broad-stream consumer market (VCR is still the dominant technology for home movie viewing), compact disks (CDs) are now the primary means of distributing and listening to high-fidelity music. The introduction of laser diode devices made compact disk systems a viable consumer product. Laser diodes operating within the near-IR (infrared) spectrum allowed 1μm embossed pits to be easily detected. The new laser technology, in combination with powerful error detecting and correcting codes, enabled SONY and Philips to introduce the first CD audio product a decade ago.

Better optical media, more powerful laser diodes, and very precise, low-mass optics have propelled optical disk technology to a practical, powerful system[1-3]. The next-generation device introduced in the mid-1980s provided a flexible write-once, read-many (WORM) capability. This enabled end-users to record and playback computer data from the same drive. Rome Laboratory has continued to sponsor work in this area to further exploit the benefits of new storage technology. An early Rome Laboratory prototype used an argon laser to record and playback digital data from a 12.5-inch plastic-based optical disk. Further inve^tment led to the delivery in 1982 of a large-capacity optical jukebox for satellite imagery storage and retrieval applications. The jukebox holds 100 WORM disks that provide a storage capacity of one Terabyte.

The third generation optical disk, today's rewritable systems, offer record, playback, and erase capability. These magneto-optical (MO) disks are composed of a rare-earth alloy and transition materials, which often include terbium, iron, and cobalt elements. Optical disk storage is playing a larger role in mass data storage for many military applications; particularly, those applications that require reliable operation under harsh operational environments. Commercially, MO disk drives have also made an impact on the market.

Present devices store one-dimensional (serial) information in a two-dimensional plane. Three-dimensional memory devices store two-dimensional information (bit planes) throughout a volume. A 3-D memory is, therefore, a single memory unit where three independent coordinates specify the location of information. 3-D memories are generally classified as bit-plane-oriented and holographic. Bit-oriented 3-D memories, where each bit occupies a specific location in 3-D space, differ significantly from 3-D holographic memories. With holographic memories the information associated with stored bits is distributed throughout the memory space. Bit-oriented 3-D memories generally use amplitude recording media while holographic memories use phase recording media. In bit-oriented 3-D memories, the coordinates that specify the location of information can be spatial, spectral, or temporal, giving rise to a variety of 3-D memory concepts that use different materials with various properties. For example, materials that exhibit 2-photon absorption are the basis for true volume memories, while materials wherein spectral holes can be burned, provide a storage medium for spectral/spatial storage. In addition, materials that exhibit the photon-echo effect could, in principle, lead to temporal/spatial storage.

In 3-D memory, information is partitioned in binary planes that are stacked in the third dimension. One memory operation is performed on the entire bit plane, giving rise to a tremendous memory bandwidth increase over conventional 2-D bit-oriented memories. Also, by storing information in volumetric media, we can achieve very high density (10^{12} bits/cm^{3}). High-speed reading and writing of an entire memory plane then becomes feasible. These considerations make 3-D memory very compatible with emerging, highly integrated parallel array processors and optoelectronic multiprocessors.

In this paper, we discuss the current status of our ongoing efforts in several areas of 3-D memory. Section 2 investigates a variety of 3-D memory storage techniques. Holographic and two-photon storage methods are discussed in detail. Both system and material aspects are addressed. In addition, a less involved discussion is given for three other techniques: Persistent Spectral Hole Burning (PSHB), recording in DNA molecules, and the use of bacteriorhodopsin (BR) as a storage medium. Finally, in Section 3 a summary and some conclusions are provided

2.0 Various three-dimensional recording techniques

2.1 Holographic data storage Holograms for artistic posters and pictures started gaining popularity in recent years due to the maturation of technologies which produce very pleasing images. These same technologies provide a potential to store data using holography, with smaller system sizes but larger data capacity and faster throughput rates[4-6]. This idea is not new to the data storage world, and dates back to the late 60's, but was only recently considered practical due to the emergence of improved critical components such as improved laser diodes, Spatial Light Modulators (SLM), and Charge-Coupled Device (CCD) detector arrays. Holographic memories also process data in parallel, affording fast data transfer rates, high storage density, and small physical size.

A hologram is a recording of the amplitude and phase of a wavefront, which is in contrast to photography, where only the amplitude of the wave is recorded. A hologram is created, or written, by the interference of two beams, an object beam and a reference beam, as shown in Fig. 1. When a hologram is read using one beam, the original wavefront (amplitude and phase) is reconstructed, as shown in Fig. 2. To create the extremely high data capacity, data is stored in the same physical space of the media, but is written and read by multiplexing. Multiplexing is a way of giving each set of data a unique address. Typical multiplexing methods used for holographic storage are angular, spatial, and phase multiplexing.

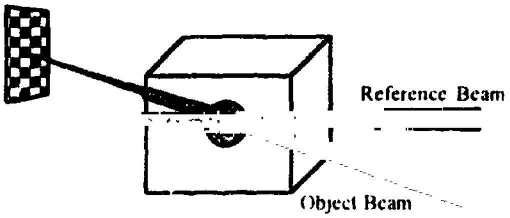


Figure 1. System used to record holographic data. A reference beam and object beam are interfered within the recording medium. The resulting interference pattern is recorded within the medium.

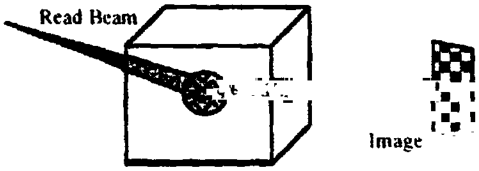


Figure 2. System used to read previously recorded holographic data. The reference beam playback the recorded data. The data is then imaged onto a CCD.

Angular multiplexed holograms, shown in Fig. 3, change the angle of incidence of the reference beam on the storage medium without changing the location of the area, or spot, being recorded. This is one method of recording several holograms, or pages of information, on top of each other in the same location. Figure 4 shows how angular multiplexing changes the reference beam angles but the recording location does not change. The schematic diagram shows three collimated beams with different angles of incidence on the storage medium. Note all three beams stay within a common location. An important consideration for angular multiplexing systems is the need for sophisticated beam steering devices, since without it this method can result in some very complex optical paths for large storage systems.

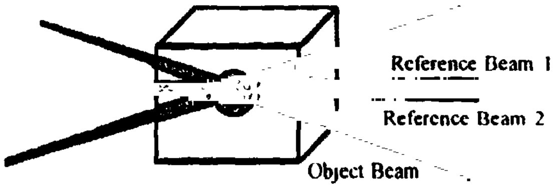


Figure 3. Basic premise of angle multiplexing. Multiple holograms stored in the same spot, all recorded with a reference beam at a different angle.

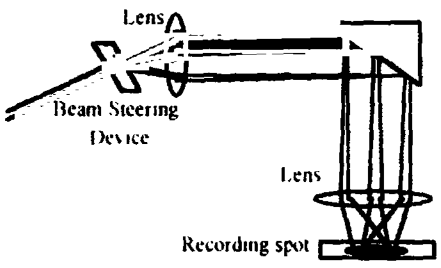


Figure 4. A system used to perform angular multiplexing. A beam steering device is used to modulate the angle of incidence of the reference beam.

Phase multiplexing uses reference beams with different phase fronts to write composite holograms associated with the object beam. Each reference beam consists of a set of plane waves with a unique phase distribution. This phase distribution represents the address of the recorded information. Holographic memories using phase multiplexing have the same data storage capacity as angle multiplexing, but involve different problems generating the phase codes. We are working with Surface Optics Corporation to develop a compact 3-D storage system using this method, which will store 26 Gigabytes and have a footprint of one square foot for the entire system.

Spatial multiplexing is a method used to change the recording spot location to a different location in the medium. In volume holography, holograms recorded on top of each other will ultimately reach saturation of the recording medium unless redirected to a different spot. Spatial multiplexing can be combined with angle or phase multiplexing to increase the storage capacity of the memory device. Holograms can be stored in one spot up to their practical limit using angular or phase multiplexing, then the information beams are redirected to a different spot using spatial multiplexing to fill the next area with the new pages of information.

Optimizing the number of holograms per spot is the key to finding the best system. The most successful efforts apply angle and spatial multiplexing. Researchers with the Psaltis Group at the California Institute of Technology have demonstrated the storage of 160,000 individual holograms in a 2 cm x 2 cm x 4 cm crystal of $LiNbO_3$. The photorefractive $LiNbO_3$ crystal is thick, and can have a high number of holograms per spot. The thickness determines the number of holograms that can be stored in a given area of material before reaching the point where saturation occurs and the data resolution decreases. The higher the density of data, the lower its resolution. However, thick holograms require more complex optics for data addressing. Thin holograms cannot store as much as thick holograms, but can have simpler optics, and therefore a simpler system. We are funding two efforts with Holoplex, Inc to make data storage systems out of thin film photopolymer disks. Each of these disks will store 2.8 Terabits, with 17 ms access time, and each disk can be recorded in 60 sec. Overall holographic storage looks more reliable than current solid state or disk memories since the individual bits of information are stored collectively throughout a homogeneous recording medium. Recorded information is not susceptible to the same kind of losses magnetic or magneto-optic devices have from dust, scratches, or other imperfections. A holographic recording medium is generally a solid volume of homogeneous material, requiring few manufacturing constraints to make, where semiconductor memories require fine line lithography. Since information is stored optically, the system is a "free-space" system and does not require elements of the system to be in contact with each other to operate, leading to greater speed and reliability.

2.2 Two-Photon Storage In addition to our holographic work, we are also investigating 3-D memory storage which utilizes two-photon absorption[7-8]. Currently, we are working with Call/Recall to develop a two-photon based optical memory. Such a system is capable of data capacities up to 10^{12} bits/cm^3. This system takes advantage of the parallel nature of optical recording, allowing for a page of digital data to be written (or read) simultaneously. We have invested in this technology for the past 5 years and have recently developed a demonstration WORM system. It is hoped to have a functional read/write/erase system within the next few years. Currently, a great deal of work is being done both in the system and material areas of this technology. The basic premise of a two photon recording system is the simultaneous absorption of two photons whose combined energy is equal to the energy difference between the initial and final states of the recording material. This simultaneous absorption results in a change in the molecular structure of the material. This structural change alters various properties of the material, including index of refraction, the material's absorption spectrum and the material's fluorescence spectrum. Therefore, by intersecting two optical beams, either spatially or temporally, the material's optical properties can be altered locally and addressed anywhere within a three-dimensional space. Digital data is written in this fashion, with the ultimate limitation on storage capacity set by diffraction effects in the optical system. Data readout is accomplished by probing the material with a single read beam to measure a change in one of the material's optical properties.

The current system architecture uses picosecond pulses of the first and second harmonic (λ_1 = 1064 nm, λ_2 = 532 nm) of a Nd:YAG laser. A schematic of the optical system is shown in Fig. 5. Image storage (recording) is shown in Fig. 5(a), while image retrieval (readout) is shown in Fig. 5(b). Digital information is recorded in the two-photon material as pages of digital data, the data planes separated in the axial direction. (Currently, the material is fabricated in a cubic shape, but we are investigating other geometries.) The input (data) arm of the system (λ = 532 nm) is spatially modulated with a SLM and imaged to the proper plane within the cube. A second, addressing beam (λ = 1064 nm),

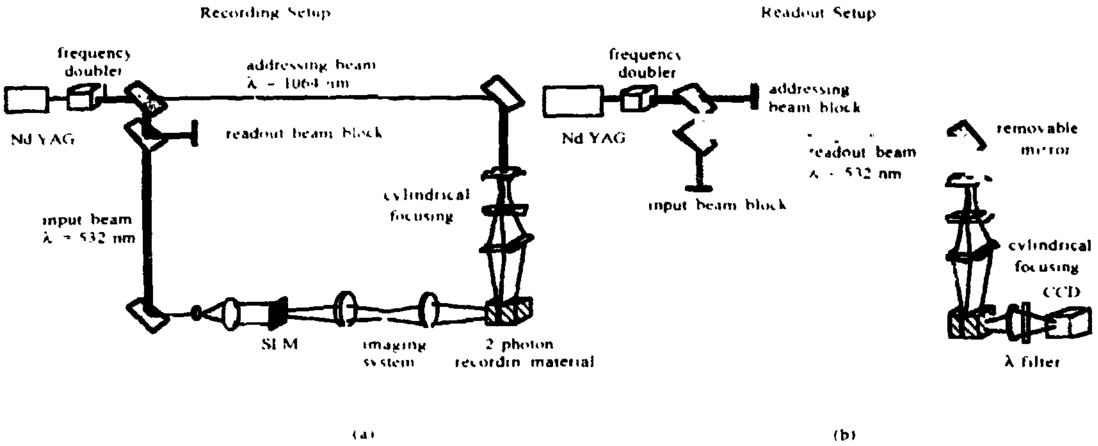


Figure 5. The recording (a) and readout (b) systems used for a 2-photon memory. In (a), recording occurs where the spatially modulated input beam and cylindrically focused add^r^essing beam overlap. The system is readout by measuring the excited fluorescence of each recorded data plane as shown in (b).

propagating orthogonal to the input beam, is cylindrically focused throughout the cube. A page of data is recorded when the inut beam and addressing beam simultaneously illuminate the image plane. The axial 'esolution is limited by the divergence of the addressing beam, while the lateral resolution is determined by diffraction effects and/or aberrations in the input arm. A more detailed discussion of data parallelism versus density is given later in this section.

Readout is accomplished b' ıeasuring the fluorescence of the material. The input beam is blocked and the 532 nm bear. is cylindrically focused to read the proper data page. For the unv :tten material, the 532 nm beam is unab_orbed and passes through the system. However, for the written form of the material, the readout beam is absorbed, exciting a fluorescence at a longer wavelength. The readout plane is then imaged through a wavelength selective filter onto a CCD. Thresholding of the individual pixels of the CCD is performed to digitize the data.

A variety of recording materials have been proposed. The proper material must posses several key characteristics in order to be considered. Some of these characteristics are: photocron:isim (the change of chemical structure after excitation by light), a fluorescence in one of the two chemical states, stability of both states at room temperature, ability to read data 10^6 times without loss of information, high quantum efficiency of the read form fluorescence, and a wide enough wavelength shift between the read beam and peak of the fluorescence spectrum to prevent cross-talk. In this paper, we will discuss one of the more promising candidates, Spirobenzopyran (SP).

Spriobenzopyran molecules are composed of two distinct molecular components linked by a sp^3 hybridized carbon. Upon excitation simultaneous absorption of a 532 nm photon and a 1064 nm photon the chemical state of the material is altered, as shown in Figs. 6(a) and 6(b). The unwritten form, Fig. 6(a), is colorless in appearance and shows strong absorption in the ultraviolet (λ < 400 nm) portion of the spectrum. However, the written form, Fig. 6(b), is colored in appearance and shows strong absorption in the 550 nm region. The fluorescence spectra of SP, shown in Fig 6(c), is peaked around 600 nm. Note that the this peak is sufficiently separated from the read wavelength to prevent cross-talk during readout. Figures 6(a) and 6(b) show two species of SP. To fabricate solid, stable recording materials, the SP is dispersed in polymers such as PMMA. This alters the absorption and fluorescence spectrums slightly, but does not effect the basic system premise discussed in the previous paragraphs.

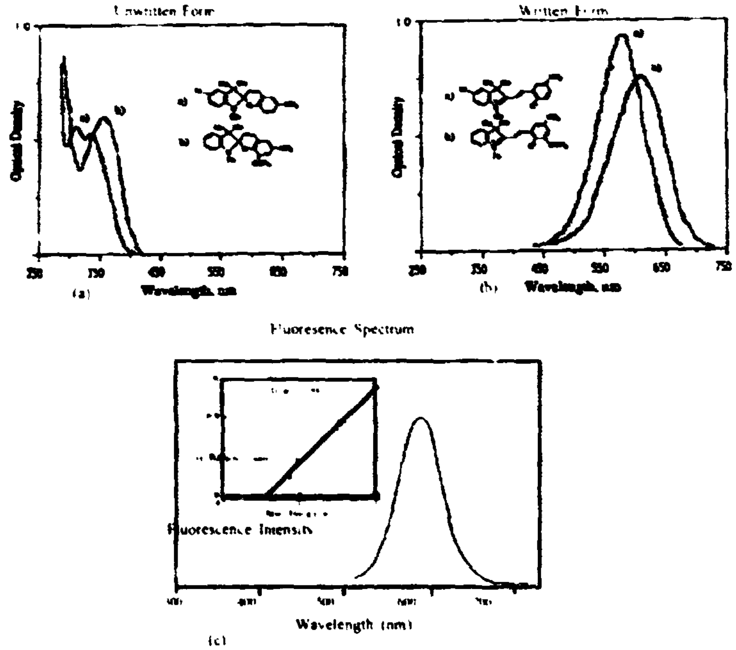


Figure 6. The absorption and fluorescence spectrum for two different types of SP compositions. In (a) the spectrum for the unwritten form is shown, while in (b) the spectrum for the written form is shown. In (c) the fluorescence of the written form is shown.

Currently, a working WORM system has been built and is used for writing and readout of pages of digital data. The system uses a 1cm^3 cube of SP for a recording material. The system has been used to record 100 planes of data with a separation of 80 μm between planes. The input beam is spatially modulated with the use of chrome masks. The mask's resolution is 100 x 100 μm^2. Dynamic focusing is performed by moving the cube. Currently no dynamic aberration correction is performed. With a demagnification in the system of 3.4, the recorded bits are 30 x 30 x 80 μm^3 resulting in a density of 1 Mb/cm^3. Data readout is accomplished with the use of a HeNe laser (λ = 543.5 nm), and a 640 X 480 CCD array. While the system's density is well below our desired goal of 1 Tb/cm^3, the system allows us to demonstrate and improve on the current technology.

Using this system initial Bit Error Rate (BER) characterization will be performed. In order to be competitive with existing mass storage systems, raw BERs on the order of 10^{-6} must be obtained. In addition, we have begun work with the University of Pittsburgh on an optoelectronic cache memory system which will act as cache memory between the optical memory, and a uniprocessor or multiprocessor computing environment. By caching a page of data during a memory read, the relatively slow access time to the user will be significantly lowered, since the access time of the cache memory is orders of magnitude faster than the two-photon memory.

There are several obstacles to overcome in this rapidly developing technology. The first involves cube fabrication. Important characteristics are surface quality and material homogeneity. Surface quality is especially important, since defects in the material's surface will result in scattering of the light beam and optical aberrations of the transmitted beam. Using injection molding techniques, it is hoped to produce surfaces with RMS roughness of approximately 0.25 μm. Another significant obstacle is the current need for high intensity beams for data recording. The probability of recording a mark (i.e. causing a local transition of the material from the unwritten to written form) is proportional to the product of the two beams intensities. Typically very high intensities are needed to alter the material (intensities on the order of 1.6 GW/cm^2 are typical). In addition to designing the materials for lower powers, we are currently investigating modifying the material's energy gap to allow writing at existing laser diode wavelengths. Finally, we are hoping to design materials which can be used in a read/write/erase system. We are currently exploring an exciting class of material which have shown repeated read/write/erase cyclability.

A major concern with the optical system is the ability to dynamically image through various layers of the material. As different planes are recorded in the cube, the magnification and amount of spherical aberration in the system changes. To dynamically compensate for this requires very expensive high performance imaging optics. One way to overcome this obstacle is to fabricate the recording material in a three-dimensional disk format. Using this 3D-CD approach, pages of images are stored as "spokes" in the rotating disk. The addressing beam is again cylindrically focused through the side, while the recorded page of data is imaged into the proper plane of the 3D-CD. Since the disk is constantly rotating, the conjugate planes remain in the same location and the magnification remains constant. In addition, with the use of a properly designed compensator in the input arm, spherical aberrations can be corrected.

A major design issue is the trade-off between data density and parallelism during the recording operation. Due to the parallel nature in imaging spatially modulated beams, whole arrays of data can be recorded at once. Assuming that a 1024 x 1024 SLM is used, it is possible to simultaneously write up to 1 Mb of data. However, as larger data pages are written, the longitudinal distance the addressing beam must remain collimated increases. Unfortunately, since the addressing beam diverges as it propagates, parallel recording results in the need for larger spacing between data planes, and thus, lower data densities. An alternative approach would be to write data in a bit-by-bit fashion allowing the density to be limited only by the Airy spot diameter of the two beams. While this drastically increases the data density, it eliminates the parallel nature of data recording. In Table I, three different data recording format are presented.

Data Format	**Pixel Size P_x x P_y µm x µm**	**Array Size N_x x N_y**	**Addressing Beam Dimensions**			**MVD**
			Width $W = P_xN_x$ µm	**Length $L = P_yN_y$ µm**	**Thickness µm**	**(bits/cm^3)**
Bit	1 x 1	1 x 1	1	1	1	$1\text{x}10^{12}$
Vector	5 x 5	256 x 1	1280	5	2	$20\text{x}10^9$
	5 x 5	1 x 256	5	1280	33	$1.2\text{x}10^9$
Image	5 x 5	16 x 16	80	80	8.2	$4.9\text{x}10^9$
	5 x 5	128 x 128	640	640	23	$1.7\text{x}10^9$
	10 x 10	1024 x1024	10240	10240	93	$0.1\text{x}10^9$

Table 1. The Maximum Volumetric Density (MVD) is shown for various data recording formats.

In the bit format bits are recorded serially. In the vector scheme, a vector (a column or row) of data is recorded in parallel, while with the image scheme two-dimensional data arrays are recorded at once. Computations are performed assuming Gaussian beam propagation of the addressing beam. The beam width and length are derived from the size of the imaged data array within the cube, while the thickness corresponds to the minimum plane-to-plane spacing of data pages which can be used. Finally, the maximum volumetric density (MVD) correspond to the maximum data density which can be stored within the cube. Note, that going from a bit-by-bit recording format to one in which 1024 x 1024 data arrays are stored in parallel increases the parallelism by 10^6 bits/recording step, but results in a density reduction of 10^4 bits/cm^3. It is important to point out that these calculations neglect aberrations in the input path's imaging system which becomes worse when larger arrays, resulting in larger field angles, are recorded.

2.3 Other Techniques In addition to our work with holographic and two-photon memory, we also have ongoing programs in other multi-dimensional optical storage systems. These approaches are all developmentally in a much earlier stage, but each offers significant advantages in terms of capacity, throughput or material fabrication. *Persistent Spectral Holeburning* (PSHB) takes a step ahead of one-bit-per-spot memories, allowing multiple bits to be written, erased and rewritten in a single location and offering the potential of storing up to 10^{15} bits/cm^3. The two techniques currently being explored are frequency-domain and time-domain PSHB.

Frequency-domain Holeburning involves burning "holes" in a material's absorption band [9]. The ideal material has many narrow, individual absorption lines that blend together into a broad absorption band. When writing information, a frequency-tunable laser would focus on a single spot, and scan down in wavelength while sending a stream of bits. At peak laser output (one pulse at a certain frequency), an absorbing center would make the transition from one stable state to another, so when the same spot is spectrally scanned there would be "holes" at certain frequencies which would indicate the presence of stored bits; this is one-wavelength (non-gated) holeburning. However, since non-gated holeburning tends to be volatile, a second, fixed wavelength is used (gated holeburning) to make the change permanent but still erasable.

Time-domain (also known as photon echo) holeburning also utilizes spectral holes for memory storage, but relies on coherent optical transient phenomena to store data[10]. Two temporally modulated beams are set to spectrally interfere, and that interference pattern is stored in the form of spectral holes. This concept is spectrally analogous to holograms, which are spatially constructed from the interference between the reference and modulated (object) beam. When a reference beam illuminates the spot for reading, the material emits a delayed coherent output signal that resembles the temporal waveform of the original data pulse.

There are a number of technical challenges to overcome before PSHB becomes viable for storage, however. One issue revolves around how to gate the material; current techniques inflexible or irreversible. Another problem is temperature; typically, practical operating temperatures have not exceeded 10 degrees Kelvin, making it difficult and costly to write and maintain data. One particular Rome Lab effort is an exception, achieving room temperature holeburning with a novel material, but even this requires considerable development before an acceptable storage density has been attained. These obstacles have slowed PSHB advancement, and have yet to be resolved before any practical applications can be produced.

Biological molecules would certainly be Darwin's media of choice for three dimensional recording. Serendipitous natural selection has given researchers a promising material in the protein *bacteriorhodopsin* (bR). This material is the light transducing protein found in the bacterium *Halobacterium halobium*. This light harvesting protein is found in the purple membrane of the bacterium. This membrane is essential, protecting from the harsh environment of salt marshes where the salt concentration is six times that of ordinary sea water. Bacteriorhodopsin provides the *H. Halobacterium* the ability to convert light energy to a metabolically useful form when conditions do not allow for aerobic respiration. This flexibility allows the bacterium to switch from photosynthesis to aerobic respiration depending on the environmental conditions. The conditions of the marsh dictate that the protein be resistant to thermal and photochemical damage. These qualities combined with its natural cyclicity (exceeding 10^6) of the protein make it an ideal media for optical recording[11].

The photocycle of bR is shown in Fig. 7. The main photocycle consists of the left-hand side of the figure[12]. In the course of studying this material, a branching reaction was identified. This is identified by the **P** and **Q** states. The resting state of the molecule, the bR state, can be elevated to the **K** state by the primary photochemical event. The other transitions are caused by thermal reactions and result once again in the resting state, bR. The entire photocycle takes about 6-10 milliseconds depending on the temperature. The interesting reaction which creates this branched photocycle happens when the last

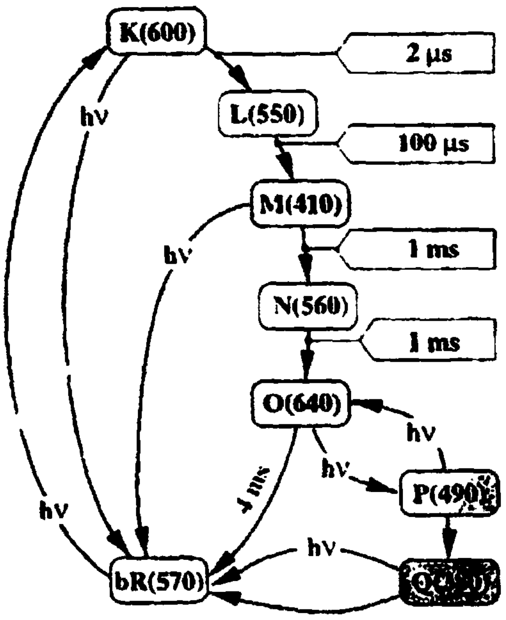


Figure 7. The photocycle of bR.

intermediate state, **O**, is converted by light to **P** which subsequently decays to **Q**. **P** and **Q** are the only states involved in the branched reaction and it exists as its own entity to the original photocycle. The **Q** state is the one used for recording data and is created by a sequential one-photon process. This implies that the timing of the reaction must be precisely controlled. The material must be illuminated a second time while the molecules are in the **O** state about 2 ms after the initial writing pulse The room temperature lifetime of the written state is roughly five years.

The architecture shown in Fig. 8 is used to record on thi, media. This setup is similar to the architecture of the two-photon approach discussed earlier, with the main difference being that data beam and the addressing beam not only must intersect at a given data plane in the material, but they must follow a timed sequence with the paging pulse activating the media followed by the data pulse 2 ms later changing the **O** state to the **P** state, as shown in Fig. 9. The entire memory can be bulk thermally erased, or selective page erasure is possible by illuminating with blue light (λ = 413 nm). This wavelength will convert both the **P** and **Q** states back to bR. Alternately, the entire memory can be bulk erased with incoherent light in the 360 - 450 nm range. Some investigators have suggested, however a WORM system as the amount of bacteriorhodopsin required for a memory capable of storing many hundreds of megabytes is on the order of milligrams and mass quantities of

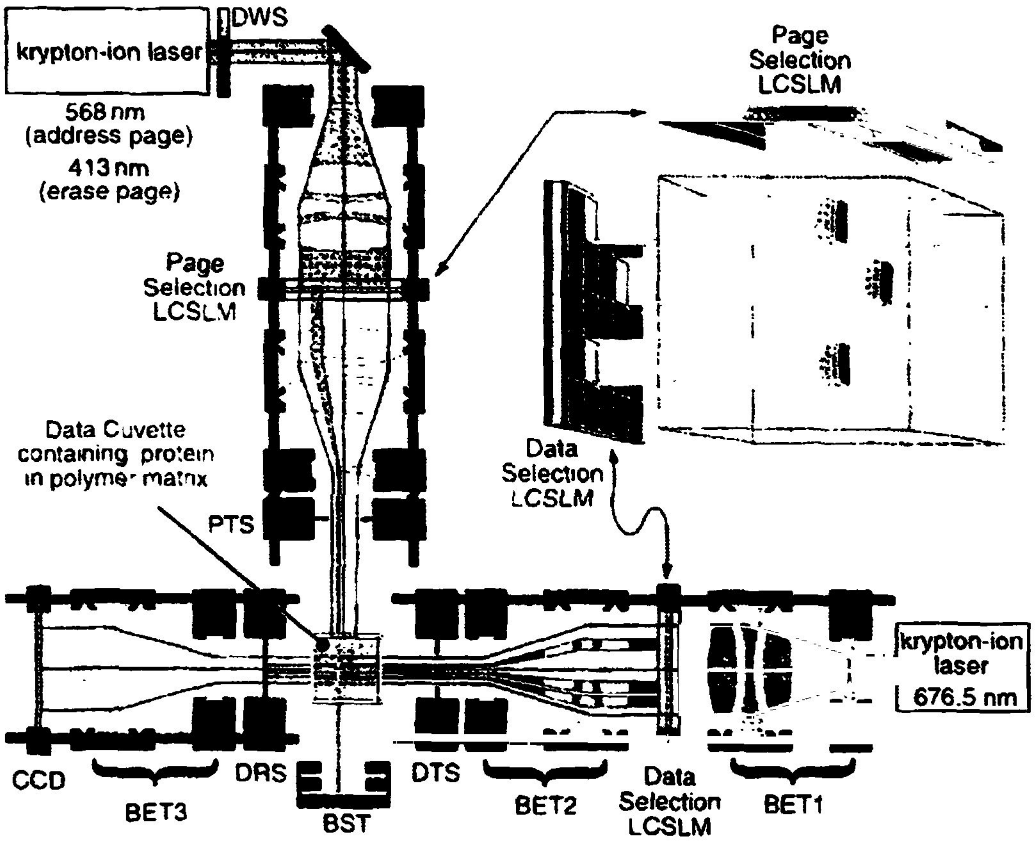


Figure 8. Schematic diagram of the branched-photocycle volumetric memory prototype base on bR.

this material can be made through fermentation processes. Ultimately, this may provide extremely cheap memory media. Individual bulk erasers could be provided for those wishing to re-use the media.

Volumetric 3-D optical recording is a relatively new process in the data storage world, but not nearly as new as *multi-wavelength DNA storage* - based on the use of enhanced

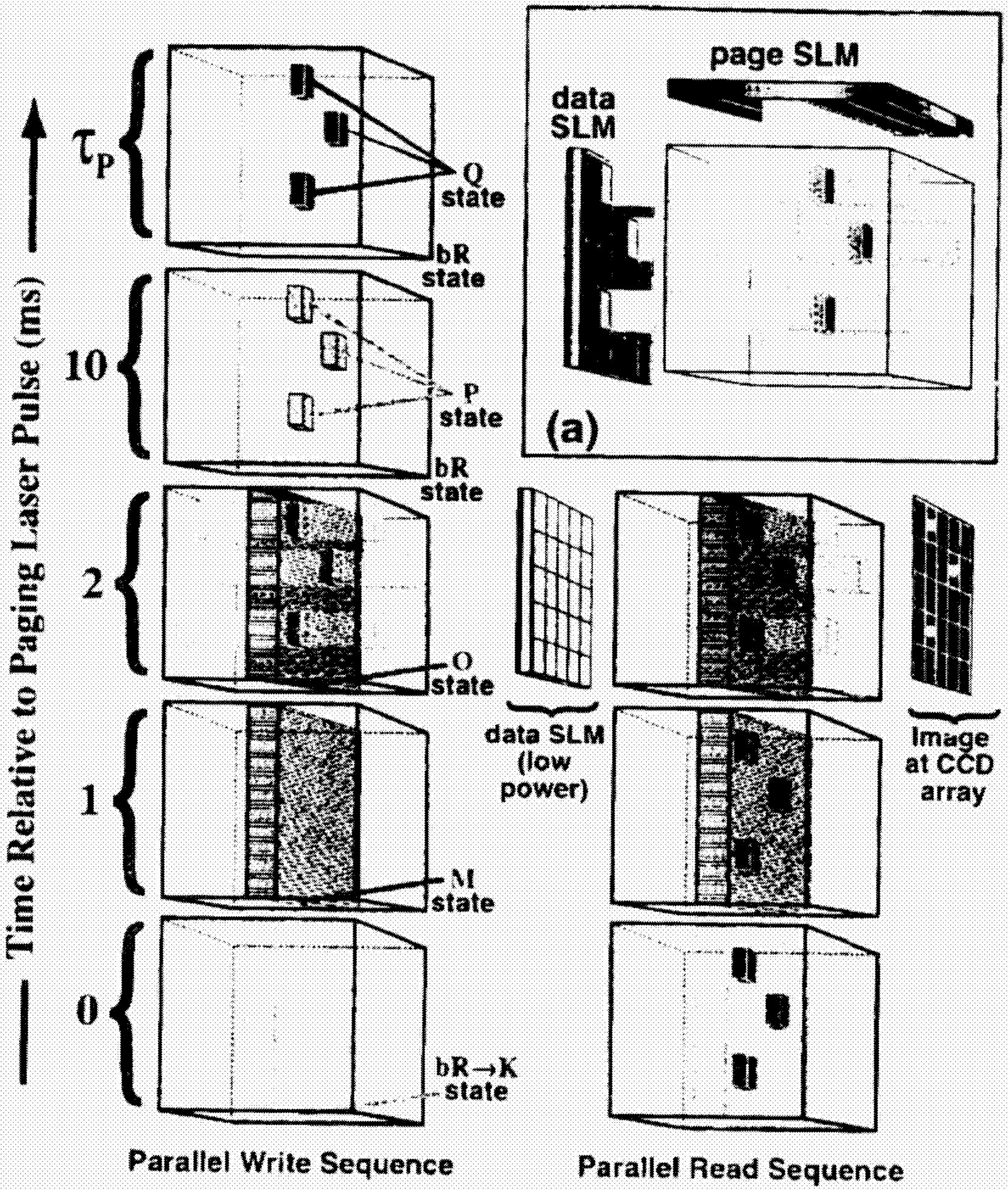


Figure 9. A schematic diagram of the parallel write and read timing sequences, with time progressing from top to bottom.

deoxyribonucleic acid (DNA) polymers as the three dimensional structure in prototype optical volumetric memories[13]. DNA polymers provide a robust and programmable synthetic material that can be organized into a wide variety of pre-determined structures. Because of the precise geometric and distance requirements of Forster energy transfer - which allows for the use of a single wavelength excitation wavelength and multiple wavelength emissions during the read process- energy is transferred at the molecular level, with almost 100% quantum efficiency, similar to photosynthesis in plants. Organized DNA chromophore structures, both natural and synthetic, produce relatively intense sub-micron fluorescent light sources. The intensity levels in the synthetic structures are within the range of commercially available detectors. The development of a customized optoelectronic read/write system is presently a primary research objective to make a practical DNA optical memory.

Working with Nanotronics, Inc, DNA polymers have been successfully attached to glass, aluminum, and silicon, to determine compatibility for use in a practical solid architecture in future systems. DNA chromophores exhibited energy transfer properties when attached in the solid state comparable to their solution state, and they exhibited quenching properties in the solid state dramatically superior to properties in a solution state. Experiments demonstrated the potential for multiwavelength optical storage material using DNA polymers - the early efforts demonstrated a 6 color response, and using this property, ongoing efforts show optical data storage using this organic material will allow very high data density - on the order of 100 bits/micron (100 times the storage of present storage densities). A conceptual DNA storage device is shown in Figure 10. As more devices shrink in size, synthetic DNA will be a major player in future technologies which require nanotechnology.

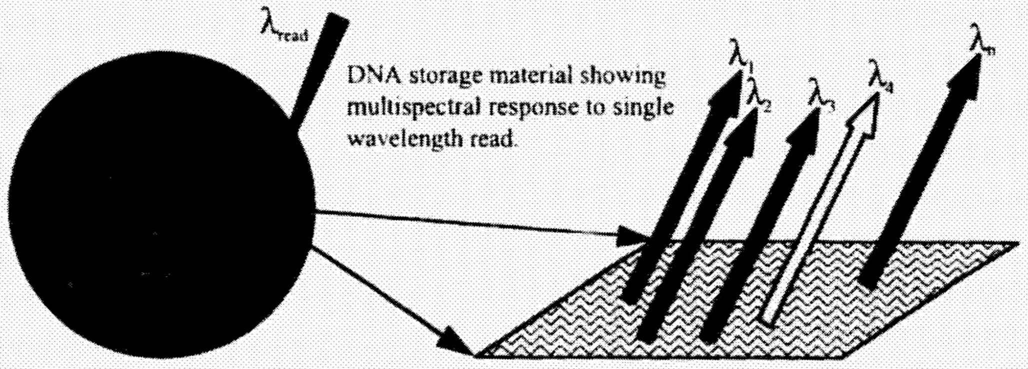

λ_{read}
DNA storage material showing multispectral response to single wavelength read.
λ_1
λ_2
λ_3
λ_4
λ_n

Figure 10. Visualization of a recorded spot in DNA material.

3.0 Summary and Conclusions

In summary, the technologies described here have potential in all three memory requirements, primary, secondary, and tertiary. Three-dimensional optical memory development can be applied to the secondary and tertiary memory requirements offering large data capacities, medium access time, and archivability. The 3-D optical memory concepts with terabit capacities, gigabit throughput rates, and 10-nanosecond access times are potential solutions to the prim-ry and secondary memory requirements.

With our two-photon and holographic work we have developed working demonstration systems, and anticipate operational systems within the next few years. Unfortunately, the prospects of a future system are still somewhat limited by key system components such as SLM's, fast (non-mechanical) beam deflectors, two-dimensional error correction and detection encoders/decoders and dynamic focusing/aberration control. However, these systems ultimately offer the advantage of extremely high data densities, and reasonable data I/O rates due to the parallel nature of data recording/readout. We are also pursuing some more elaborate techniques and materials such as spectral hole burning and DNA recording systems and bacteriorhodopsin as a recording medium. These techniques represent technologies which may be further from maturation, but represent significant progress in mass data storage technology.

References

1. Haritatos, Fred N., "High Performance Optical Disk Systems," SPIE/OSA Optical Data Storage Conference, Feb. 1991.

2. Haritatos, Fred N., "Optical Recording," Defense Electronics, vol. 25, no. 10, Sept. 1993.

3. Haritatos, Fred N., "High Performance Optical Disk Systems for Tactical Applications," IEEE, May 1991.

4. Burr, G., Mok, F., Psaltis, D., "Large-Scale Holographic Memory: Experimental Results," SPIE Proceedings, vol. 2026, p. 630-641, 1993.

5. Mok, F.H., "Angle-Multiplexed Storage of 5000 Holograms in Lithium Niobate," Optics Letters, vol. 18, no. 12, p. 915-917, June 1993.

6. Burr, F. Mok, D. Psaltis, Large-Scale Holographic Memory: Experimental Results, SPIE Proceedings, vol. 2026, 1993, p. 630-641.

7. Esener, S., Fainman, Y., Ford, J.E., Hunter, S., "Two-Photon Three-Dimensional Memory Hierarchy," (Photonics for Computers, Neural Networks and Memories, San Diego CA, 22-24 July 1992), Proceedings of the SPIE-The International Society for Optical Engineering 1993, 1773, p. 346-355, 1993.

8. Dvornikov, A.S., and Rentzedis, P.M., "Two-Photon 3-D Optical Memory," IEEE Proc. of 1994 Conference on Solid State Memory Tech., p. 109-118, 1994.

9. Maniloff, E., Altner, S., Bernet, S., Graf, F., Renn, A., Wild, U., "Spectral Hole Burning Holography in Optical Memory Systems," SPIE Proceedings, vol. 2026, p. 592-603, 1993.

10. Babbitt, W.R. and Mossberg, T.W., "Time-Domain Frequency-Selective Optical Data Storage in a Solid State Material," Optical Communications, vol. 65, p. 185, 1988.

11. Horspool, W., Song, P, *CRC Handbook of Organic Photochemistry and Photobiology*, CRC Press, Boca Raton, Chap. 33, 1995.

12. Birge, R., "Protein-Based Branched Photocycle Three-Dimensional Optical Memories," Rome Laboratory Final Report F30602-93-C-0135, 1996.

13. Heller, M., Tu, E., "DNA Optical Storage," U.S. Patent Application, Pending.

NEXT DOCUMENT

Durable High-Density Data Storage[1]

Bruce C. Lamartine
Roger A. Stutz
Los Alamos National Laboratory
Mail Stop B229
Los Alamos NM 87545
rstutz@lanl.gov
505-665-5615
505-665-3456 fax

Abstract

Information technology has completely changed our concept of record keeping. The advent of digital records was a momentous discovery, as significant as the invention of the printing press, because it allowed huge amounts of information to be stored in a very small space and be to examined quickly. However, digital documents are much more vulnerable to the passage of time than printed documents, because the media on which they are stored are easily affected by physical phenomena, such as magnetic fields, oxidation, material decay, and by various environmental factors that may erase the information. Even more important, digital information becomes obsolete, because, even if future generations may be able to read it, they may not necessarily be able to interpret it. This paper will discuss the Focus Ion Beam milling process, media life considerations, and methods of reading the micromilled data.

The Focus Ion Beam (FIB) micromilling process for data storage provides a new non-magnetic storage method for archiving large amounts of data. The process stores data on robust materials such as steel, silicon, and gold-coated silicon. The storage process was developed to provide a method to insure the long-term storage life of data. We estimate that the useful life of data written on silicon or gold-coated silicon to be on the order of a few thousand years without the need to rewrite the data every few years. The process uses an ion beam to carve material from the surface, much like stone cutters in ancient civilizations removed material from stone. The deeper the information is carved into the media, the longer the expected life of the information.

The process can record information in three formats: 1) binary at densities of 23Gbits/square inch, 2) alphanumeric at optical or non-optical density, and 3) graphical at optical and non-optical density. The formats can be mixed on the same media; and thus, it is possible to record, in a human-viewable format, instructions that can be read using an optical microscope. These instructions provide guidance on reading the remaining higher density information. The instructions could include information about the formats

[1] Approved for release LAUR# 96-2205

of the data, how to interpret the data bit-stream, and information on the types of readers or methods that can be used to recover the data.

There are several methods to read the information written with the ion beam. The selection the method is based on the density of the written data. Human-viewable data written at optical densities can be read with optical microscopes; binary data written at optical densities can be read much like currently CDs. Data written at non-optical densities can be read using force/tunneling microscopes or SEM readers. In any case the information read can be integrated with a computer.

Introduction

Information technology has completely changed our concept of record keeping. The advent of digital records was a momentous discovery, as significant as the invention of the printing press, because it allowed huge amounts of information to be stored in a very small space and be to examined quickly. However, digital documents are much more vulnerable to the passage of time than printed documents, because the media on which they are stored are easily affected by physical phenomena, such as magnetic fields, oxidation, material decay, and by various environmental factors that may erase the information. Even more important, digital information becomes obsolete, because, even if future generations may be able to read it, they may not necessarily be able to interpret it.

For data storage over hundreds to thousands of years, there is reasonable concern about effects of man-made or natural disasters. Fires and floods have destroyed many major data bases, for example, the great library at Alexandria, burned in about 642 AD. Exactly who was responsible is debated, but an irreplaceable storehouse of knowledge was almost totally destroyed. HD ROM (ion micromilling) technology discussed below would survive most such disasters. The melting point of stainless steel is approximately 2500 degrees F (1370 C) and can be used as the media for data storage with this process. Most building fires burn at about 1300 degrees F (700 C), thus the probability of data survival is quite high. It is noted that there are circumstances in which sustained fires can reach higher temperatures. Again, choice of materials for HD ROM storage can be designed to resist the most aggressive fires. However, simple placement of storage media in buildings that do not contain materials with high temperature combustibility would provide adequate protection. Of course, flooding is of little concern provided abrasion can be eliminated.

Comparisons with other technologies

With these concerns in mind, one can look for methods where information has been preserved for very long periods of time. A primary example is paper. Paper has been used for several thousands of years and has proved, on the whole, to be a reasonably stable media for the storage of information. Paper provides a means of storing information in a native language format that can be understood by large numbers of people. The printing press expanded the role and dependence on paper as a means for storing information. Nevertheless, paper has limitations that reduce its usefulness as a long term medium. The Europeans are now beginning to see deterioration in 500-1000 year old documents produced on low acid paper. Many documents produced within the last hundred years on common paper (high acid content) are so deteriorated that they are even hard to micro-film.

Limitations of paper:

- Fire
- Mold
- Environmental reactivity
- Slow information search and read rates
- Fading of inks
- Media life of about 1000 years depending on storage conditions

Advantages of paper:

- Native language
- Easy to copy
- Does not require special equipment to read or write

The next example, from recent times, is the use of microfilm as a storage media. Microfilm provides a native language capability. It also provides a method of reducing the volume required to hold the information. Copies are easy and cheap to make but still harder than making copies of paper documents. Microfilm is accepted in courts and generally as a replacement for paper documents. It is relatively hard to tamper with the information that is copied to microfilm. However, microfilm generally has the same disadvantages as paper with the addition that it requires an enlargement device to read the information.

Limitations of micro film:

- Fire
- Mold
- Environmental reactivity
- Slow information search and read rates
- Loss of resolution
- Requires an enlargement device to read
- Requires chemicals for processing the film
- Media life of about 50 - 500 years depending on storage conditions

Advantages of microfilm:

- Reduced space requirements
- Hard to modify information
- Accepted by legal system
- Native language

The development of magnetic storage was next in the chain that provided a means to store ever increasing amounts of information on compact media. Magnetic storage can be divided into two main groups - tape and disk. Both can provide high data densities in the order of 1 Gbit per square inch. Magnetic technologies are reaching the limits for storing information. Furthermore, large improvements in areal density are not anticipated because of the minimum size of magnetic domains. The main advantages to magnetic media are the ability of machines to quickly read and write information, to store large amount of information, to update and append information depending on formats, and to correct some errors with built-in error correction information. However, there are a number of disadvantages to the digital storage of documents over printed documents. First, magnetic tape and disk media that are used to store digital documents are easily affected by physical phenomena, such as magnetic fields, oxidation, material decay, and by various environmental factors that may erase the information. Even more important, digital information becomes obsolete because, even if future generations may be able to read it, they may not necessarily be able to interpret it. This is the result of requiring a bit stream interpreter to convert the information from a sequence of one and zero to numbers and text. Another concern for the archivist is that information stored on magnetic medium can be changed without leaving any indications of a change.

Limitations of magnetic storage:

- Fire
- Local RF and EMP fields
- Environmental reactivity
- Overwrite capability (advantage in some applications)
- Magnetic fade
- Requirement for bit stream interpreters
- Medium life of about 2-10 years depending on storage conditions
- Not native language

Advantages of magnetic storage:

- Rapid reading and writing
- Relativity high data densities
- Error correct capabilities

Optical storage systems are a recent addition to storage methods. They provide a non-magnetic method for storing information. They can support high data densities and can be divided into three general classes. The first are WORM devices that are write once - read many times. The second class is read only. The third class is erasable optical disks. All device classes provide a reasonable method to store information at relativity high data densities in digital formats. There are advances being made in this area and densities will continue to increase. Some classes of optical disks are accepted in some court systems but there is no uniform acceptance of digitally stored information. The main draw-back to these devices is still the medium. The medium is much like a current CD-ROM and is subject to many of the same limitations; for example, optical media will melt at relatively low temperatures.

Limitations of optical storage:

- Fire
- Environmental reactivity
- Requirement for bit stream interpreters
- Not native language
- Media life of about 40 years depending on storage conditions
- Slow data writing
- Not uniformly accepted by the courts

Advantages of optical storage:

- Relativity high data densities
- Relativity rapid reading
- Error correction capabilities
- Not affected by EMP or RF

None of the above methods have been able to match the process developed before the time of the pharaohs for long term storage of information. In those times men chiseled messages in stone as a means of creating enduring records. To be sure, these glyphs have imparted the information of their sculptors for readers millennia later. Some of the important factors that allow the information to be understood millennia later relate to the fact that the information is written in a native language. The Rosetta stone provided the key that allowed to translation of one native language to another. Greek and Latin writing can still be read without the need for a "rosetta stone" because they were written in a still active native language. The new method developed at Los Alamos National Laboratory uses an ion beam to chisel information into durable media. In fact, the durability of the this high-density technique is so great that one observer suggested that "long-term" should be replaced by "geologic," when describing the longevity of this data storage method. The method allows data to be written in native languages, direct human-viewable images, and in binary formats. The information types can be mixed on the same media. Therefore, it is possible to include in a human-viewable format, instruction on the bit-stream interpreter required to read the binary information.

Limitations of HD-ROM:

- Highest data densities require Scanning Electron Microscope for reading -- SEM are large devices
- Large size of writer

Advantages of HD-ROM:

- Very high data densities 23Gbits - 11,000 Gbits per square inch, higher densities are possible
- Media life of thousands of years
- Not effected by EMP or RF
- Not environmentally reactive depending on material used
- Several reading methods are available
- Native language formats are possible but not required
- Can have mixed densities on the same media
- Can have mixed data formats on the same media
- Rapid reading and writing
- Error correction capabilities
- WORM device (good for archiving information)

The HD-ROM serves two main functions: (1) it stores archival data for very long periods of time, and (2) it stores high-density data in binary, alphanumeric, and graphic formats. Refractive errors from thermal or mechanical shock are unimportant to HD-ROM. Additionally, it is resistant to reversals of magnetic fields that could affect the integrity of the data. This is contrary to the performance of current magnetic storage technologies. All present day data storage media rely on at least one soft, reactive, malleable, or flammable material for data integrity. However, HD- ROM materials are nonflammable, relatively unreactive, hard, and nonmalleable.

Writing Procedure

The high-density data storage is achieved by writing data with a micromill that employs a single focused ion beam. The micromill was built from existing parts, uniquely configured. The process allows writing at the nanoscale level with deep features, thus obtaining a very high data density. Data may be recorded in any vacuum-compatible material. Ion beams can produce high-aspect-ratio (the ratio between depth and width) features with channel widths as small as 75 atoms, or about 5 nanometers and aspect ratios approaching 45. Although these features are extremely small, they are still well under thermal stability limits (that is, the temperature above which atoms rearrange, a process that results in data loss). Data can be written at a larger scale that would further enhance the survival of the stored information.

The ion beam writing system used for the development of the process is composed of an ultrahigh vacuum system, a load lock, a secondary electron detector, and a liquid metal ion source column. Media are loaded into the load lock chamber and then pumped to a medium vacuum (5 E-7 Torr). The media is subsequently transferred to the ultrahigh vacuum chamber (7 E-11 Torr, or about one-ten trillionth of an atmosphere). The ion beam then is used to image the physical location of the medium by introducing sufficient secondary electrons to produce a contrast image similar to the more familiar scanning electron microscope (SEM). Subsequently, when operating the ion beam under higher current density, complex milling of digital, graphical or man-readable data is carried out by placing the beam position and dwell time under computer control. The level of control is similar to that available in typical computer aided manufacturing (CAM) software, and the operation in practice is similar to that of a waterjet mill.

The heart of the writer is the ion bean column. A liquid metal (typically gallium) is drawn to the tip of a source under high electric field and is then ionized. Shaping and focusing of the beam is accomplished with well known electrostatic (not magnetic) elements including the apertures, condensers, stigmators, and blanking elements. The resulting current density at the sample surface can be as high as 50 Amps/cm^2. Features such as channels and holes can be milled at aspect ratios approaching 45 at beam spot sizes near 0.5 microns. Alternatively data can be written at higher areal densities using smaller beam spots and lower currents. The minimum spot size achievable in our current system is about 500 angstroms. Using such a beam, channels as small as 770 angstroms have been reproducibly milled. A practical limit of milled features (channel or dot) size for data storage work appears to be about 5 nanometers (50 angstroms) for archival storage, depending on the materials used.

Since writing done with an ion beam can be controlled, very much like writing done with a dot matrix printer, multiple formats are also possible. Each character is represented by an array of points, each point characterized by a position and dwell time. This means that a feature can represent a binary value, a three-dimensional graphical image or an alphanumeric character. Moreover, different data formats and densities can coexist on the same physical medium.

Currently, the writing capability is limited to the speed of a single ion beam micromill. This allows writing at 276 Gigabyte/day using the higher current densities. To be effective at storing large data bases, advances must take place to allow simultaneous etching (writing) with multiple beams. This is seen as a mechanical issue and plans for a multi-head writer are progressing.

Reading Procedure

One of the unique features of the ion beam writing is that the data can be read in a number of different ways. Scanning electron microscopes provide the capability to read the highest data density while simple laser methods much like current CD-ROMs can be used for the lower density data. The readers are designed so the highest density reader can also read the lowest density information as well. This allows for data migration from low density reader systems to higher densities systems without the need to rewrite the data. Another unique feature is that the media can contain information written at human-viewable low density that describes how the higher density information can be read. The low density information could also contain instructions about any bit-stream interpreters required to make use of the higher density data. This would provide a means to insure that data written today could be read several thousand years from now even with new reader systems and even if the formatting and engineering information related to the media was lost.

There are three basic types of information that the readers must be able to read. The first is binary that is used for the storage of many styles of information such as numeric information, text, and bit map images. The second is one level image and text information in human readable form. The third is multi-level such as gray scale images and 3-D shapes.

Optical Systems

Optical systems fall into two general classes and three basic types of readers. The first class is for low density optical scale information much like current CD-ROMs. This density is a little better than current CD-ROMs and read rates of about 12x. The high density optical reader is designed to push the limits of optical reading efficiency. This density is much greater than current optical systems and read rates approaching 500 Mbytes per second are estimated.

Sub-optical

The sub-optical readers also fall into to two main classes. The first is a reading system designed to read sub-optical features without the use of a scanning electron microscope. These readers operate at much higher densities than the high density optical readers discussed above and have read rates in the order of 2 Gbytes per second. The high density sub-optical reader is a modified scanning electron microscope that can read features as small as 3 nm. The read rate for the scanning electron microscope is in the order of 2 to 20 Gbytes per second.

Other reading methods

In addition to the above method, data can be read with atomic force microscopes and STM. At present these methods are slower and more costly.

Applications

Jeff Rothenberg, in the January 1995 *Scientific American*[1], addressed the advantages of digital media for document storage and the need for a long-term solution for preservation of select records. HD-ROM offers such a solution. Of course, some argue there is little data worth storing for such a long period. Others suggest they would prefer a read-write system so that files can easily be updated. These arguments are valid. However, there are a substantial number of files for which read-only, (or read maybe) are appropriate. One example is need for "data assurance" where data must be safe from modifications. As another example, several institutions have an intense interest in maintaining genealogical data. The largest genealogical repository estimates their storage requirements at 12 pentabytes (10^{15}) of digital data. They also have an interest in the storage of genealogical data in a human-viewable form with greater longevity than micro-film. Since the actual storage cost is so low, an estimated $20.00 per terabyte for media materials, and the storage space so small, it would make sense for many industries and governmental agencies to take advantage of the technology.

As Rothenberg points out, magnetic media, the current choice for digital data storage, are vulnerable to "the ravages of time" through both material degradation and exposure to electromagnetic pulses(EMP). On the other hand, HD ROM is virtually impervious to EMP, and the degree of physical degradation can be controlled by choice of materials. For most circumstances, stainless steel should offer sufficient protection to ensure longevity. The major concern in long-term storage would be to ensure the potential for abrasion was minimized. Given the possibility for atmospheric contamination through increases in acidic content, materials should be protected nominally through prudent encapsulation. No extreme environmental measures such as cryogenic or high vacuum containers would be required.

Numerous recent articles have extolled the virtues of advances in magnetic data storage. Simonds, in *Physics Today*[2], April 1995, suggested recordings are the most significant market for magnetic technology. He states the business sector has mass storage requirements that amount to petabytes of digital data. In predicting advances in magnetic storage density, Simonds states, "10 Gbits/in^2 would be reached by the year 2005." He further predicts commercial densities of 5 gigabytes/in^2 and 5 terabytes/in^3 by 2003.

These predictions pale when considered against demonstrated current HD-ROM technology. Extant capability for data storage is 23 Gbits/in^2. Writing done on high-strength, 10 micron steel tape would produce storage greater than 50 terabytes/in^3 even allowing an air space packing factor. If 3 micron tape was used for data storage, then densities in excess of 190 terabytes/in^3 are possible. It is believed 3 micron, or thinner, tape would be strong enough for commercial applications. All of these figures are based on current capability. Further, the HD-ROM process does not currently entail any data compression. Many of the present and projected magnetic recording techniques employ data compression to achieve their storage densities. Should HD-ROM employ data compression techniques, then even higher densities than those described are possible.

A recent article by Terry Cook in *Technology Review*[3] discusses the need for better methods of storing information in the computer age. The paper also discusses some common problems associated with storage that do not exist to the human eye.

There are many applications for HD-ROM. Movies offer a substantive example. We are already aware of the substantial number of films that have been lost due to disintegration of materials. An industry is emerging to reconstitute film classics, but this is a time and labor intensive process. While nothing can be done short of this restoration process for degiaded films, something can be done to ensure that new films can have long life expectancies. With an estimated 250 Gigabyte/film data requirement for three color separation, each major studio could permanently record a film in less than two days using the multi-head writer.

The National Archives would also be a candidate user. There are many national records, including the Congressional Record, that should be stored for a long duration. Financial institutions would find long-term storage of incorruptible data a major advantage. Then, events such as the stock market information warfare attack depicted in Tom Clancy's recent book, *Debt of Honor*, would have minimum impact. It is the inability to change these records, making retroactive adjustments impossible, that would be of importance to that industry. In fact any business that values a sound data base would appreciate HD-ROM.

This also applies to the scientific community. Large, permanent data bases would be invaluable to researchers doing longitudinal studies. One example, would be NASA's Earth Observation System (EOS) data that is estimated will be collected at a rate of one Terabyte per day. NASA would like to keep the data for 90,000 years. Cross references to such data bases would assure the accuracy of base line data. Deep space probes too could benefit from HD-ROM. Detailed instructions could be etched on very small surfaces. These etchings would be impervious to the extremely harsh environments and unknown electromagnetic fields that might be encountered.

References

[1] Jeff Rothenberg, Ensuring the Longevity of Digital Documents, *Scientific American*, vol. 272 number 1, p 42-47,1995

[2] John L Simonds, Magnetoelectronics Today and Tomorrow, *Physics Today*, vol. 48 number 4, p 26-40,1995

[3] Terry Cook, It's 10 o'clock: Do you know where your data are?, Technology Review, vol. 98 number 1, pages 48-53

NEXT
DOCUMENT

A Note on Interfacing Object Warehouses and Mass Storage Systems for Data Mining Applications[*]

Robert L. Grossman

Magnify, Inc.
815 Garfield Street
Oak Park, IL 60304
Email: rlg@opr.com
Tel: 708-383-7002
Fax: 708-383-7084

University of Illinois at Chicago
Laboratory for Advanced Computing
851 South Morgan Street
Chicago, IL 60607
Email: grossman@uic.edu
Tel: 312-413-2176
Fax: 312-996-1491

Dave Northcutt
Magnify, Inc.
815 Garfield Street
Oak Park, IL 60304
Tel: 708-383-7002
Fax: 708-383-7084

Abstract

Data mining is the automatic discovery of patterns, associations, and anomalies in data sets. Data mining requires numerically and statistically intensive queries. Our assumption is that data mining requires a specialized data management infrastructure to support the aforementioned intensive queries, but because of the sizes of the data involved, this infrastructure is layered over a hierarchical storage system. In this paper, we discuss the architecture of a system which is layered for modularity, but exploits specialized lightweight services to maintain efficiency. Rather than use a full functioned database for example, we use light weight object services specialized for data mining. We propose using information repositories between layers so that components on either side of the layer can access information in the repositories to assist in making decisions about data layout, the caching and migration of data, the scheduling of queries, and related matters.

Introduction

Data mining is the automatic discovery of patterns, associations, and anomalies in data sets. The data mining of large data sets is a special challenge because the process requires numerically and statistically intensive queries on large amounts of data. Our assumption is that data mining requires a specialized data management infrastructure, but because of the sizes of the data involved, this infrastructure is layered over a hierarchical storage system. Our concern in this paper is an appropriate open, layered architecture to support this.

[*] This work was supported in part by the Massive Digital Data Systems Program.

A common layered architecture for this type of system is illustrated in Figure 1. There are three layers: the storage management layer, the data management layer, and the data mining and analysis layer. Unless these three layers coordinate how the data is physically laid out, how it is cached and migrated, and how it is prefetched, these layers can work at cross purposes and drastically impair the performance of the overall system.

The traditional approach forgoes the convenience and modularity of a layered approach for efficiency: with this approach, the data management system manages storage itself, while the data mining and data analysis applications manage the data themselves. In practice, this has meant that generally data mining applications simply work with flat data that fits into main memory. Of course, this data may be obtained by sampling large databases, but the point is that the data mining applications *themselves* work with small amounts of relatively simple data. This may be thought of as a sample-based approach to data mining.

In this paper, we are concerned with an alternative approach: the system is layered for modularity, but exploits specialized lightweight services to maintain efficiency. Rather than use a full functioned database for example, we use light weight object services specialized for data mining. With this approach, the data mining applications can work with large amounts of complex data. Another advantage of this approach is that the data management services can be used to manage the internal data structures required by the data mining algorithms. This may be thought of as a data-driven approach to data mining.

One of our specific concerns in this note is how the different layers can share information, especially in a heterogeneous environment. We propose using information repositories between layers so that components on either side of the layer can access information in the repositories to assist in making decisions about data layout, the caching and migration of data, the scheduling of queries, and related matters.

This proposal generalizes and extends the proposal in Brown et. al. [1] for providing a repository between a mass storage system and a relational database management system and is a refinement of the architecture described in Grossman [2] and [3] for a scaleable data mining system.

This work is preliminary. A fuller treatment is in preparation.

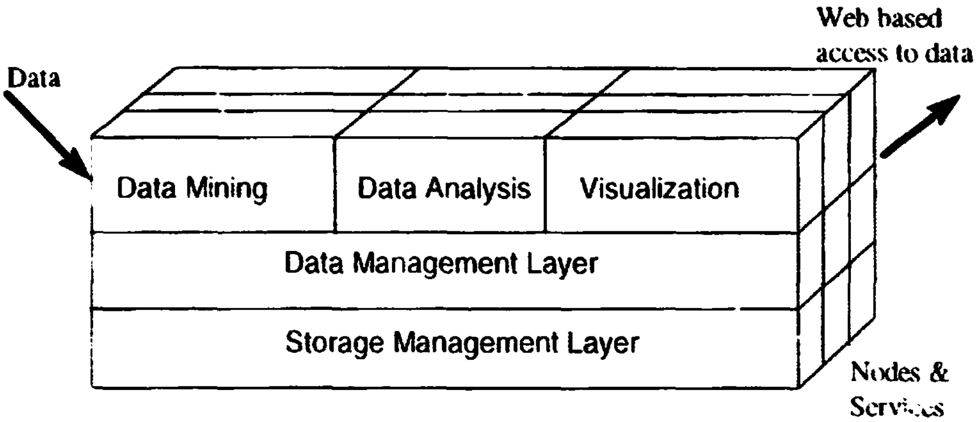


Figure 1. In a layered approach to data mining, rather than manage their own data, data mining applications use services from a data management layer, which in turn use storage services from a lower layer.

Background and Related Work

Broadly speaking, there are two relevant traditions: one system-based and one service-based. In the first, the essential question is how a database management system can interface to a storage management system. In the second, the essential question is what services are required so that data management, storage management, and application services can interoperate in an open network environment.

Relational database-mass storage system interfaces

Historically, databases have managed the storage of single disks; more recently, they have managed the storage of distributed disks. For some applications though much of the data is distributed on a storage hierarchy, including tape and other tertiary storage, which is m naged by a mass storage system. One of the most important interfaces effecting performance is the interface between a relational database client and the mass storage system. A group at Lawrence Livermore National Laboratory has proposed an interface between a client of a relational database management system and a mass storage system Brown et. al. [1]. This interface which they call an Information Data Repository (IDR) would serve as the home fo several relational tables, including: one for relational tables from the client database (called the bundle table), one for instances of the various components in the storage hierarchy (called the store table), one for mapping regular sub-components of bundles to stores (called the block table), and one for a list of pending requests for moving data between stores (called the movement table). In addition, the proposal [1] suggests using a standard relational database management system to manage the various tables in the IDR. The IDR would be external to both the database and the storage system and all interactions between the database and the mass storage system would be required to go through the IDR.

Light Weight Object Management Using Network Services

Another approach is to develop a data management system specifically designed for the mining and analysis of data. This type of system does not require the full functionality of a database, but instead is optimized to provide low overhead, high performance access to data which is read often, occasionally appended, but infrequently updated. In addition, data may be pre-computed and specialized indices may be provided. This can be thought of as providing specialized *lightweight* application specific data management services Grossman et. al. [4]; or alternatively, as providing an object warehouse specialized for data mining applications Grossman [3].

As usual with databases, with this approach there is a manager for physical collections of objects (called segments). In addition, to achieve scalability, physical collections of segments are themselves gathered into larger physical units called folios. There is also a folio manager which interacts with file and storage services, including mass storage systems. Just as the segment manager can query the folio manager, so can the mass storage system. The folio manager maintains a table of folios and their physical locations. In some sense, the folio manager can be viewed as the interface between a database and a (hierarchical) storage system. See [3] and [4] for more information about this approach.

Distributed Object Services

The Object Management Group's Common Object Request Broker Architecture (CORBA) is an industry standard for the development of distributed object-oriented applications across heterogeneous platforms. The OASIS environment developed at UCLA by Mesrobian et. al. [5] is an open environment for working with scientific information based upon CORBA. CORBA is optimized for working with relatively large-grained objects in heterogeneous environments in contrast to the use of lightweight data management and data warehousing described above. In some sense, CORBA is pessimistic about the physical layout of data and provides the infrastructure to support this in order to work in heterogeneous environments, while a lightweight approach is optimistic and only translates the physical format of data when necessary.

Requirements and Objectives

Our over all objective was to design an open system for data mining and data analysis which scales as the amount of data and the numerical complexity of the query increases. More specifically, we had the following requirements:

- *Large data sets.* Our most important goal was to support the mining and analysis of very large data sets, including data sets large enough to require multiple disks or tertiary storage.

- *Numerically intensive queries.* Our second most important goal was to provide very low overhead, high performance access to the data. In some sense databases are optimized to provide safe access to data which is expected to change; our goal was to provide high performance access to data which is relatively static.

- *Transparent access to data.* Because of the size of the data sets, much of the data is expected to be either on tertiary storage or on large arrays of disks. An important goal was to provide transparent access to the data, independent of its location or media type.

Architectural Description

Our architectural framework consists of a storage management layer, a data management layer, and an application layer consisting of clients of the data management services. We are primarily concerned with data mining and data analysis clients. Between each of the layers is a repository for information: a Storage Interface Repository (SIR) between the storage management and data management layers and a Data Interface Repository (DIR) between the data mining applications and the data management layer.

Data Interface Repository (DIR)

Traditionally, data mining has looked for patterns in small amounts of flat file-based data or sampled small amounts of data from relational databases using SQL queries. Data-driven data mining requires working with large amounts of complex data, much of which has to be warehoused because of performance considerations. The DIR has several roles, including:

- The data required for data mining and data analysis queries may be distributed in several data management systems, including data warehouses and operational and archival data management systems. The DIR provides a uniform interface for data mining and data analysis queries. The DIR maintains a list of logical data sets and the systems which are maintaining them.

- For performance reasons, some of the data for data mining applications may be warehoused, and specialized index and access structures may be provided. This requires periodically refreshing the data from the operational and archival databases. The DIR maintains the information required for this to take place.

- The DIR can also maintain the information for the optimization of data mining queries using information obtained from the results of previous queries.

Storage Interface Repository (SIR)

Data management systems by necessity divide the data they manage into regular sized extents. For example, access to file-based data is through blocks of equal length, while a common type of object-oriented database provides access to objects through extents of equal length called segments. These extents can then be managed by the data management systems themselves or by file or storage systems. In particular, they may be managed by hierarchical storage systems. The SIR has several roles, including:

- The demands upon extents imposed by the database management system are not necessarily those imposed by the hierarchical storage system. Not all extents are treated the same by the data management system: for example, some may contain directory or index information, which it would prefer remain pinned to secondary storage, even if infrequently accessed. The SIR provides a mechanism for a database and a hierarchical storage system to exchange information about desired movements of extents or sequences of extents.

- A database must be able to estimate the time to access data. If the physical management of the data is delegated to the hierarchical storage system, then the SIR must contain enough information so that the database can still make these estimates.

- To work with very large data sets, a hierarchy of extents, as described above, must be supported by the SIR. For example, for terabyte size data sets, there are simply too many segments to be managed directly by the database. Instead, it is important to group objects into segments, and segments into larger units.

The SIR discussed here is an extension of the IDR proposed in Brown et. al. [1].

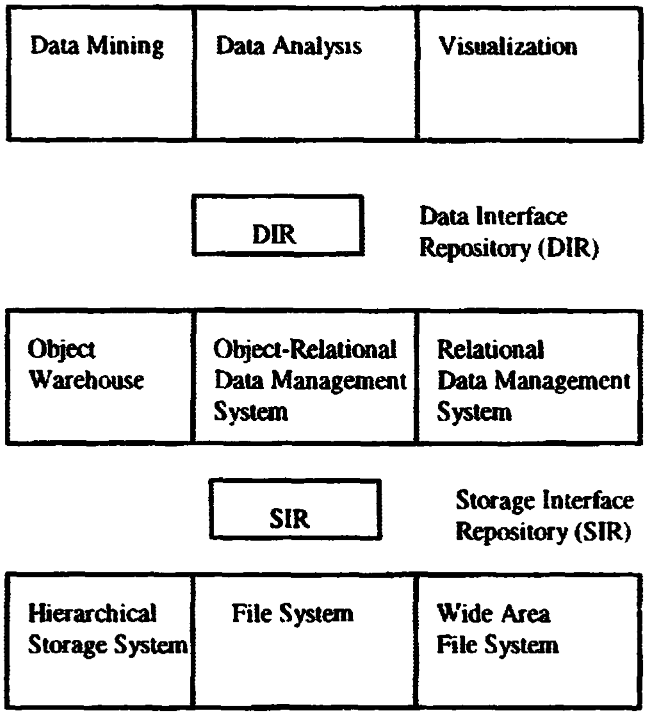

Data Mining
Data Analysis
Visualization
DIR
Data Interface Repository (DIR)
Object Warehouse
Object-Relational Data Management System
Relational Data Management System
SIR
Storage Interface Repository (SIR)
Hierarchical Storage System
File System
Wide Area File System

Figure 2. The role of the Data Interface Repository (DIR) and the Storage Interface Repository (SIR) is to maintain information so that services and applications in different layers can interoperate.

Discussion

In this section, we discuss some issues regarding the architecture.

- *Is the interface mandatory or advisory?* Systems can be built either way. If the interface is mandatory, then performance may suffer, since some of a component's essential services may have to be accessed externally. On the other hand, if the service is advisory, inefficiencies are likely and deadlocks are possible because different components accessing the service may make conflicting choices.

- *Is the interface part of one of the components or independent?* Traditionally, for example, the management of table information, block information, and the mapping from tables to blocks has been a component of the data management system. The role of the SIR is to provide this information through a separate service. Alternatively, the SIR could be incorporated into one of the layers and accessed from the other layer.

- *How should the DIR and SIR be implemented?* A variety of implementations are possible: The DIR and SIR could simply be implemented as a network service. Alternatively, a relational database can be used as proposed in Brown et. al. [1], or a CORBA Object Request Broker (ORB) could be used.

- *What is the granularity of access?* For this approach to succeed, it is important to be able to adjust the granularity of the objects referenced in the SIR and DIR so that performance is not adversely effected.

Status

This approach arose out of work with a system for data mining developed by Magnify, Inc. called PATTERN. PATTERN currently consists of beta versions of an object warehouse [3] and data mining modules for classification, prediction, and optimization Grossman et. al. [6]. A demonstration of the system mining and analyzing high energy physics data took place at Supercomputing 95. A performance evaluation of the system is currently being prepared and will appear elsewhere.

Currently, the SIR is part of the object warehouse and interfaces to the High Performance Storage System (HPPS) Teaff [7], while the functionality proposed by the DIR is currently shared between the different data mining modules.

Summary

In this paper, we propose a layered approach to a data mining system. Data mining applications exploit specialized data management services from a lower level, which in turn exploit specialized storage management services. We propose providing information repositories between each level so that services on either side can efficiently exchange information. To maintain performance, we use specialized lightweight data management services instead of a full functioned database, and adjust the granularity of the data passed between the layers to lower the cost of accessing the information repositories.

References

[1] P. Brown, D. Fisher, S. Louis, J. R. McGraw, R. Musick and R. Troy, "The Design of a DBMS/MSS Interface," Lawrence Livermore National Laboratory Technical Report, 1995.

[2] R. L. Grosman, H. Hulen, X. Qin, T. Tyler, W. Xu, "An Architecture for a Scalable, High Performance Digital Library," Proceedings of the 14th IEEE Computer Society Mass Storage Systems Symposium, S. Coleman, editor, IEEE, Los Alamites, CA, 1995, pages 89-98.

[3] R. L. Grossman, "Early Experience with a System for Mining, Estimating, and Optimizing Large Collections of Objects Managed Using an Object Warehouse ,"

Proceedings of the Workshop on Research Issues on Data Mining and Knowledge Discovery, Montreal, Canada, June 2, 1996.

[4] R. L. Grossman, S. Bailey, and D. Hanley, "Data Mining Using Light Weight Object Management in Clus'ered Computing Environments," Proceedings of the Seventh International Workshop on Persistent Object Systems, Morgan-Kauffmann, 1996.

[5] E. Mesrobian, R. Muntz, E. Shek, S. Nittel, M. LaRouche, and M. Krieger, "OASIS: An Open Architecture Scientific Information System," 6th International Workshop on Research Issues in Data Engineering, New Orleans, La. February, 1996.

[6] R. L. Grossman and H. V. Poor, "Optimization Driven Data Mining and Credit Scoring, Proceedings of the IEEE/IAFE 1996 Conference on Computational Intelligence for Financial Engineering (CIFEr), IEEE, Piscataway, 1996, pages 104-110.

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

NEXT
DOCUMENT

Towards the Interoperability of Web, Database, and Mass Storage Technologies for Petabyte Archives

Reagan Moore, Richard Marciano, Michael Wan,
Tom Sherwin, Richard Frost
San Diego Supercomputer Center
P.O. Box 85608
San Diego, CA 92186-9784
E-mail: {moore, marciano, mwan, sherwint, frost}@sdsc.edu
Phone: 619-534-5073
Fax: 619-534-5152

Abstract

At the San Diego Supercomputer Center, a Massive Data Analysis System (MDAS) is being developed to support data-intensive applications that manipulate terabyte-sized data sets. The objective is to support scientific application access to data whether it is located at a Web site, stored as an object in a database, and/or stored in an archival storage system. We are developing a suite of demonstration programs which illustrate how Web, database (DBMS), and archival storage (Mass Storage) technologies can be integrated. An Application Presentation Interface is being designed that integrates data access to all of these sources.

We have developed a data movement interface between the Illustra object-relational database and the NSL UniTree archival storage system running in production mode at the San Diego Supercomputer Center. With this interface, an Illustra client can transparently access data on UniTree under the control of the Illustra DBMS server. The current implementation is based on the creation of a new DBMS storage manager class, and a set of library functions that allow the manipulation and migration of data stored as Illustra *"large objects"*.

We have extended this interface to allow a Web client application to control data movement between its local disk, the Web server, the DBMS Illustra server, and the UniTree Mass Storage environment. This paper describes some of the current approaches for Web, DBMS, and Mass Storage interoperability, and presents a framework for successfully integrating these technologies. This framework is measured against a representative sample of environmental data extracted from the San Diego Bay Environmental Data Repository. Practical lessons are drawn and critical research areas are highlighted.

1. Introduction

A series of projects are being undertaken at the San Diego Supercomputer Center to develop the software technology that is needed to support data-intensive scientific applications (Moore [1]). These projects explore various aspects of distributed data handling capabilities, including integration of object-relational database management systems (ORDBMS) (Moore [2]) with archival storage, development of Web and Java interfaces for databases and archival storage systems, and development of a standard API for accessing data from heterogeneous sources.

The ability to manipulate very large data sets and large collections of data sets is a chief goal of the Massive Data Analysis System (MDAS) project (Moore [3]). Two features are essential components of this system: accessing data sets by attribute rather than UNIX file name, and transporting very large data sets across parallel I/O channels. Object-relational database technology is used to support query by attribute, and archival storage technology is used to support third-party parallel transfer of data sets. The MDAS project is integrating these technologies to create a data handling environment capable of supporting terabyte-sized data sets. *Large objects* that are controlled by the database are stored in the archive instead of the database local disk. This allows the ORDBMS to manage collections of data objects which exceed the local database disk capacity. By using transportable methods for manipulating data objects, it is also possible to minimize CPU execution constraints. When a query is processed, both the data object and the transportable method are sent to a system on which the analysis is then done. The data handling system effectively serves as a data scheduler, moving data and associated computational methods to available compute resources.

The data-handling system architecture is presented in Figure 1. Three different clients are shown accessing the system, corresponding to interactive Web-based access, scientific application access, and DBMS access to support data movement between multiple data handling systems. The DBMS maintains large objects within the archive and has the ability to schedule computationally intensive work on various production systems. The system is designed to support third-party transfer of data from the archive directly to the requesting system across parallel I/O channels.

To gain insight into issues associated with database/mass-storage integration, we built a prototype using Illustra (Illustra [4]) as the database engine and NSL UniTree as the mass storage system. NSL UniTree is a hierarchical archival storage system currently running in production mode at SDSC. The hardware platform consists of a single IBM RS/6000 99J workstation, a disk cache of 100 GB, two StorageTek tape silos and 8 tape drives transferring data at rates up to 2.9 MB/s with 1 controller per four drives. The system is capable of storing up to 20 TB (terabytes) of data. The most recently accessed files are staged on the large disk cache and the rest are migrated to tape.

A second research prototype has been created through a similar integration of Postgres95 Stonebraker [5, 6]) with NSL UniTree. The database runs on a 17-node IBM SP-2, which controls a 500-GB IBM Serial Storage Architecture (SSA) Disk Subsystem and a 60-TB IBM 3494 Tape Library Dataserver using six high-capacity 3590 Magstar tape drives. This system will be used in collaboration with IBM to develop a Massive Analysis Testbed that integrates the DB2 Parallel Edition DBMS with the High Performance

Storage System (HPSS) mass storage system (Archival Storage Research at SDSC [7]). Data transfer rates of 300 MB/sec are expected from this system. The nominal design point for expansion of this testbed is to sustain at least 1 GB/sec data access rate for each additional terabyte of disk. The design point for data access to tape is 1 GB/sec per 100 terabytes storage capacity.

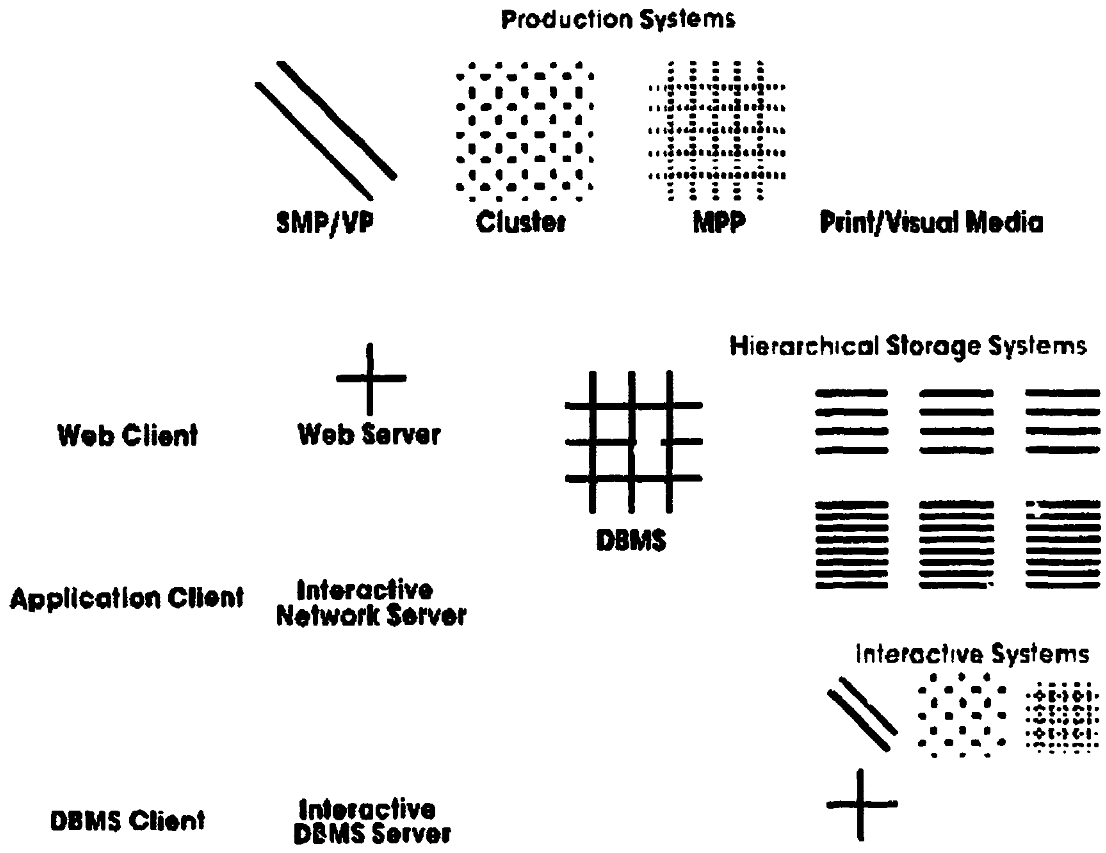


Figure 1: MDAS System Architecture

This paper presents the various software interfaces that have been developed in the research prototypes. The mass storage interface is described in section 2, the database interface in section 3, and a Web interface in section 4. A real-world example consisting of a representative sample of environmental data extracted from the San Diego Bay Environmental Data Repository is shown in section 5, and concluding remarks and expanded data access scenarios are given in section 6.

2. Mass Storage interface (MSI)

We have developed a data movement interface between the Illustra object-relational database and the NSL UniTree archival storage system. With this interface, an Illustra client is able to transparently access data on UniTree through the Illustra server by sending appropriate queries and commands.

2.1 MSI software features

Metadata describing the data set attributes are stored on the local disk under the database control. *Large objects* (data items larger than approximately 8K bytes) are stored in UniTree through the Illustra/UniTree interface. A *large object* is a defined data type that is created using the Illustra DBMS facilities. *Large objects* stored in UniTree have all the database properties of any Illustra object, such as transaction rollback, crash recovery, and multi-user protection. Unreferenced *large objects* can be removed from the database by issuing the *"vacuum"* SQL statement. However, once created they cannot be overwritten or appended to. Illustra supports a built-in data type for pointing to a large object, called *"large_object"*. When a user selects a *large object* from a table, the returned value is a handle to the *large object*. The handle is a character string, such as *'109872398721l'*, which is used to define a unique data set within the UniTree system or the local database disk.

From an Illustra client standpoint, except for the difference in access speed between local disk and remote archive, *"large objects"* stored in UniTree behave exactly the same as other *"large objects"*. A user can use normal queries and commands to perform the following tasks:

- Store and retrieve *large objects* between local disk and UniTree.

- Vacuum unreferenced *large objects* stored in UniTree.

- *"Dump"*, *"restore"* and *"recover"* *"large objects"* stored in UniTree.

To test the integration, *"large object"* files stored in UniTree were intentionally deleted after a *"dump"*. *"Restore"* and *"Recover"* were then used to restore the deleted files.

2.2 MSI software implementation

The implementation of the MSI is done by adding a storage type - "UniTree" to the ORDBMS storage manager. This required creating a set of 35 new UniTree specific access functions for operating on data sets. Example functions are *open*, *close*, *read*, *write*, *flush*, *abort*, and *synch*. The design provides a one-to-one counterpart for each UniTree access function with the corresponding function for accessing magnetic disk storage.

Similar to the magnetic disk storage type functions, the UniTree access functions do not make direct I/O calls. Instead they perform I/O through Virtual File Descriptor functions that call the *libnsl.a* and *libnsltree.a* UniTree libraries to interact with the UniTree Mass Storage System. These libraries provide client processes with UNIX-like I/O access functions as well as functions that are specific to UniTree such as file staging and migration.

3. Database software interface

User functions have been developed to allow user-level control over the storage location of the data sets within the integrated database/archival storage system. Note that the data sets might initially be stored on the user's local disk, then stored as a large object on the database system disks, or stored in the archival storage system. The responsiveness of the system typically improves, the closer the cache level is to the user. Hence user control is needed to optimize access performance.

Three *DataBlade* functions - **myFileToLO(), LocalToUtree()** and **UtreeToLocal()** have been created to provide an easier way for an Illustra user to convert local files to large objects on UniTree and to migrate objects between UniTree and database file systems. A *DataBlade* is a mechanism to extend the Illustra server to manage new data types and functions on these data types.

The *DataBlade* terminology comes from the following analogy: just like a general purpose utility knife can be extended to perform different cutting jobs by inserting special-purpose blades, so can the Illustra Server be extended to manage new data types by snapping in the required *DataBlade*. Basically, these functions use the *large_object* manipulation functions of Illustra to move *large objects* between database magnetic disk and UniTree.

3.1 Data caching functions

1) myFileToLO (filename, flags, smgr) - used to copy a local disk file to a *large object* stored in the archival storage system. This is the same as the *FileToLO ()* function that comes with Illustra with the exception that a parameter - *smgr* has been added to allow users to specify the storage type for the *large object*.

Filename = The name of the file to be converted to large object.

Flags = the location of the file :
0 = the file is on the client machine.
1 = the file is on the server machine.

Smgr = the storage type where you want to store the large object.
0 = local disk.
2 = UniTree.

The *returned value* is the LO handle of the newly created object.

2) UtreeToLocal(large_object) - used to migrate large objects from archival storage to the database disk.

Large_object = The LO handle of the large object to be migrated.

The *returned value* is the LO handle of the newly created large object.

3) LocalToUtree(large_object) - It is used to migrate large objects from the database disk to archival storage.

Large_object - the LO handle of the large object to be migrated.

The *returned value* is the LO handle of the newly created large object.

3.2 Examples

The following script illustrates the use of these three *DataBlade* functions. One could interactively enter this script using the *msql* command shell. The text bold corresponds to the system's response. The *large object* handle value encodes the cache location of the data set (I2... means that the *large object* actually resides in the UniTree archival storage system, and I0... means that it is on the Illustra Server disk). Two data sets are stored in the system; "foo1" on UniTree and "foo2" on database disk. "foo1" is then migrated onto database disk, and "foo2" is migrated into UniTree.

```
---- First, create a table named LOTest.
create table LOTest
(
        name text,
        myLO large_object
);

---- The following command will store the large object in Unitree
insert into LOTest values ('foo1',
                 myFileToLO ('file1', 0, 2));
one row inserted

---- The following command will store the large object to local
---- disks.
insert into LOTest values ('foo2',
                 myFileToLO ('file2', 0, 0));
one row inserted

select * from LOTest;
---------------------------
|name        |myLO          |
---- ----------------------
```

```
|foo1         |I2105826192499|
|foo2         |I0108798611396|
----------------------------

----
---- The next 2 commands migrate the large objects between the
---- local disk and UniTree.
----
update LOTest set myLO=UtreeToLocal(myLO) where  name='foo1';
update LOTest set myLO=LocalToUtree(myLO) where  name='foo2';

select * from LOTest;
----------------------------
|name         |myLO          |
----------------------------
|foo1         |I0109780388206|
|foo2         |I2100786116526|
----------------------------

---- Illustra does not delete the old object automatically, so
---- you need to vacuum it.
vacuum from LOTest;
```

4. Web software interface

A Web Server side C-language CGI (Common Gateway Interface) to Illustra was developed. This program allows the user to build or specify existing SQL queries which are then passed to the Illustra server. In essence the C interface program is a multi-purpose program acting as a Web Client program (generating HTML) and also as an Illustra DBMS client program (connecting remotely to the Illustra Server, issuing SQL commands, collecting SQL command result sets, disconnecting from Illustra, extracting information from the result sets and displaying it to the screen). The DBMS client part is done by linking the code to the *libmi.a* Illustra C-programming interface library.

Other than the fact that the SQL commands which are sent to the server allow the use of the new UniTree DataBlade functions, the Web software interface is a standard interface that one would find in most Web to DBMS integrations.

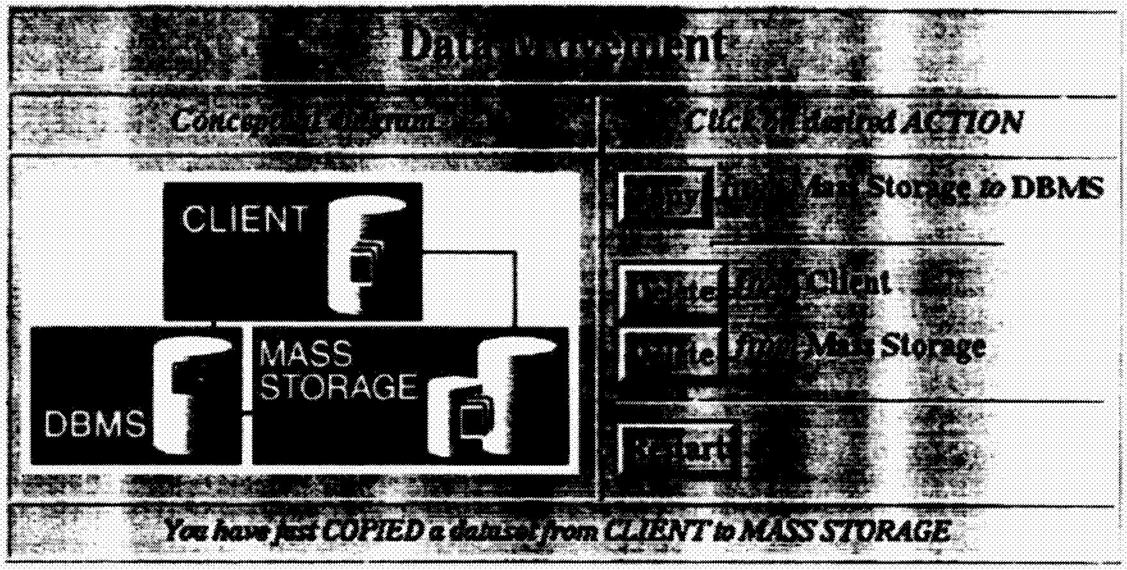
This section illustrates how one might be able to control this integrated environment on a simple web example (section 4.1) and concludes with general considerations on how we dealt with server time-out issues (section 4.2) .

4.1 An integrated example

The WFB demo presented here can be executed from the following Web page:

http://www.sdsc.edu/projects/MassDataAnal/Demo_Illustra+Unitree/

The following explanations are meant to serve as an introductory guide to this web demo.



You have just COPIED a dataset from CLIENT to MASS STORAGE

Figure 2: Data Movement Integration demonstration screen

The demo presents the equivalent of a finite state machine. The three states are: Client, DBMS, and Mass Storage. The demo illustrates data movement between a Client (Web client, for example), a DBMS and a Mass Storage environment. Tokens representing data objects are allowed to flow along the connecting arcs and are associated with state boxes. There are two kinds of data tokens: *Metadata* and *Datasets*. A *Dataset* token is a file in this demo and can appear in any of the three states. A *Metadata* token can only appear in the DBMS state and represents the existence of a non-empty Illustra table. The actual table contains two fields including a *large_object* field which is a handle to a *large object* stored either in the DBMS state or in the Mass Storage state.

The initial state of the system indicates that a file resides on the client side. Context-sensitive action buttons allow you to choose the data flow paths of interest. For example, initially, one could load the file into the DBMS and have the file's final destination be on the DBMS machine ("*Copy to DBMS*" ACTION button) or one could load the file into the DBMS but have its final location be on the Mass Storage file ("*Copy to Mass Storage*" ACTION button, as in Figure 2.). In either case a *Metadata* token would appear on the DBMS state box, indicating the existence of a non-empty SQL table.

Allowed actions include "*Copy*", "*Delete*", and "*Restart*". Explanations of what was just carried out appear on the bottom of the diagram as well as detailed instructions of what the Illustra Metadata table's content is and how this operation was carried out. Features including a "*large object display*" section are provided. This allows you to display the contents of the file object directly from its current location (Client, DBMS, or Mass

Storage) directly streaming it to the Web browser window without ever going through any kind of intermediate storage. This allows you in particular to verify that the file object has successfully been migrated to its new destination.

While the data movement window is being updated and explanations are being provided, the operations are carried out behind the scenes in real-time. This demo provides a window of observation into the integrated "Web-Database-Mass Storage" environment (Marciano [8]).

4.2 Dealing with time-outs

Time-outs are a delicate issue, given that all three servers (Web, Illustra, UniTree) have their own default time-out thresholds. For example, an unsatisfied Web request will time out after a preset amount of time, generating a message warning you that the server you are trying to connect to might be temporarily unavailable. The following solutions are first-level attempts at dealing with some of the more obvious time-out problems.

Access of *large_objects* stored in UniTree may hang for a long time because of the following two reasons:

1. The UniTree server has died.

2. The *large object* file has migrated to tape. It could take 10 minutes or more to stage a file from tape to disk.

There are at least three scenarios that need to be handled:

1) The Illustra server tries to connect to the UniTree server but the UniTree server is not present. The current UniTree library causes the *Illustra server to hang indefinitely* or almost indefinitely.

> Our **solution** is to make the connection request time out in 30 seconds. An error message is sent to the client when a time-out occurs.

2) The *large object* file has migrated to tape and it may take 10 minutes or more to stage the file from tape to disk. This causes the Illustra server to block until the file is staged.

> Our **solution** is to make the open request time out in 2 minutes. A warning message is sent to the Illustra client every 30 seconds to inform the user what has been taking place. When the 120 second time-out is triggered, another error message is sent to the client before failing.

3) The UniTree server dies when the Illustra server is doing read/write operation to and from UniTree. There is a 2 second time-out for read/write in *libnsl.a*. In this case a regular read/write error message is sent.

5. Environmental Data Testing

This section describes how the example interface in section 3.1 was extended to handle real-world data on an existing environmental sciences application.

Please refer to the web location **"http://www.sdsc.edu/~sdbay"** for more information on the San Diego Bay Project, an environmental data repository which contains chemical, physical, and biological data for the bay of San Diego and which can be accessed over the Web through a clickable map of the Bay. Currently, the project uses flat files and only emulates a database engine.

The integration effort has involved porting a representative subset of this environmental data directly to the Illustra database and experimenting with clickable map search scenarios that allow the data to be displayed over the Web and stored both in the local store of the Illustra DBMS as well as on the UniTree mass storage store.

An example of a clickable map search interface for the *Integrated San Diego Bay prototype* that we are developing is shown in Figure 3. After defining the appropriate geographic box, an SQL query would be submitted to the Illustra server. The query would take the user-specified bounding box and intersect it with the list of registered bounding boxes stored as metadata with each image. The image itself is stored as a *large object* that can reside either on the Illustra side or on the NSL UniTree side. To achieve this result we wrote an Illustra SQL user-defined function (UDF) called GIS_overlap():

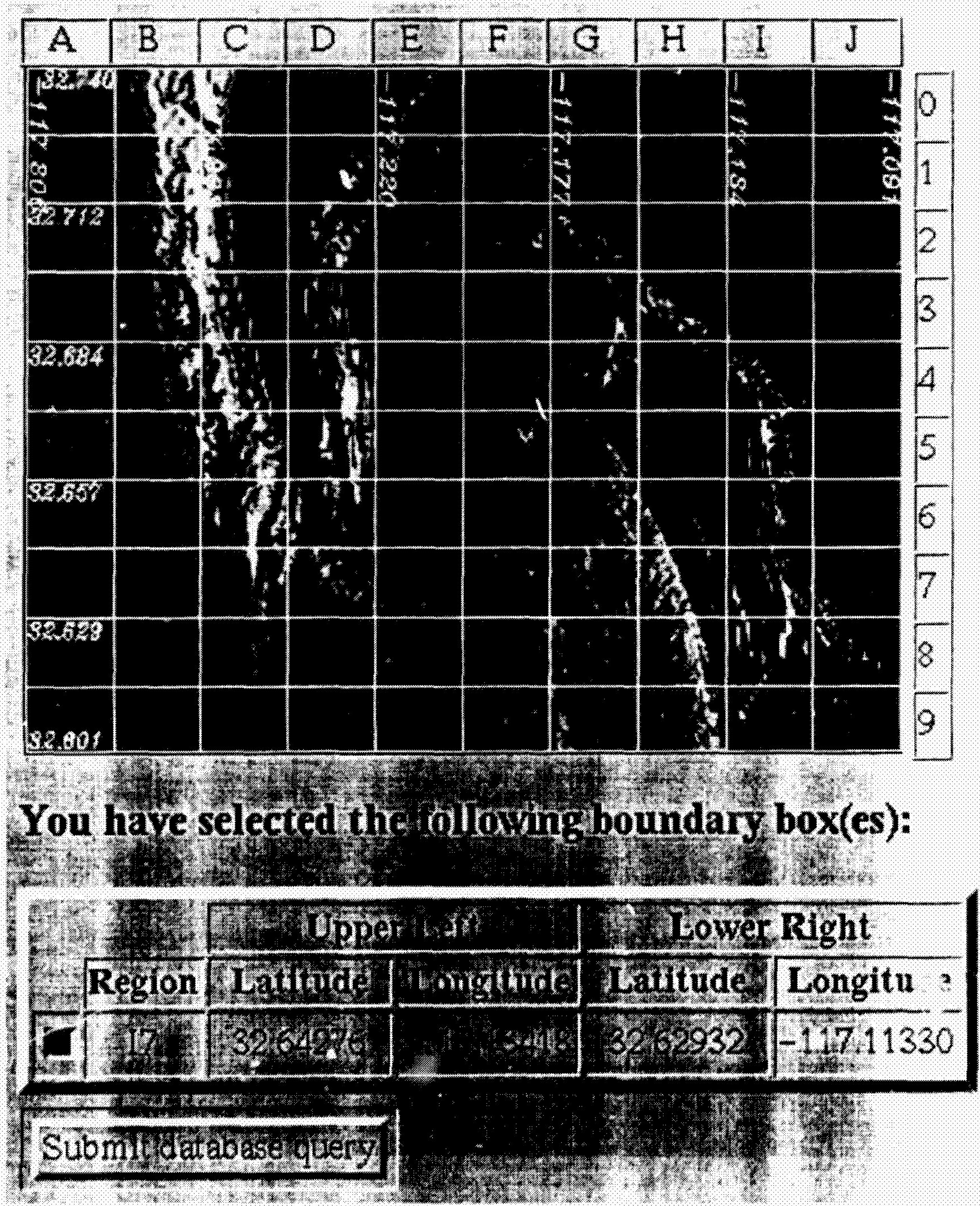
```
create function GIS_overlap( arrayof( real ), arrayof( real ) )
returns boolean
as external name 'GIS_overlap.so'
language C;
```

Databases such as Illustra provide a special 2D Spatial Data *DataBlade* with more efficient "GIS_overlap"-like functions that one could use directly.

The *clickable map interface* (see Figure 3) allows the viewer to build a running list of locations of interest in the bay and submit those to the search engine, which returns a list of environmental files of interest, broken down into those three categories (physical, chemical, and biological).

Clicking on the file name in RECORD 1 in Figure 4 would display the actual file (see Figure 5). Note here that the file is directly streamed from NSL UniTree to the Web browser window.

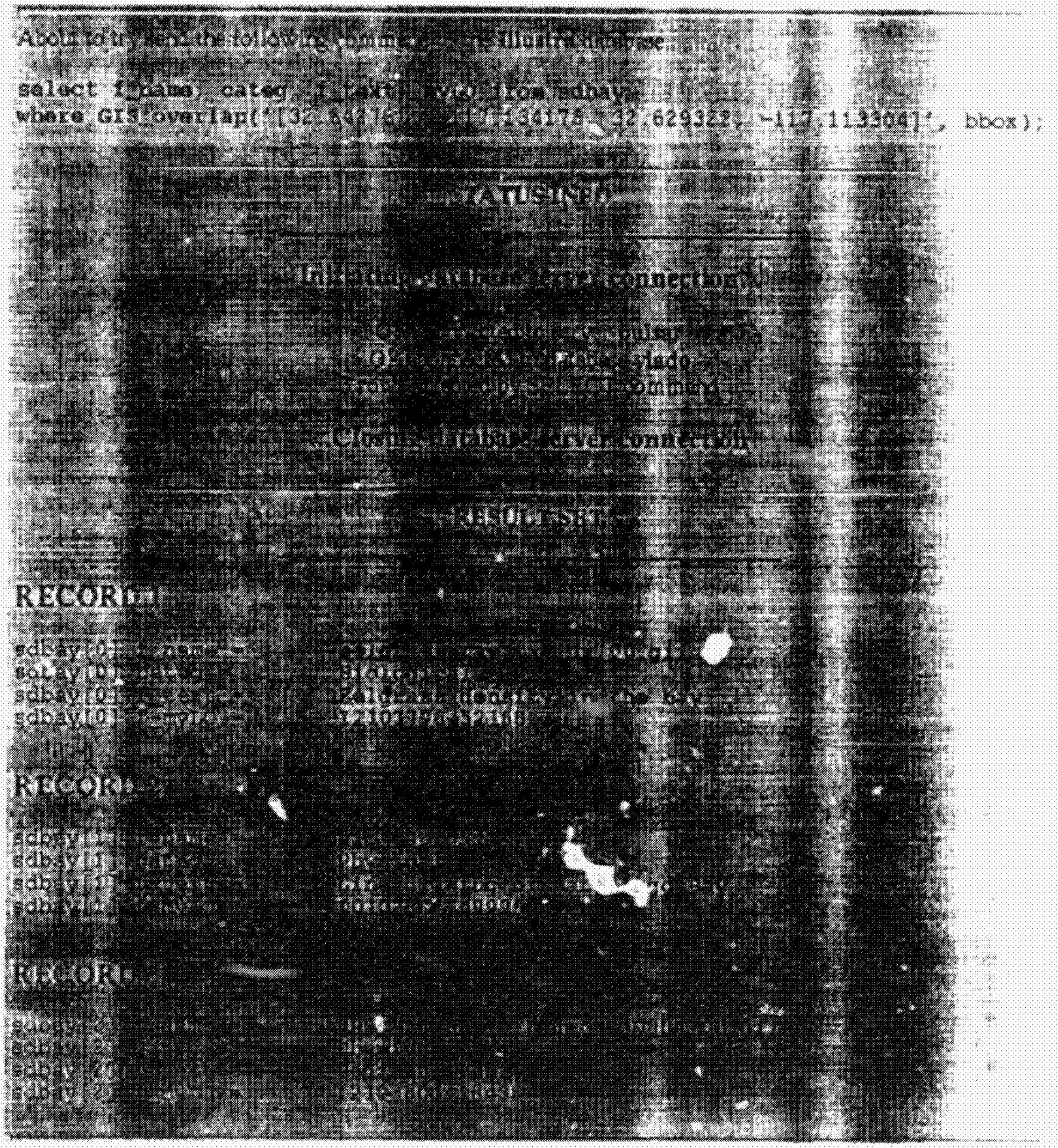
As far as the user is concerned, this is fairly transparent, except when a file has been totally migrated off to tape, and longer waiting periods occur. This flexible scheme allows us to choose from a hierarchy where specific data items might reside on the Web Server's local disk, on the DBMS's disk, or all the way out on the Mass Storage area, which itself allows pre-caching on the host RS-6000 disk. We are currently evaluating where to store this environmental data in the storage hierarchy.



You have selected the following boundary box(es):

		Upper Left		Lower Right	
	Region	Latitude	Longitude	Latitude	Longitude
■	17	32.64276	-117.13418	32.62932	-117.11330

Figure 3: Clickable Map Interface



```
About to try [illegible] the following [illegible] illustrate [illegible]

select f_name, categ [illegible] from sdbay [illegible]
where GIS_overlap('[32.64[illegible] [illegible] 34178 32.629322, -117.113304]', bbox);

[illegible]TATUS INFO

Initiating [illegible] connection

[illegible]
[illegible]
[illegible] SELECT command

Closing [illegible] connection

[illegible] RESULT SET

RECOR[illegible]

sdbay[0[illegible]      [illegible]
sdbay[0[illegible]      [illegible]
sdbay[0[illegible]      [illegible]
sdbay[0[illegible]      [illegible]

RECOR[illegible]2

sdbay[1[illegible]      [illegible]
sdbay[1[illegible]      [illegible]
sdbay[1[illegible]      [illegible]
sdbay[1[illegible]      [illegible]

RECOR[illegible]

sdbay[2[illegible]      [illegible]
sdbay[2[illegible]      [illegible]
sdbay[2[illegible]      [illegible]
sdbay[2[illegible]      [illegible]
```

Figure 4: SQL query results

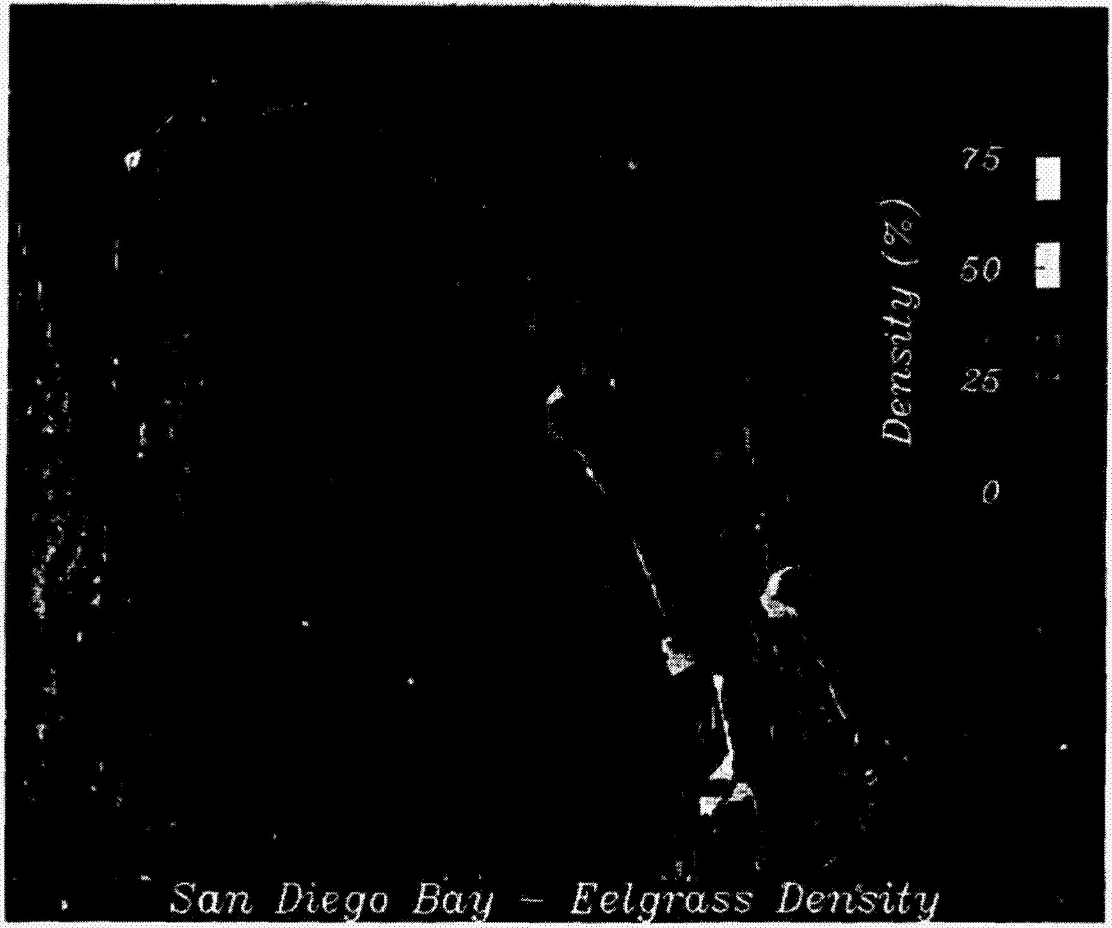


Figure 5: Sample San Diego Bay Repository environmental data set

6. Expanded data access scenarios

The integration of database, archival storage and Web technology promises to facilitate ':e manipulation of large data sets and large collections of data sets. One goal is to enable data analysis on terabyte-sized data sets retrieved from petabyte archives, at an access rate of 10 GB/sec. Current supercomputer technology supports a 1 GB/s access rate to 1 terabyte of disk. For a teraflops supercomputer with 10 TB of disk, data rates on the order of 10 GB/s will be feasible. This will require, however, support for parallel I/O streams, and support for striping data sets across multiple peripherals. Fortunately, the software technology to support third party transport of data sets across parallel I/O streams is being developed in the HPSS archival storage system (Coyne [9], Watson [10]). Data redistribution mechanisms for the parallel data streams are being standardized as part of the MPI-IO (Snir [11, 12]) effort. The expectation is that the initial usage prototypes described above can be extended to support supercompu.:r applications that analyze arbitrarily large data sets.

A second goal is to provide ubiquitous access to scientific data sets. Scientific applications should be able to access data and cache it locally no matter where the data is originally located. Some of the key requirements of such a system are:

- **heterogeneous data sources:** Possible sources for data include databases, archives, file systems, and anonymous FTP servers on the Web. An API is needed that will allow an application to specify a data source, establish a connection, select a data set based on requested attributes, and then cache the data set locally.

- **parallel I/O:** Because of the size and number of data sets that can be accessed for analysis, mechanisms for redistribution of data sets from multiple peripherals onto parallel compute nodes are needed. The emerging MPI-IO standard will be the foundation for the API we are constructing.

- **distributed computation support:** Data sets may be distributed to multiple platforms, for analysis by methods that are retrieved from ORDBMS. Support for distribution of computation objects is needed.

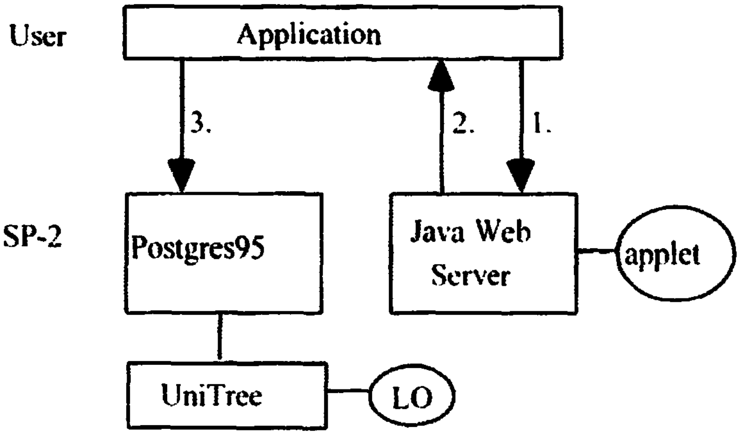
- **third-party data access:** If both data sets and computational methods are distributed to a remote platform. mechanisms are needed to allow the method to access a temporarily cached data set.

- **third-party authentication:** Similarly, methods and data sets need to validate their interoperation through an authentication mechanism that is independent of the local operating system.

The end result is a data handling environment where the focus is on moving and caching data rather than moving and distributing applications. The operating system at each server or compute platform controls use of the local resources. The data handling environment provides a higher-level infrastructure that supports remote access, authentication, and data movement.

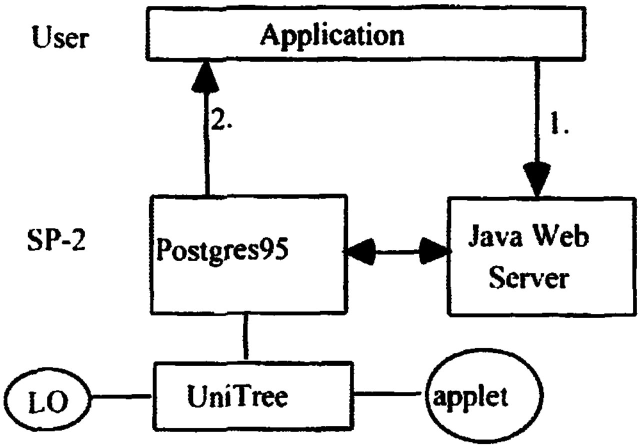
An example of this environment is shown in **Scenario I**. A user makes a request of a remote system for a particular data set. The process consists of retrieval of an applet stored on the local disk of the system, which is then used to access an ORDBMS database. The data object is retrieved from the archive that is linked to the database.

A prototype of this system based on a Java interface to the Postgres95 ORDBMS is being developed. Interfacing Java to Postgres95 required porting the Postgres95 client interface library to Java. This enables a Web client that has Java capabilities to interact directly with the Postgres95 DBMS. Part of this work has been inspired by a prototype developed by John Kelly at the Blackdown site (ftp: // substance.blackdown.org / pub / Java / Java-Postgres95). In particular, we have added support for large objects, a functionality that was not provided earlier.



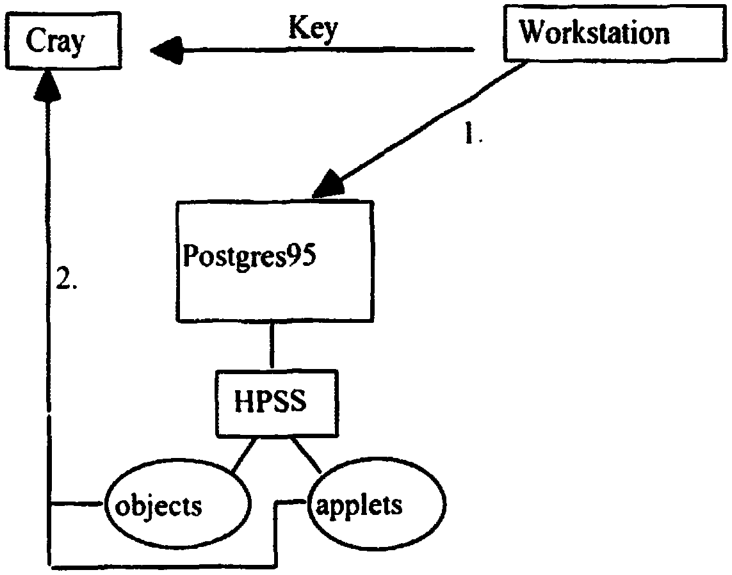
Scenario I: Pulling out applets and large object data

An improvement to this architecture is shown in **Scenario II**. The applet is stored as a method within the ORDBMS. The request to the Web server results in the extraction of the applet out of the archival storage system, and its transmission to the remote user. The applet is then executed on the remote system to access data objects through the ORDBMS system.



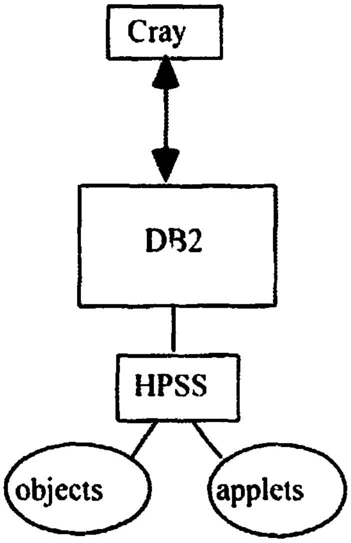
Scenario II: Pulling out large object applets

A further extension of the system is shown in **Scenario III**. The Web server interface is directly integrated into the ORDBMS. A request for analysis of a data object results in both the data object and the associated method being moved to a compute platform. To provide data privacy, the data set may be encrypted. The encryption key is sent to the method, thus providing both third-party authentication and mechanisms for controlling third-party data access.



Scenario III: Third-party data access and third-party authentication

Finally, as shown in **Scenario IV**, the above capabilities can be implemented directly within I/O libraries that are used : · a scientific application. The application then directly accesses the remote database/archival storage system to retrieve a data set. Data subsetting and redistribution can be provided by application of appropriate methods to the data set on the compute platform which supports the ORDBMS.



Scenario IV: Supercomputer analysis of scientific data sets

Acknowledgments

This work was supported in part by DARPA grant F19628-95-C-0194 on Massive Data Analysis Systems, and in part by the NSF cooperative agreement ASC-8902827 for the San Diego Supercomputer Center.

References

1. R. W. Moore, "High Performance Data Assimilation," Proceedings of the Committee on Information and Communications R&D (CIC) of the National Science and Technology Council, July, 1995.
http://www.sdsc.edu/EnablingTech/InfoServers/HPDA.html

2. R. W. Moore, "Distributed Database Performance," SDSC Report GA-A20776, (December 1991).

3. The Design of a Parallel Data Handling System for Scientific Data Management and Mining, Reagan W. Moore, Richard Frost, Mike Wan, Joe Lopez, Richard Marciano. Submitted to PDIS, December 1996, Daytona Beach, Florida. http:// www.sdsc.edu / EnablingTech / InfoServers / parallel-mining.html

4. Illustra - Illustra Information Technologies Inc, "Illustra User's Guide",1995.

5. M. Stonebraker et al., "The Implementation of Postgres", IEEE Transactions on Knowledge and Data Engineering (March 1990).

6. M. Stonebraker and G. Kemnitz, "The POSTGRES Next-Generation Database Management System," Communications of the ACM, 34 (10), 78-92 (October 1991).

7. Archival Storage Research at SDSC, http://www.sdsc.edu/EnablingTech/archstor.html

8. Richard Marciano, "High Performance Computing Web-Based Simulation Environments", High Performance Computing 96, New Orleans, LA, April 8-11, 1996.

9. R. A. Coyne, H. Hulen, and R. W. Watson, "The High Performance Storage System." Proc. Supercomputing 93, Portland, IEEE Computer Society Press (November 1993).

10. R.W. Watson and R.A. Coyne, "The Parallel I/O Architecture of the High-Performance Storage System (HPSS)", the 1995 IEEE MSS Symposium.

11. M. Snir, S. W. Otto, S. Huss-Lederman, D. W. Walker, and J. Dongarra. MPI: The Complete Reference. MIT Press, 1995.

12. Marc Snir, Peter Corbett, Dror Feitelson, and Jean-Pierre Prost. Draft Document for a Parallel MPI IO Library. (Draft document presented for informal consideration in MPI-2 standardization process), January 14 1994.

NEXT

DOCUMENT

The Challenges Facing Science Data Archiving on Current Mass Storage Systems

Bernard Peavey and Jeanne Behnke
Earth Science Data and Information Systems Project
Code 505
Goddard Space Flight Center
Greenbelt, MD 20771
bernie.peavey@gsfc.nasa.gov
301-614-5279
jeanne.behnke@gsfc.nasa.gov
301-614-5326

Introduction

This paper discusses the desired characteristics of a tape-based petabyte science data archive and retrieval system (hereafter referred to as "archive") required to store and distribute several terabytes (TB) of data per day over an extended period of time, probably more than 15 years, in support of programs such as the Earth Observing System (EOS) Data and Information System (EOSDIS) Kobler [1]. These characteristics take into consideration not only cost-effective and affordable storage capacity, but also rapid access to selected files, and reading rates that are needed to satisfy thousands of retrieval transactions per day. It seems that where rapid random access to files is not crucial, the tape medium, magnetic or optical, continues to offer cost effective data storage and retrieval solutions, and is likely to do so for many years to come. However, in environments like EOS, these tape based archive solutions provide less than full user satisfaction. Therefore, the objective of this paper is to describe the performance and operational enhancements that need to be made to the current tape based archival systems in order to achieve greater acceptance by the EOS and similar user communities.

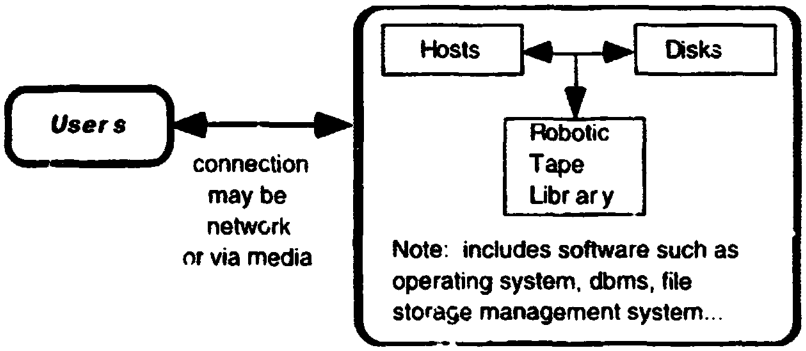

Users
connection may be network or via media
Hosts
Disks
Robotic Tape Library
Note: includes software such as operating system, dbms, file storage management system...

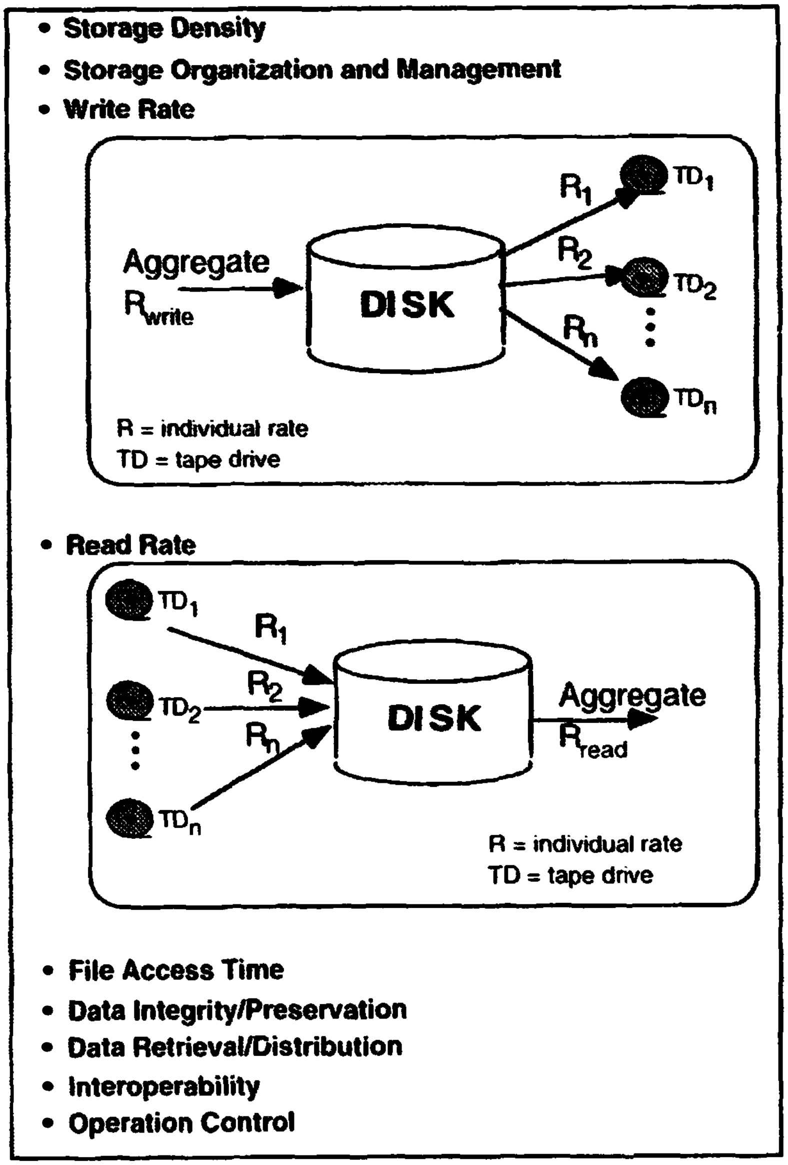
Figure 1: Generic Tape-based Archive

The archive discussed in this paper is shown in Fig.1. Its basic components - host, magnetic disk (perhaps solid state or holographic memory in the not too distant future) for caching/staging (hereafter referred to as "disk"), robotic tape library, input/output media devices, and associated software (operating system, database, file management, resource management, network, communication protocol, operation control, etc.) are assumed to be fully integrated as an operational system, which could be centralized or distributed as appropriate to the user environment and data sources. The archive architecture and configuration are assumed to be such as to allow expansion or growth from a nominal one petabyte to 100 petabyte storage and performance capacity as data continue to be accumulated and the number of users continues to increase. The archive is expected to store and retrieve a variety of data types, the files of which may range from 1 KB (kilobyte) to 10 GB (gigabyte) in size, and handle thousands of user transactions a day. Being an operational system required to satisfy a multitude of users (vs. a laboratory facility), this archive is, therefore, characterized from a system's rather than a component's perspective. For example, the performance of a given tape drive is not addressed directly; rather, the data transfer rate from disk to tape or from tape to disk, including all overhead associated with managing each data file before it lands in a given location, is specified. Thus, the salient archive characteristics addressed in this paper are: storage density, storage organization and management, write rate, read rate, file access time, data integrity/preservation, data retrieval/distribution, data interchange or interoperability, and operation controi. They are examined from an operational system's perspective to highiight their significance in realizing the archive's desired capabilities.

Given the state of current technology and available archive components as desciibed in the literature Shields [2] and observed in the field, can the subject archive be offered by the vendor community at an affordable price? This twofold question of performance and cost is examined from the standpoint of real progress already made i. this area - a reality check, and what remains to be done to reach the goal of achieving the desirable archive characteristics at an affordable price.

Salient Characteristics

In discussing the archive's salient characteristics, it is assumed that the system arc. itecture allows the use of multiple tape drives, robots, disk banks and hosts as appropriate to achieve the desired capacity and performance, and the local network bandwidth is sufficient to support this performance. As mentioned previously, these characteristics, which become specifications when they are given specific/particular values, are considered from the standpoint of a fully operational system, and their measurements are made on this basis as well. This means that for systems which utilize multiple components operating in parallel, e.g., tape drives or disk drives, characteristics such as data transfer rate (write or read) are given as aggregate values, as illustrated in Fig. 2. In general, characteristics associated with data transfer or data flow are considered to be "end-to-end", viz., for storage, data transfer begins when the data enters the host, and for retrieval, data transfer ends when data lands on the archive disk shown in Fig.1. System level characterization of the archive is key to describing the archive's capabilities in realistic terms and relating them to operational expectations. Regrettably, the practice of characterizing archives at the system level is not yet standard or even prevalent, perhaps because the vendor community does not usually offer integrated archives as products. Instead, archives are typically specified in terms of performance of their components such as tape drives, tape libraries, etc., which means that a great deai of system engineering and development effort must be applied by or provided to the customer in order to realize the complete archive solution. From the archive customer's perspective, procuring the archive on the basis of system level characteristics presents the vendor community with an opportunity to offer fully integrated archive systems as products and, hopefully, at lower cost to the customer. In any event, what follows are the desired archive characteristics as seen by the end user. It should be noted that at this time there are no commercial-off-the-shelf (COTS) tape-based archive systems that include all of the desired characteristics. Adding new features to COTS products tends to be very costly. Thus, by examining the following characteristics, it may be possible to identify opportunities to enhance existing COTS products or to develop new products.



- Storage Density
- Storage Organization and Management
- Write Rate

- Read Rate

- File Access Time
- Data Integrity/Preservation
- Data Retrieval/Distribution
- Interoperability
- Operation Control

Figure 2: Salient Characteristics of a Tape-based Archive

Storage Density

Storage Density, given in terms of bytes/in, bytes/cm, or bytes/tape (with known tape dimensions, i.e., width and length), is directly related to the archive's storage capacity. For example, the D3 tape cartridge is advertised to hold 50 GB. Actually, from a system's perspective, the effective storage density is lower due to the associated file management overhead, which increases with the number of files. In addition, data compression, if used, must also be taken into account. Therefore, this characteristic should be given in te., ns of effective storage density. A petabyte (PB) archive using 50 GB tape cartridges requires 20,000 cartridges which, at $50/unit, amounts to $1,000,000! Both numbers are prohibitive, especially when extended to a 100 PB archive. Clearly, a tenfold increase in storage density would be welcome within the next few years, and a 1 TB per tape capacity wou'd be required in the near future. But, increased capacity (at the same cost and overall size, of course) alone is not enough without higher read/write rates, and shorter file access time to sustain a reasonable performance level. Is this a technological challenge, economic (commercial demand) challenge, or both? The likely answer is that the challenge is economic, but time will tell.

Storage Organization And Management

Storage Organization And Management (SOM) provides the capability to control the way in which data files (hereafter referred to as "files") are stored on and retrieved from tape. For example, SOM selects tape drives (hereafter referred to as "drives") and tapes, directs the flow of files to/from selected drives, provides logical and physical file organization, maintains knowledge of file location and status, causes the robotics to load or unload selected tapes (volume mounting/dismounting), controls access to each file, and keeps statistics on file access frequency. In discussing this characteristic, it is assumed that SOM also controls the availability of the cache/staging disk (Fig. 1), which is part of the archive. The criticality of this system level characteristic cannot be overstated with regard to system performance, especially when ordered files such as those arriving from Landsat USGS [3] are requested to be retrieved in random subsets, and the system has to manage a continuously increasing file inventory on the order of 1-10 billion files.

In order to allow system performance tuning, the SOM should include, among others, the following selectable options for writing files onto tapes:

(1) Chronological order

(2) No file splitting across tapes

(3) File continuation on second tape (The first tape must identify the existence of a partial file and provide the identification of the second tape. The second tape must identify the existence of a continuation file and provide the identification of the first tape. Note that no more than 2 partial files may exist on a given tape: the beginning part of one, and the continuation part of another.)

(4) Unique file grouping (Writing a uniquely identifiable file collection on the same tape, e.g., files from a certain scientific instrument)

(5) Superfiles (Writing a collection of files as a super file so as to be retrieved as one super file or as individual files)

(6) Data compression (Per whole tape)

(7) Maximum tape utilization (Random collection of files to minimize unused tape)

(8) File replication (Writing the same file to different tapes or to the same tape)

(9) Tape duplication (Writing multiple tapes of same files simultaneously)

(10) Simultaneous file recording (Writing multiple files to multiple tapes simultaneously, see Fig. 2)

For data retrieval, the SOM must provide the following read options:

(1) Ordered files from a single tape (Per requested sequence)

(2) Ordered files from multiple tapes (e.g., k files from tape 1, m files from tape 2, n files from tape 3, etc.)

(3) Interleaved files from multiple tapes (e.g., file A from tape 1, file B from tape 2, File C from tape 3, etc.)

(4) Superfiles (Collecting multiple files from a single tape or multiple tapes into one file as requested)

(5) Compression/decompression

(6) Tape quality information

The SOM must also include the capability to produce or write tapes that are self describing so as to be read on any compatible drive external to this archive.

As a file manager of a growing archive, the SOM must be scalable to accommodate a 100 fold (from 100 million to 10 billion) increase in the number of files. In addition, it should be applicable to centralized as well as distributed archive architectures. It would be nice if the disk shown in Fig. 1 could be eliminated while still providing the desired SOM, since by so doing the scalability problem could easily be solved, and one data flow hop could be eliminated as well. However, barring that possibility, separating file management and volume management should be considered as part of the scalability problem solution. In addition, advantage should be taken of this disk to improve the efficiency of file storage management and retrieval (e.g., executing the various writing options stated above, distributing a given file to multiple users, collecting files located on multiple tapes to satisfy a single data request). In general, the SOM should have the necessary features to optimize the overall file storage and retrieval performance, while being independent of any operating system (OS) as much as possible. This independence is crucial for the SOM software to be able to run on any hardware platform, present or future, which is key to evolvability.

Although a number of SOM versions such as UniTree, AMASS, FileServ, which are known as File Storage Management Systems (FSMS), are presently in use, they incorporate only a few of the SOM options, and are strongly dependent on the platform's OS. Also, these FSMS do not conform to any standard since none exists yet. To achieve plug and play COTS FSMS (or SOM) products, the vendor community must support the development and adoption of a FSMS standard. It appears that the efforts made by the IEEE and ISO over the years to develop an open systems standard have not borne fruit yet. However, some activity in this area has been afoot which provides an opportunity to revitalize this effort. Kobler [4], Jones [5].

Write Rate

This characteristic defines the time required to read incoming files from the disk and writ· (store) them to tape so that they can be retrieved upon request. As a system levei characteristic, it includes the time to uniquely identify each file, append location metadata, select the drives, load the tapes, perform compression (when required) perform error protection for error detection and correction, write the files, update the catalog/database, and return status. The write rate is given for a single or a multiple drive configuration. For a multiple drive configuration, the write rate is the aggregate rate, viz., $R(w) = R(1) + R(2) + ... + R(n)$, where R() are the individual write rates with all n drives writing simultaneously (See Fig. 2). For example, if the incoming data rate is 10 MB/sec (as expected from EOS), the system could handle it with one drive, which must be capable of writing at a rate greater than 10 MB/sec in order to compensate for delays due to FSMS overhead, and physical tape handling functions such as robotics, loading and unloading. Alternatively, the system could accommodate this incoming data rate with multiple drives writing simultaneously at an individual drive write rate lower than 10 MB/sec. Therefore, the write rate (which could also be referred to as "storage rate") is the effective end-to-end system rate at which files can be stored in the archive. It is assumed that in cases where unique file grouping is required, the disk provides sufficient staging and buffering capacity to feed the drives. To write files onto a 50 GB D3 tape cartridge at 10 MB/sec requires the use of drives that cost $150,000 each, which is expensive. It appears that the drive write rate needs to be increased by a factor of 2 or more, and the drive cost needs to be reduced considerably to make a petabyte archive more affordable.

Read Rate

This characteristic defines the time required to read (retrieve) files from tape and write them to the disk for distribution. As a system level characteristic, it includes the time to read the data request, identify and locate the tapes of the requested files, access the files, read the files and write them to the disk with error detection and correction (EDAC) applied, append the metadata, and return status. This read time is comprised of 2 components: file access time, and the time to read the file. The file access time is described in the next paragraph as a separate characteristic, although it is included here as part of the read rate definition for completeness. The read rate is given for a single or a multiple drive configuration. For a multiple drive configuration, the read rate is the aggregate rate, viz., $R(r) = R(1) + R(2) + ... + R(n)$, where R() are the individual read rates with all n drives reading simultaneously (See Fig. 2). For example, if the required outgoing data rate is 30 MB/sec (as expected for EOS), the system could support it with one drive, which must be capable of reading at a rate greater than 30 MB/sec in order to compensate for the delay due to file access time. Alternatively, the system could accommodate this outgoing data rate with multiple drives reading simultaneously at individual drive read rates lower than 30 MB/sec. (Of course, if all requested files were to be located on the same tape, the multiple drive configuration would not meet the 30 MB/sec output rate). Therefore, the read rate (which could also be referred to as "retrieval rate") is the effective end-to-end system rate at which files can be retrieved from the archive. It should be noted that, based on current technology, the file access time can become so significant when many files have to be accessed on many tapes as to require additir al drives to compensate for it. The requirement for multiple drives should also be considered in lignt of the user response requirements, namely, the number of users that need to be served simultaneously. This aspect is discussed later as part of the Data Retrieval/Distribution Characteristic. Generally, the read rate requirement is significantly more stringent than that for the write rate, not only because more data is going out of the archive to users, but also due to the need to minimize waiting time for non-uniform data request distributions. Therefore, to accommodate thousands of transactions a day, the archive may have to utilize 10-20 drives which, on the current market, may cos! $1.5 million to $3 million. This is prohibitive, and points to the need for improved drive performance and cost reduction.

File Access Time

File Access Time (FAT) which is part the previous read rate characteristic, is the total system time required to locate a given file in a tape-based archive following the issuance of the request to retrieve it. This time includes file identification, drive and tape selection, robotic motion/travel, loading the tape, reaching the desired file in a position ready to be read, unloading and returning the tape to its bin. The current technology achieves a FAT of 1-2 minutes, depending on the tape length and file location. Clearly, this lowers the effective retrieval rate, especially when many files have to be retrieved from many tapes. To cope with such a delay, today's archives must utilize multiple drives, with attendant cost increases. Therefore, the FAT must be reduced by a factor of 3 or more to improve the cost performance ratio, and allow the on-line user to start receiving data within less than one minute from the time of having made the request.

Data Integrity/Preservation

A persistent archive requires that files stored on tape be entirely preserved with no degradation of their content during the archive's life (30 years). Therefore, the system must be capable of monitoring the state of data quality (e.g., BER), and the physical condition of the medium to determine when to refresh (transcribe to a new tape) or just rewind a given tape, and do so automatically or under operator control. These actions should be based on frequent checks of the BER, which should not exceed 1 in 10 to the 12th bits (each time a file is read or at specified time intervals), file access frequency, and time in storage. In addition, a backup capability is needed to make and manage copies of selected tapes or files. Since in today's systems the capability of this characteristic seems to be limited to manual intervention, this capability should be enhanced to the fullest level.

Data Retrieval/Distribution

This characteristic defines the manner in which files are to be retrieved and distributed to users electronically or on media (tape, CD-ROM, the drives of which are assumed to be included in the archive). For example, it should be possible to retrieve and distribute files in whole or in part, in specified order (e.g., chronological - oldest file first, or most recent file first; per list specified in the request; or other), grouped by category (e.g., instrument; science discipline; product type), random file collections, file interleaved by tape (a given file from tape 1 followed by a given file from tape 2, etc.), and compressed or uncompressed format. Format conversion is a separate service which may be included in the archive system. This system level characteristic applies to both software (FSMS or SOM, DBMS, request processing) and hardware components' performance in order to achieve the desired data outflow rate. It is assumed that an appropriate DBMS is available and is included in the archive to serve the file catalog and file search functions, however, the schema design and implementation is a user provided application. It is also assumed that the disk capacity and speed (data transfer rate), the number of drives and their read rates are sufficiently high to support the required data distribution rate and the number of simultaneous data requesters.

As mentioned previously in the Read Rate paragraph, to support the requirement to retrieve and distribute several terabytes of data per day in response to thousands of transaction requests is very demanding of software (FSMS, DBMS, NFS) and hardware performance. With today's technology available on the market, this requirement can be met only by using lots of expensive hardware. Therefore, it is imperative that the hardware performance and reliability be greatly improved to make petabyte archives less costly.

Interoperability

This characteristic is intended to allow the archive components to be changed out in a "plug and play" manner without affecting the archive's functionality, and to support media-based data interchange (providing data to and distributing data from archives and users) among archives and users. In addition, the archive architecture must provide for the application software and user interface software to be independent of a given hardware platform and its OS. Thus, this characteristic, allows the archive to be scalable and evolvable as capacity and performance requirements continue to grow, and superior technology becomes available. To realize such a characteristic, COTS products (hardware and software) must comply with appropriate standards which are yet to emerge. Regrettably, today's products do not lend themselves to open interchanges. For example, tape formats are unique to the systems, FSMS are tailored to specific platforms and OS, and information describing their implementation is proprietary.

With regard to developing archive system standards, it should be mentioned that the work begun under the IEEE and ISO sponsorship has not progressed as far as was expected. Perhaps this slow progress can be attributed to the approach undertaken by these groups, without realizing that advances in archive and Internet technology are occurring at a much more rapid pace than anticipated, thus diminishing the desire of system developers and vendors to wait for these standards before participating in the market and application opportunities. A better approach to developing archive system standards would be the model of the IETF. As Dave Clark of the IETF said in 1992: The IETF (Internet Engineering Task Force) credo is:

> "We reject kings, presidents, and voting.
> We believe in rough consensus and running code."

Perhaps this nontraditional approach taken by the IETF group should be followed in developing the standards for Mass Storage Systems (MSS) and FSMS. Rather than following a top-down approach to include "all or nothing", it might be more productive and effective to pursue the incremental and less rigorous approach with the notion that "having a standard is better than none". The EOSDIS Project at the Goddard Space Flight Center is participating in the effort to develop these standards, and is committed to using them.

Operation Control

This characteristic describes the extent to which system operation should be controlled automatically. The most desirable feature would be full automation or "lights out" mode of operation, where the only required interface is the user, while the operator/technician performs maintenance, or user services type functions. To achieve a high degree of automatic control, the system must be capable of self checking, monitoring ongoing activities, sensing critical conditions and reacting to them, controlling resources, balancing workloads, managing request queues, tracking user requests to the file level, accounting for resource utilization per user request, helping users, monitoring system performance and quality, collecting production statistics, reporting and logging events, issuing remedial instructions, etc. (Also, it would be nice to have the system repair itself, but for now this must remain a dream to come true). Unfortunately, today's systems require considerable operator intervention in running an archive. Therefore, such intervention should be minimized at best in order to control the operation cost.

Discussion

A growing tape-based petabyte archive for science data, which is the subject of this paper, is described in terms of its salient characteristics, and their implication on the architecture, implementation, acquisition, and cost thereof. Ideally, these functional and performance characteristics should be sufficient to specify the desired archive (large or small) so that it could be procured at a reasonable price from a given vendor as a COTS product, consisting of COTS components which the vendor would select, integrate, test, demonstrate, and turn over to the customer as a fully operational archive. The customer's involvement in this process would be minimal except for a fixed price proposal/bid evaluation and acceptance testing. To use the archive acquisition approach described above, which is expected to result in considerable cost savings, the customer must know what is needed, the technology must be mature, suitable components must be available as COTS products that are compliant with industry standards, and there must be a market for these components. By examining these characteristics in light of available COTS products, the aforementioned premises are not all satisfied at this time. The most critical of these premises are technology and standard COTS products that would satisfy the desired functionality and performance requirements at a reasonable cost. Historically, not much has happened until 1995, when new tape drives and cartridges were introduced that boosted the read/write rates to 10 MB/sec, and increased the storage capacity to 20 GB per 3480 type cartridge (higher capacities are on the way, e.g., the D3 cartridge). However, more work is needed to produce a 1 TB cartridge, and a 30 MB/sec read rate drive with a file search time of less than 20 seconds anywhere on the tape. In the DBMS and FSMS areas, plug and play products are not yet available. Perhaps there will be an opportunity to develop a standard modular (to allow for incremental addition of features and scalability) SOM product which can be plugged into a microkernel type OS. Of particular interest and concern are the scalability and evolvability aspects of FSMS and DBMS COTS products in the absence of open system standards. The promises made in 1991 Rybczynski [6], McLean [7] toward the realization of petabyte archives have been slow in coming. It seems that the challenge to do so is still up for grabs.

The salient characteristics approach describes and specifies the archive at a system level because these characteristics are directly related to the user's needs or expectations, and can be measured on that basis. By so doing, the vendor is offered the opportunity to be creative and cost-effective in producing the optimum archive system in terms of functionality and performance. For example, selection of the type and number of tape drives should be a key consideration for a petabyte tape-based archive to achieve the required storage and retrieval rates, and to satisfy the required number of simultaneous user requests. Similarly, the vendor has the choice of selecting the hardware platforms and disks, as well as the appropriate software components. (Please note the emphasis on the vendor rather than the customer). Thus, vendors have the opportunity to offer standard archive components, or fully integrated, scalable turn-key archives. At this time, it is still necessary to stage files on disk as part of the storage and retrieval operation. (How nice it would be if disks could be eliminated from this operation). Therefore, adequate disk capacity and speed (data transfer rate) must also be a key consideration.

In conclusion, it appears that affordable (less than $10 million) tape-based petabyte archives for science data are difficult to find on today's market. However, it might be possible to find them in the near future with the help of enhanced technology, standard COTS products supporting plug and play system architectures, system level procurement specifications, integrated archive system products, turn-key system acquisition, and open storage system standards. The time must come when a 1 petabyte archive could be expanded or scaled up 100 times by simply replacing (plugging in) existing components with new more powerful components as they become available, in a manner completely transparent to the user, and at reasonable cost. That is still a challenge.

Acknowledgements:

We wish to acknowledge the invaluable assistance of P.C. Hariharan of Systems Engineering and Security, Inc in the preparation of this document.

References:

1. Kobler, B. J. Berbert, P. Caulk. P.C. Hariharan, "Architecture and Design of Storage and Data Management for the NASA Earth Observing System Data and Information System (EOSDIS)," *Fourteenth IEEE Symposium on Mass Storage Systems*, Monterey, CA, November 1995.

2. Shields, M., "Toward a Heterogeneous Common/Shared Storage System Architecture," *Fourteenth IEEE Symposium on Mass Storage Systems*, Monterey, CA, November 1995.

3. US Geological Survey and National Oceanic and Atmospheric Administration, "Landsat User's Guide," available from the EROS Data Center at URL: http://edcwww.cr.usgs.gov/glis/hyper/guide/landsat_tm

4. Kobler, B. and J. Williams, "A Straw Man Proposal for a Standard Tape Format," *AIIM International Conference (IEEE)*, Chicago, IL. 1996.

5. Jones, M., J. Williams, and R. Wrenn, "A Proposed Application Programming Interface for a Physical Volume Repository," *Fifth NASA Goddard Conference on Mass Storage Systems and Technologies*, September 1996.

6. Rybczynski, F., "Network Accessible Multi-Terabyte Archive," *Proceedings of the NSSDC Conference on Mass Storage Systems and Technologies for Space and Earth Science Applications*, Goddard Space Flight Center, July 1991.

7. McLean, R. and J. Duffy, "ICI Optical Data Storage Tape," *Proceedings of the NSSDC Conference on Mass Storage Systems and Technologies for Space and Earth Science Applications*, Goddard Space Flight Center, July 1991.

NEXT
DOCUMENT

Processing Satellite Images on Tertiary Storage: A Study of the Impact of Tile Size on Performance[1]

JieBing Yu
David J. DeWitt
Computer Sciences Department
University of Wisconsin-Madison
1210 W. Dayton St.
Madison WI 53706
{jiebing, dewitt}@cs.wisc.edu
608-263-5489 FAX: 608-262-1204

1. Introduction

In July 1997, NASA will begin to launch a series of 10 satellites as part of its *Mission to Planet Earth*, more popularly known as EOSDIS (for Earth Observing System, Data Information System). When fully deployed, these satellites will have an aggregate data rate of about 2 megabytes a second. While this rate is, in itself, not that impressive, it adds up to a couple of terabytes a day and 10 petabytes over the 10 year lifetime of the satellites [1]. Given today's mass storage technology, the data almost certainly will be stored on tape. The latest tape technology offers media that is very dense and reliable, as well as drives with transfer rates in the same range as magnetic disk drives. For example, Quantum's DLT-4000 drive has a transfer rate of about 3.0 MB/sec (compressed). The cartridges for this drive have a capacity of 40 GB (compressed), a shelf life of 10 years, and are rated for 500,000 passes [2]. However, since tertiary storage systems are much better suited for sequential access, their use as the primary medium for database storage is limited. Efficiently processing data on tape presents a number of challenges [3]. While the cost/capacity gap [4] between tapes and disks has narrowed, there is still about a factor of 2 in density between the best commodity tape technology (20 gigabytes uncompressed) and the best commodity disk technology (10 gigabytes uncompressed) and at least a factor of 4 in total cost ($2,000 for a 10 GB disk and $10,000 for a 200 GB tape library).

Raw data from a satellite is termed level 0 data. Before the data can be used by a scientist it must first undergo a number of processing steps including basic processing (turning the electrical voltage measured for each pixel in a image into an digital value), cleansing, and geo-registration (satellites tend to drift slightly between passes over the "same" area). The end result is a level 3 data product consisting of a series of geo registrated images that an earth scientist can use for his/her research. Processing actually expands the volume of data collected by a factor of 2 or 3 and the original data received from the satellite is never deleted. Thus, the processing and storage requirements actually exceed the 2 terabytes/day figure cited above. As part of the EOSDIS project, NASA has contracted with Hughes to build such a system.

Once processed the data is ready for analysis by an earth scientist. Analysis involves applying a series of algorithms (typically developed by the earth scientists themselves) to a large number of images in a data set. Frequently a scientist will be interested in a certain type of images for a particular region of the earth's surface over an extended period of time.

The focus of this paper is how best to handle images stored on tape. We make the following assumptions:

[1] This work is supported by NASA under contracts #USRA-5555-17, #NAGW-3895, and #NAGW-4229, ARPA through ARPA Order number 018 monitored by the U.S. Army Research Laboratory under contract DAAB07-92-C-Q508, IBM, Intel, Sun Microsystems, Microsoft, and Legato

1. All the images of interest to a scientist are stored on a single tape.
2. Images are accessed and processed in the order that they are stored on tape.
3. The analysis requires access to only a portion of each image and not the entire image.

With regard to the first assumption, while the images from a single sensor will undoubtedly span multiple tapes, it makes little sense to mix images from different sensors on the same tape. Analysis requiring access to multiple tapes (for data from either the same or different sensors) can use the techniques described in [5] to minimize tape switches in combination with the techniques described below. The second assumption requires that the reference pattern to the images be known in advance so that the references can be sorted into "tape order." In some cases, this order can be determined by examining the meta data associated with the data set. In a companion paper [6] we show how a new tape processing technique that we call "query pre-execution" can be used to automatically and accurately determine the reference pattern. The third assumption is based on the fact that satellite images are quite large (the size of an AVHRR image is about 40 megabytes) and scientists are frequently interested in only a small region of a large number of images and not each image in its entirety. The EOSDIS test bed [7] also places a strong emphasis on providing real-time dynamic subsetting of AVHRR images.

There are two alternative approaches for handling tape-based data sets. The first is to use a *Hierarchical Storage Manager (HSM)* such as the one marketed by EMASS [8]. Such systems almost always operate at the granularity of a file. That is, a whole file is the unit of migration from tertiary storage (i.e. tape) to secondary storage (disk) or memory. When such a system is used to store satellite images typically each image is stored in a separate file. Before an image can be processed, it must be transferred in its entirety from tape to disk or memory. While this approach will work well for certain applications, when only a portion of each image is needed it wastes tape bandwidth and staging disk capacity by transferring entire images.

An alternative to the use of an HSM is to add tertiary storage as an additional storage level to the database system. This approach is being pursued by the Sequoia [9] and Paradise [10] projects. Such an integrated approach extends tertiary storage beyond its normal role as an archive mechanism. With an integrated approach, the database query optimizer can be used to optimize accesses to tape so that complicated, ad-hoc requests for data on tertiary storage can be executed efficiently. In addition, the task of applying a complicated analysis to a particular region of interest on a large number of satellite images can be performed as a single query [11].

Integrating tertiary storage into a database system requires the use of a block-based scheme to move data between different layers of the storage hierarchy in the process of executing a query. While 8 KB is a typical block size for moving data between memory and disk, it is too small to use as the unit of transfer between tape and either memory or disk, especially when dealing with large raster images. The approach used instead by Postgres [5] and Paradise [10] is to partition each satellite image into a set of *tiles*. Tiles become the unit of transfer between tape and memory or disk while a smaller disk block (e.g. 8K bytes) is used to transfer data between disk and memory (i.e. the database system buffer pool). When a query references a portion of an image residing on tape, the meta data associated with the image is used to determine the minimum number of tiles necessary to satisfy the request. These tiles are first moved from tape to disk in tile-sized units and then from disk to memory in units of disk block size.[a]

[a] Actually data cannot be moved directly between two mechanical devices such as tape and disk without first passing through main memory Thus, a tile is first read by the tape controller into memory and then written to a tape block cache

This paper examines the impact of tile size on the time required to retrieve one or more partial (i.e. clipped) images residing on tape. The evaluation employs a simplified analytical model, a simple simulation study to verify the analytical model, and actual implementations using both a stand-alone program and the Paradise database system, extended to include support for tertiary storage [6]. Our results indicate that the careful selection of tile size can reduce the time required to clip a series of images residing on a single tape. In particular, we demonstrate that for tape drives such as the Quantum DLT-4000, a tile size in the range of 32 KB to 512 KB provides the best performance for a variety of image and clip region sizes.

The remainder of this paper is organized as follows. Section 2 describes the problem and the derivation of the analytical model. Section 3 describes the simulation experiments and analyzes their results. Section 4 examines the impact of tile size under a variety of experiments using Paradise as a test vehicle. Section 5 contains our conclusions and discusses future work.

2. Analytical Model

In this section we describe a simplified analytical model to compute the time to clip a single raster image by a rectangular region. We assume that the tape head is positioned at the beginning of the image. This model is then used to study the effect of tile size on the time to clip a raster image.

2.1 Problem Description

As discussed in Section 1, *tiling* partitions an image into smaller, rectangular pieces that preserve the spatial locality among adjacent pixels. Figure 1 depicts three alternative partitioning strategies of a single image and their corresponding linear layout on tape. The top left rectangle represents an untiled image. The middle and right rectangles in the top row show the same image partitioned into 4 tiles and 16 tiles, respectively. Once an image has been tiled, the tiles are stored sequentially on tape. The bottom portion of Figure 1 depicts how each of the three tiling alternatives is laid out on tape, assuming that tiles are placed on tape in a row-major fashion. While the choice of using a row-major layout may affect performance, we will demonstrate that tile size is the dominating factor, and not whether tiles are laid out on tape in a row-major or column-major order.

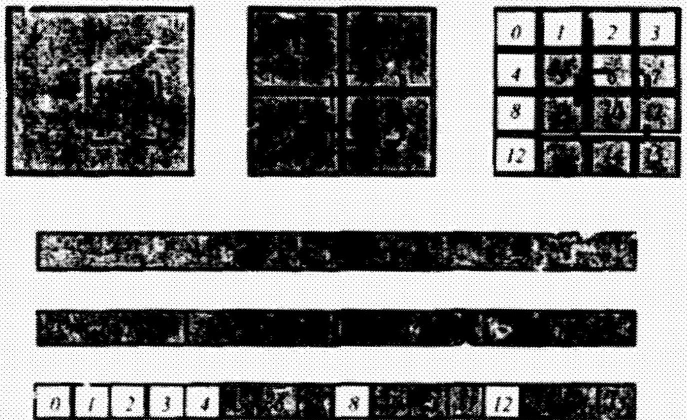


Figure 1: Single Image as 1, 4, 16 Tiles and their Linear Layout

on disk. While one could read tiles directly into the database buffer pool, this approach tends to flood the buffer pool with large amounts of information, forcing more valuable information out.

The dashed rectangle in Figure 1 corresponds to the portion of the image that the analyst wishes to examine – what we term the ***clip region***. The shaded area indicates which tiles must be retrieved from tertiary storage in order to satisfy the clip request. The impact of tiling is best illustrated in the comparison between the untiled image (top-left rectangle in Figure 1) and the 16-tile image (top-right rectangle in Figure 1). For the untiled image, the entire image must be read from tape in order to process the clip request. On the other hand for the image that has been partitioned into 16 tiles, only 9/16ths of the image (9 of the 16 tiles) must be read. However, the number of seek operations increases from 0 to 3 assuming that the tape head is initially positioned at the start of the image.

In general, the use of tiling can reduce the amount of data that must be transferred when clipping partial images. On the other hand, it can introduce additional seek operations between consecutive tile accesses. The total time spent in migrating the necessary parts of the image to memory or disk depends on the *tape seek speed, the tape transfer speed,* and the *seek startup cost*. The *seek startup cost* is a fixed overhead associated with each tape head movement while the tape seek speed indicates how fast the tape head can be advanced when not actually read/writing data. Together, these two parameters determine the random access latency on a single tape. In addition, there are a number of other factors that affect performance. For example, consider the image in Figure 1 that was partitioned into 4 tiles. For the clip request shown in Figure 1, this partition strategy has no advantage with respect to the number of seeks performed or the amount of data transferred compared to the untiled image. Other clip requests would have different results; for example, if the clip region was entirely contained inside tile *II* of the 4 tile image. In this case, the untiled image would incur no seeks but would transfer the entire image. The image tiled into 4 pieces would incur one seek (to the start of tile *II*) and would transfer _ of the image. The image tiled into 16 pieces would incur two seeks (one to transfer tiles 2 & 3 and a second to transfer tiles 6 &7) and would also transfer _ of the image. Thus, both the size and location of the clip region can affect the performance of the various tiling alternatives. In order to better understand the problem, we developed an analytical formula to model the average-case behavior.

2.2 Model Assumptions

In order to reduce the complexity of the problem, the analytical model makes the following assumptions:

1. Each tile is stored as a single *tape block*, which is the unit of migration from tape to memory.
2. The tape head is initially positioned at the beginning of the image.
3. Images are square (e.g. 5 by 5 or 9 by 9 tiles but not 4 by 6 tiles).
4. The shape of the clipping region is proportional to the image shape, and the clipping region is always contained inside the image boundary.
5. Clipped tiles are returned from tape in their original order stored on tape.

The first assumption eliminates the indirect effect of tape block size since multiple tiles could potentially be packed into a single tape block. We examine the effect of this assumption in Section 3. The second assumption allows us to concentrate on a single image without considering the residual impact from the previous tape head position. The third and fourth assumptions reduce the number of parameters that must be considered since variations in tile size and the shape of the clip region may effect performance. This will be discussed in Section 4. The final assumption minimizes the randomness between seeks within one clip operation.

2.3 Analytical Formula Derivation

We model the response time to migrate the tiles containing the clipped region from tape to memory as the sum of the time spent in the following four operations: ***Initial Seek, Intermediate Seeks, Tile T .ansfer***, and ***Total Szek Startup***. Table 1 contains all the symbols used in the analytical model. Figure 2 graphically illustrates the roles of a number of the symbols. Note that each tile has unit length 1, a is an integer, and the fraction part of b is modeled as $b - \lfloor b \rfloor$. From Figure 2 it is obvious that the number of tiles coverc.d or partially covered by the clip region varies depends on the particular position of the clip area. The probabilities of the various clip locations are calculated below. Finally, we analyze the time spent on each of the four operations to produce a formula that captures the average case behavıor.

Dr cription	Symbol	Comments
Image Size	M	KB
Single Tile Size	t	KB
Image Dimension	$a \times a$ tiles	$M = a^2 \cdot t$
Clip Dimension	$b \times b$ tiles	Clip Size $= b^2 \cdot t$
Clip Area/ Image Area	$1/c^2$	$a = c \cdot b$
Tape Seek Rate	S	KB/Sec
Startup Seek Cost	I	Sec
Tape Transfer rate	R	KB/Sec

Table 1: Parameters

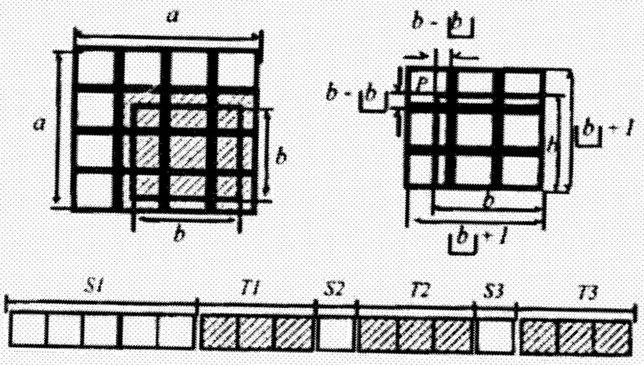


Figure 2. Target Clip Region

Four Clip Cases

As illustrated by Figure 3, there are four different ways that a region of constant area and shape can clip an image:

Case 1 -- touches $(\lfloor b \rfloor + 2)^2$ ti:cs;

Case 2 -- touches $(\lfloor b \rfloor + 2) \times (\lfloor b \rfloor + 1)$ tiles (elongated h rizo ntally);

Case 3 -- touches $(\lfloor b \rfloor + 1) \times (\lfloor b \rfloor + 2)$ tiles (elongated vertically);

Case 4 -- t($(\lfloor b \rfloor + 1)^2$ tiles.

In each case, the placement of the upper left corner (P) of the clip region is restricted to certain tiles in the image and certain regions within each of those tiles. The dark gray tiles in Figure 3 show the tiles where P can possibly reside, and the four regions in Figure 4 show where P can be placed within each of those tiles for each case. The probability for each of the four cases[iii] can be determined by examining the placement of P. Since the total area (A) containing P is $(a-b)^2$, the probability, Pb, for each case can be derived by considering the number of tiles (F) where P can be placed and the area (Af) within each of those tiles. Then, $Pb = F \cdot Af / A$.

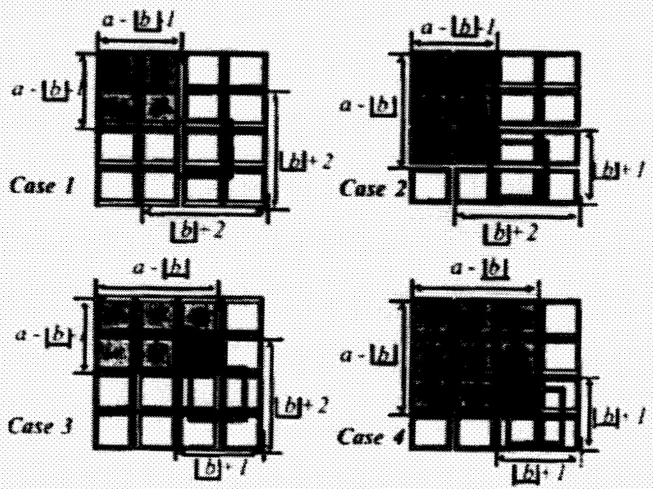


Figure 3: Four Clip Cases

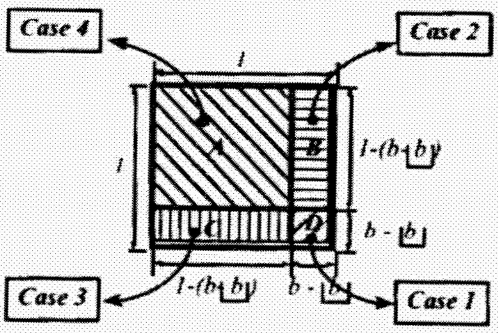


Figure 4: Regions in which P can reside for the different cases (within one tile)

The probabilities of each of the four cases occurring are specified below:

Case 1: $F = (a-\lfloor b\rfloor-1)^2$, $Af = (b-\lfloor b\rfloor)^2$ (area D), then

$Pb1 = (a-\lfloor b\rfloor-1)^2 \cdot (b-\lfloor b\rfloor)^2 / (a-b)^2$.

Case 2: $F = (a-\lfloor b\rfloor)\cdot(a-\lfloor b\rfloor-1)$, $Af = (b-\lfloor b\rfloor)\cdot(1-(b-\lfloor b\rfloor))$ (area B), then

$Pb2 = (a-\lfloor b\rfloor)\cdot(a-\lfloor b\rfloor-1)\cdot(b-\lfloor b\rfloor)\cdot(1-(b-\lfloor b\rfloor))/ (a-b)^2$.

[iii] Assuming a uniform distribution of clip sizes and locations

Case 3: $F = (a-\lfloor b\rfloor-1)\cdot(a-\lfloor b\rfloor)$, $Af = (1-(b-\lfloor b\rfloor))\cdot(b-\lfloor b\rfloor)$ (area *C*), then

$$Pb3 = (a-\lfloor b\rfloor-1)\cdot(a-\lfloor b\rfloor)\cdot(1-(b-\lfloor b\rfloor))\cdot(b-\lfloor b\rfloor)/\ (a-b)^2 .$$

Case 4: $F = (a-\lfloor b\rfloor)^2$, $Af = (1-(b-\lfloor b\rfloor))^2$ (area *A*), then

$$Pb4 = (a-\lfloor b\rfloor)^2\cdot(1-(b-\lfloor b\rfloor))^2/\ (a-b)^2 .$$

Initial Seek Time (*Tf*)

The *Initial Seek Time, **Tf**,* is the time required to move the tape head from the beginning of the image to the first tile touched by the clip region (indicated as S1 in Figure 2). From the analysis above, we know that the upper left corner (*P*) of the clip region can only reside in a restricted area depending on the various cases. Suppose this area is *M* by *N* at the upper left corner of the image, then P has an equal probability of being in any of the *X* by *Y* tiles in this region. Assume that P is in the tile defined by row i and column j ($0\le i< M,\ 0\le j< N$). Then, the number of tiles that must be skipped to reach P from the beginning of the image is $j+i\cdot a$. On average, $\left(\sum_{i=0}^{M-1}\sum_{j=0}^{N-1}(j+i\cdot a)\right)/(M\cdot N)$ tiles are skipped to reach the first tile covered by the clip. Hence, $Tf = \left(\left(\sum_{i=0}^{M-1}\sum_{j=0}^{N-1}(j+i\cdot a)\right)/(M\cdot N)\right)\cdot t/S$. Applying this to each of the four 4 cases in Figure 3, we get:

Case 1: $M = N = (a-\lfloor b\rfloor-1)$, then

$$Tf1 = \left(\left(\sum_{i=0}^{a-\lfloor b\rfloor-2}\sum_{j=0}^{a-\lfloor b\rfloor-2}(j+i\cdot a)\right)/(a-\lfloor b\rfloor-1)^2\right)\cdot t/S$$

Case 2: $M = (a-\lfloor b\rfloor-1)$, $N = (a-\lfloor b\rfloor)$, then

$$Tf3 = \left(\left(\sum_{i=0}^{a-\lfloor b\rfloor-2}\sum_{j=0}^{a-\lfloor b\rfloor-2}(j+i\cdot a)\right)/(a-\lfloor b\rfloor-1)^2\right)\cdot t/S$$

Case 3: $M = (a-\lfloor b\rfloor)$, $N = (a-\lfloor b\rfloor-1)$, then

$$Tf2 = \left(\left(\sum_{i=0}^{a-\lfloor b\rfloor-1}\sum_{j=0}^{a-\lfloor b\rfloor-2}(j+i\cdot a)\right)/((a-\lfloor b\rfloor)\cdot(a-\lfloor b\rfloor-1))\right)\cdot t/S$$

Case 4: $M = N = (a-\lfloor b\rfloor)$, then

$$Tf4 = \left(\left(\sum_{i=0}^{a-\lfloor b\rfloor-1}\sum_{j=0}^{a-\lfloor b\rfloor-1}(j+i\cdot a)\right)/(a-\lfloor b\rfloor)^2\right)\cdot t/S$$

Finally, the *Initial Tile Seek Time* is given by :

$$Tf = Pb1\cdot Tf1 + Pb2\cdot Tf2 + Pb3\cdot Tf3 + Pb4\cdot Tf4 .$$

Intermediate Seek Time (*Ti*)

The *Intermediate Seek Time, **Ti**,* is the total time spent seeking between transfers of contiguous sets of tiles contained in the clip region. For example, in Figure 2 above, after transferring the set of tiles in region T1, we must perform seek S2 before we can transfer the tiles in T2, and, after transferring the tiles in T2, we must perform seek S3 before transferring the tiles in T3. After the transfer of a set of contiguous tiles, the tape head must be moved to the next group of tiles affected by the clipping region. Assuming that the tiles touched by the clip region form a *X* by *Y* region, then the number of tiles to be skipped over after the initial seek is $(a-X)$, and there are $(Y-1)$ such movements. Thus, $Ti = (a-X)\cdot(Y-1)\cdot t/S$. Applying this to all the cases in Figure 3, we obtain:

Case 1: $X = Y = (\lfloor b\rfloor+2)$, then $Ti1 = (a-(\lfloor b\rfloor+2))\cdot(\lfloor b\rfloor+1)\cdot t/S$;

Case 2: $X = (\lfloor b \rfloor + 2)$, $Y = (\lfloor b \rfloor + 1)$, then $Ti2 = (a - (\lfloor b \rfloor + 2)) \cdot \lfloor b \rfloor \cdot t/S$;

Case 3: $X = (\lfloor b \rfloor + 1)$, $Y = (\lfloor b \rfloor + 2)$, then $Ti3 = (a - (\lfloor b \rfloor + 1)) \cdot (\lfloor b \rfloor + 1) \cdot t/S$;

Case 4: $X = Y = (\lfloor b \rfloor + 1)$, then $Ti4 = (a - (\lfloor b \rfloor + 1)) \cdot \lfloor b \rfloor \cdot t/S$.

Finally, the ***Intermediate Seek Time*** is:

$$Ti = Pb1 \cdot Ti1 + Pb2 \cdot Ti2 + Pb3 \cdot Ti3 + Pb4 \cdot Ti4 .$$

Transfer Time (*Tr*)

The *Tile Transfer Time*, ***Tr***, is the total time spent transferring tiles in the clipped region to memory (***T1***, *T2* and *T3* in Figure 2). Based on the same assumptions made when calculating the ***Intermediate Seek Time***, $X \cdot Y$ tiles must be transferred . This leads to: $Tr = X \cdot Y \cdot t/R$. Again, analyzing the different cases in Figure 3 using this formula, we obtain:

Case 1: $X = Y = (\lfloor b \rfloor + 2)$, then $Tr1 = (\lfloor b \rfloor + 2)^2 \cdot t/R$;

Case 2: $X = (\lfloor b \rfloor + 2)$, $Y = (\lfloor b \rfloor + 1)$, then $Tr2 = (\lfloor b \rfloor + 2) \cdot (\lfloor b \rfloor + 1) \cdot t/R$;

Case 3: $X = (\lfloor b \rfloor + 1)$, $Y = (\lfloor b \rfloor + 2)$, then $Tr3 = (\lfloor b \rfloor + 1) \cdot (\lfloor b \rfloor + 2) \cdot t/R$;

Case 4: $X = Y = (\lfloor b \rfloor + 1)$, then $Tr4 = (\lfloor b \rfloor + 1)^2 \cdot t/R$.

The overall *Transfer Time* is:

$$Tr = Pb1 \cdot Tr1 + Pb2 \cdot Tr2 + Pb3 \cdot Tr3 + Pb4 \cdot Tr4 .$$

Seek Startup Time (*Ts*)

Each tape seek is associated with a fixed startup overhead which we model with the variable ***Seek Startup Time***, ***Ts***. This overhead only depends on the number of seeks performed, and not the size of each seek operation. Using the same $X \cdot Y$ region as above, then Y seeks are needed for each clip and $Ts = Y \cdot I$. Breaking this into the different cases yields:

Case 1: $Y = (\lfloor b \rfloor + 2)$, then $Ts1 = (\lfloor b \rfloor + 2) \cdot I$;

Case 2: $Y = (\lfloor b \rfloor + 1)$, then $Ts2 = (\lfloor b \rfloor + 1) \cdot I$;

Case 3: $Y = (\lfloor b \rfloor + 2)$, then $Ts3 = (\lfloor b \rfloor + 2) \cdot I$;

Case 4: $Y = (\lfloor b \rfloor + 1)$, then $Ts4 = (\lfloor b \rfloor + 1)\ I$.

Finally, $Ts = Pb1 \cdot Ts1 + Pb2 \cdot sr2 + Pb3 \cdot Ts3 + Pb4 \cdot Ts4$.

Total Response Time

The *Total Response Time*, *T*, is the sum of the four terms above: $T = Tf + Ti + Tr + Ts = f(a, b, S, R, I)$" To make our results easier to interpret, we substitute image

" The whole formula of T, even after simplification, is too complicated to present Instead, we will show its parameters and present them graphically size M, tile size t, and clip selectivity c for a and b ($c = a/b$ and $M = a^2 \cdot t$). Now $T = f'(M, t, c, S, R, I)$.

For analysis purposes, we fix the image size M and the clip region size ($1/c^2$ of an image). Under these conditions, the formula reveals the following interesting properties: as the tile size increases, the seek time (including Tf, Ti, and Ts) decreases while the transfer time (Tr) increases. The combination of these two opposite effects makes the response time a complex function of the tile size.

Analytical Model Analysis

To help understand the implications of the response time formula T derived above, we next evaluate it for a variety of parameters, plotting the response time as a function of the tile size. The values for the various parameters are listed in Table 2. The image size is varied from 8 MB to 128 MB and the clip selectivity from 1/4 to 1/256 of the image. The tape-related parameters (S, R, I) are selected based on the Quantum DLT-4000 tape drive [2] with compression turned off.

Parameter	Values Evaluated
M	8 MB, 32 MB, 128 MB
c	2, 4, 8, 16
S'	2,048 KB/Sec
R	1,356 KB/Sec
I	0.1 (Sec)

Table 2: Selected values for parameters

The response times for a variety of image and clip size combinations along with the gain relative to always fetching an entire image are shown in Figures 5 to 7. These results indicate that, as the tile size is increased, the response time first decreases slightly before increasing and that tiling provides a significant benefit compared to fetching an entire image.

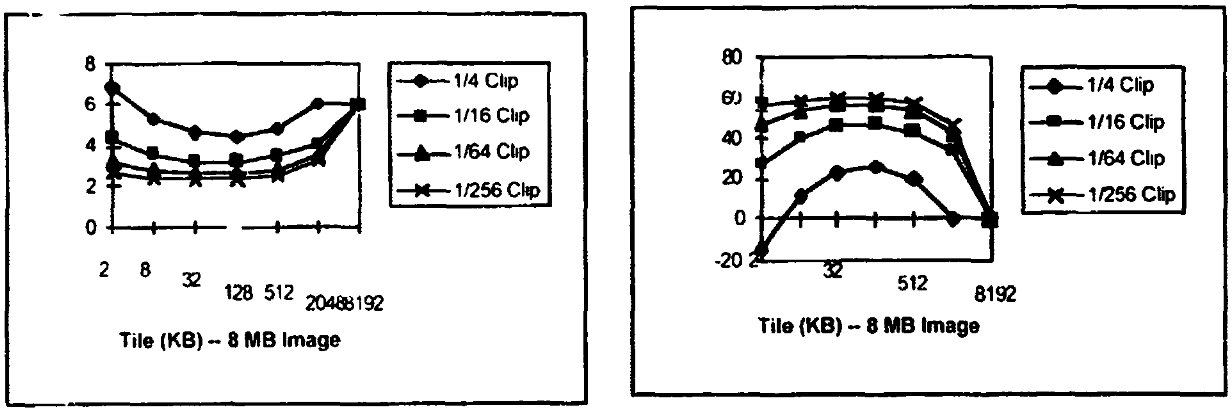


Figure 5: Response Time and Performance Gain over Entire Image Transfer for 8 MB Images

' Because DLT uses serpentine tapes the seek time is not a linear function of the seek distance. For the DLT4000 we observed a maximum seek time of 180 between two random blocks If the seek is within a single track the seek time is close to a linear function [12] contains an accurate model of seek time for DLT drives

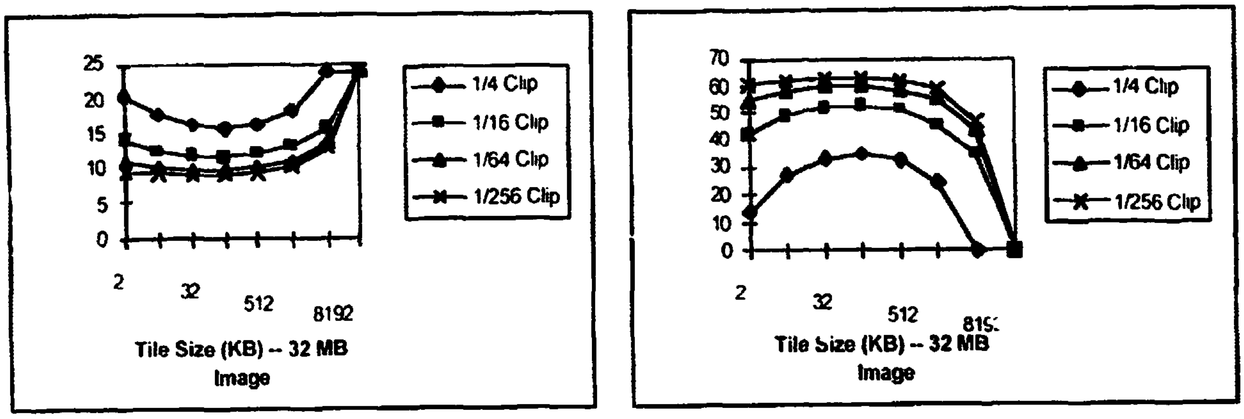


Figure 6: Response Time and Performance Gain over Entire Image Transfer for 32 MB Images

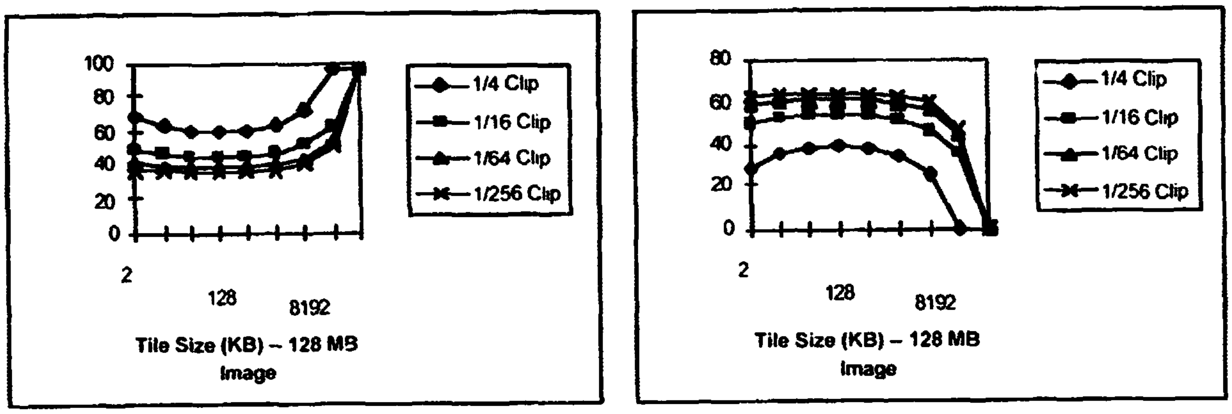


Figure 7: Response Time and Performance Gain over Entire Image Transfer for 128 MB Images

To help understand these results, Figure 8 decomposes the time required to clip 1/16th of a 32 MB image into its three component parts: *Seek, Transfer, and Seek Overhead*. While both the seek time (*Tf+Ti*) and seek overhead (*Ts*) decrease as the tile size increases, the transfer time (*Tr*) increases at a faster rate and eventually dominates the response time on the right side of the graph. The effect of tape seeks is best illustrated for tile sizes less than 512 KB where the reduction in seek time, resulting from both fewer and shorter seeks, offsets the increase in transfer time as the tile size increases.

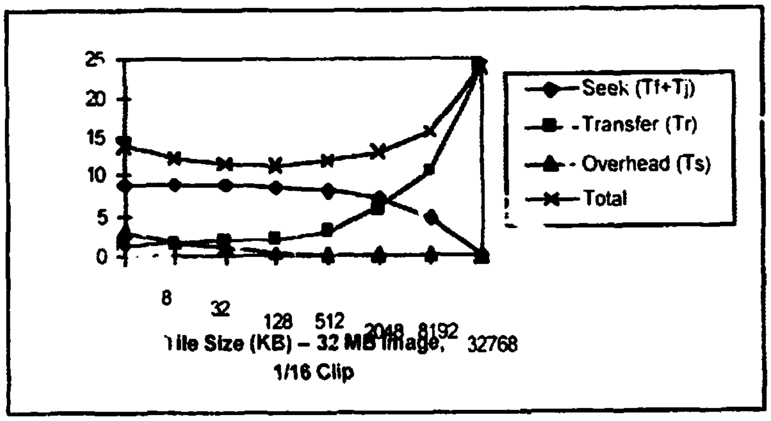


Figure 8: Breakdown of Response Time

Based on these figures, for the DLT drive a tile size in the region between 32 KB and 512 KB provides the best perform[illegible] for a variety of image and clip sizes. In general, using appropriateiy sized tiles provides between a 2[illegible] [illegible] 70% improvement compared to fetching the entire image.

3. Simulation Experiments

From the analytical model described in Section 2 it is clear that there is a trade-off between decreasing the seek time and increasing the transfer time as the tile size is increased. The analytical model, however, was based on a number of simplifying assumptions so that it would be easy to derive and analyze. There may be conditions for which the results obtained using it are not valid. To verify its accuracy, we developed a simulation model that we use in this section to explore a broader range of test conditions.

3.1 Simulation Configuration

As with the analytical model, the simulation model focuses on the response time for transferring data from tape to memory. The simulator consists of three components: a *work load generator* for generating clip requests of various sizes, shapes, and locations, an *image clipper* for generating the sequence of tile accesses required to satisfy a particular clip operation, and a *simulated block-based tertiary storage manager*. Given a request for a tile, the tertiary storage manager simulator first converts the tile number to a tape-block address. Next, it simulates moving the tape head from its current position to the desired tape block and transferring the block from tape to memory. The tape parameters from Table 2 are used to model a Quantum DLT-4000 tape drive. Each data point represents the average response time of 1000 clip requests at different locations. There are several major differences between the analytical and the simulation models. First, assumptions 3 and 4 from Section 2.1 are relaxed: images no longer must be square, and the clip shape need not be proportional to the shape of the image. In addition, the tape block size can be different than the tile size. That is, multiple tiles can be packed into a single tape block.

3.2 Analytical Model Verification

To verify the analytical model, using the simulation model we repeated the experiments presented in Figures 5, 6, and 7.. To simplify the comparison of the two sets of results, we show the relative differences in response times from the two models in Figures 9, 10, and 11. The difference is never greater than 4%. In general, the response times generated by the simulation model are slightly lower than the analytical model (i.e. a negative difference) for larger tile sizes and slightly higher for extremely small tile sizes. This is due to the randomness in selecting clip regions. This error margin is acceptable given the number of tests used. Based on these results, it is clear that the analytical model is quite accurate given the assumptions it is based on.

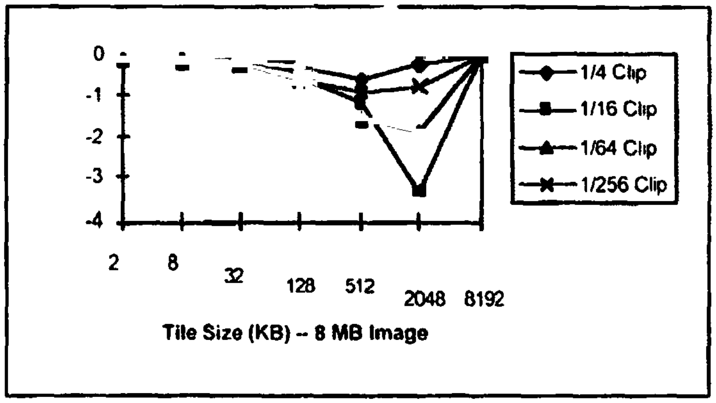


Figure 9: Relative Difference between Analytical and Simulation Result -- 8 MB Image

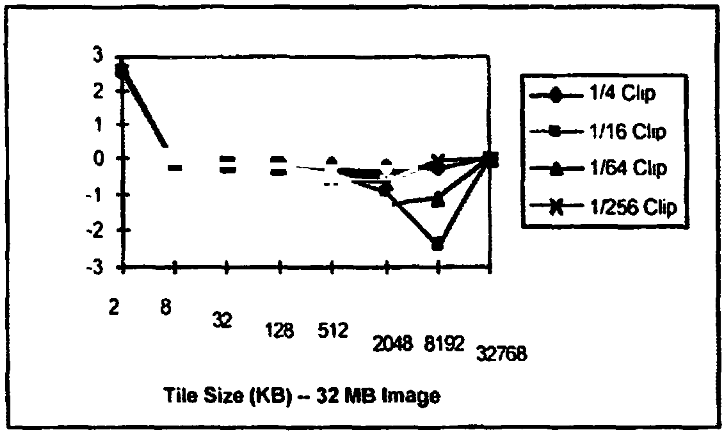


Figure 10: Relative Difference between Analytical and Simulation Result -- 32 MB Image

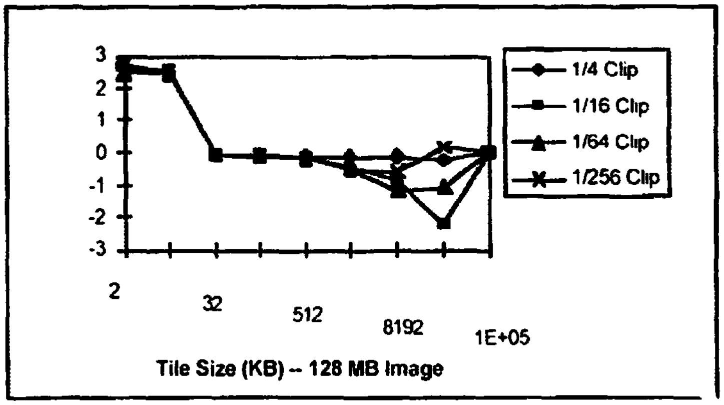


Figure 11: Relative Difference between Analytical and Simulation Result – 128 MB Image

3.3 Tape Block Size vs. Tile Size

As mentioned in Section 3.1, the simulation model allows us to examine the impact of varying the tile size and the tape block size independently. Using a tape block smaller than a tile will not significantly affect performance since each tile is then stored as a set of contiguous tape blocks. On the other hand, when the size of a tape block is larger than the size of a tile, multiple tiles can be packed into a single tape block. This can affect response time as fetching one tile is likely to result in reading tiles that are not covered by the clip operation. Figure 12 contains three curves corresponding to three different *Tape Block Size/Tile Size* ratios. Clearly, using a tape block size larger than the tile size causes performance to degrade. In general, packing multiple tiles into a single tape block has an effect similar to increasing the tile size. However, since packing tiles into a bigger tape block is done in row-major order, when too many tiles are packed into a tape block, the over-all shape of all the tiles in a tape blo^k is no longer rectangular. This irregular shape can further degrade perfor.nance. These results indicate that it is the best to use the same tile and tape block size. Consequently, all subsequent results presented in this paper use the same block size and tile size.

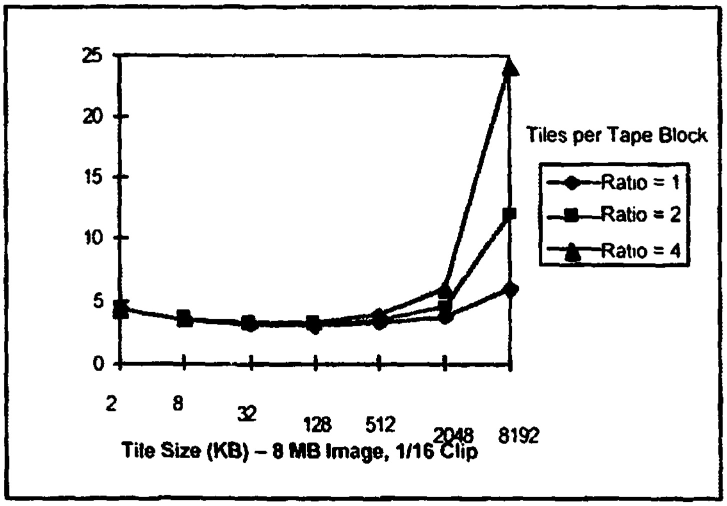


Figure 12: Effect of Tape Block Size vs. Tile Size

3.4 Alternative Clip Shapes

One of the assumptions made for the analytical model was that the shapes of the clip region and image had to be the same. Clip regions with the same area bu: different shapes can also affect performance. To investigate this effect, we experimented with three different clip shapes: *Long*, *Wide*, and *Square*. *Lung* is a thin rectangular shape whose height is four times its width; *Wide* is the *Long* shape rotated 90 degrees; *Square* is proportional to the image shape (which has been used in all previous experiments). Figure 13 shows the results from this experiment and illustrates that the different shapes indeed have different response times. It is interesting to notice that the "*Wide*" curve is consistently below the "*Long*" curve. This is caused by the row-major linear ordering of tiles on tape. Such a layout helps "*Wide*" clips reduce the number of tiles that must be sought over. The "*Square*" clips is relatively close to the average behavior, and in most ca:es, is jus' between the "*Wide*" and "*Long*" curves. However, when the tile size is much larger than the clip size (e.g. 2,048 KB), the "*Square*" shape had lower responses than the "*Wide*" shape becaus: it is less likely to overlap more than one tile. A key result from this set of experiments is that, regardless of the clip shape, the best response time occurs when the tile size is between 32 KB and 512 KB.

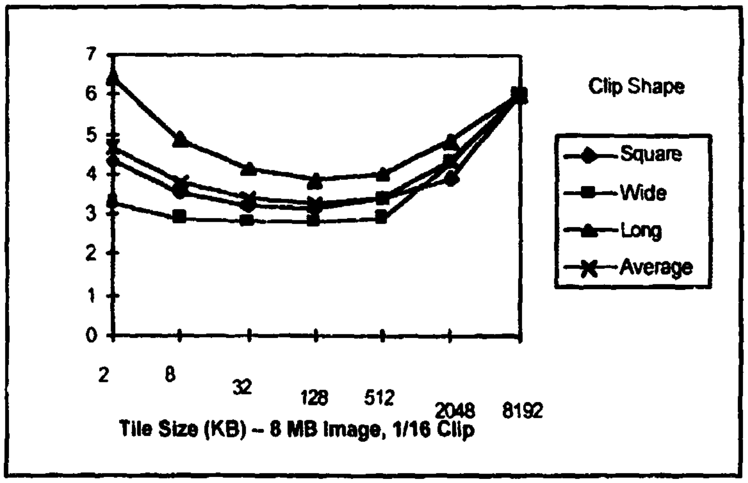


Figure 13: Alternative Clip Shapes, Simulation

4. Implementation Results

To verify the results obtained both analytically and through simulation, we next repeat some of the experiments using the following two configurations: an application-level program that accesses raster images directly on tape, and Paradise -- a DBMS extended to handle data on tape.

4.1 Configuration

The application program is a stand-alone program that is capable of accessing tiled raster image stored on tape. This corresponds to a typical scientific application accessing a tape-resident data set. Paradise [10] is an object-relational database system developed at University of Wisconsin - Madison, which is capable of handling both spatial data (vector-based) and images (raster-based) efficiently. For raster images, Paradise combines tiling with compression to maximize performance. In a separate project [6], Paradise was extended to include support for tertiary storage. Both the application-level program and Paradise share the same block-based DLT device driver and the raster image clip code. Thus, the amount of time each spends doing tape I/Os and clipping raster images in memory is comparable. However, the application program directly transfers data from tape to user space while Paradise first stages tape blocks on a staging disk. The experiments using Paradise represents the end-to-end performance in a tertiary database system.

Experiments in both configurations were conducted on a DEC CELEBRIS XL590 (Pentium 90MHz) with 64 MB memory. The tertiary device used is the Quantum DLT4000 tape drive. The block-based tape driver breaks each tape blocks larger than 32 KB into multiple 32 KB physical chunks before performing the actual tape I/O via an ioctl call. This scheme is used to simplify the mapping between physical and logical block addresses[vi]. A single logical tape block is mapped to multiple contiguous 32 KB physical records on tape. For Paradise, all queries were executed with cold buffer pools (for both main memory and disk cache). Due to the high cost of running actual tests, we had to cut down the number of randomly generated clip shapes from 1000 (used in simulation experiments) to 20.

4.2 Alternative Clip Shapes -- Application Program

Figure 14 displays the results obtained using the application program for the same set of experiments presented in Figure 13. Although there are some discrepancies at the ends of the curves, the general trends are the same. The *Wide* shape benefits the most from the layout of the tiles and, thus, has consistently lower response times. The *Long* shape tends to seek across more tiles than the other cases and hence has the highest response times. Finally, the *Square* shape is close to the average case. Note that the response times for 2 and 8 KB tile sizes are lower than the values predicted by the simulation model. This might be explained by the DLT tape drive's internal read-ahead cache. While no literature was found in the product manual on how it works, we suspect that tape head does not stop at the end of the last read but may actually read ahead to some location further down the track, thus hiding some of the seek latency. With small tile sizes it is more likely that most of the intermediate seeks can be absorbed by this read-ahead mechanism. This can also explain why the different clip shapes are closer together with the application-level program. Another discrepancy between the two sets of results is the difference in absolute times. This is caused by several factors that are not captured by the simulation model, including post-tape processing time (ie. the clip time in memory), a smaller population of random samples, and the overhead of decomposing logical tape blocks larger than 32 KB into multiple 32 KB physical chunk transfers.

[vi] The maximum system I/O buffer size for a single read/write request without being translated into multiple kernel level I/O calls is 63 KB Since each kernel-level tape write call generates a separate record on tape it is easier to manage smaller physical chunks and provide a large (variable sized) record interface for the higher level

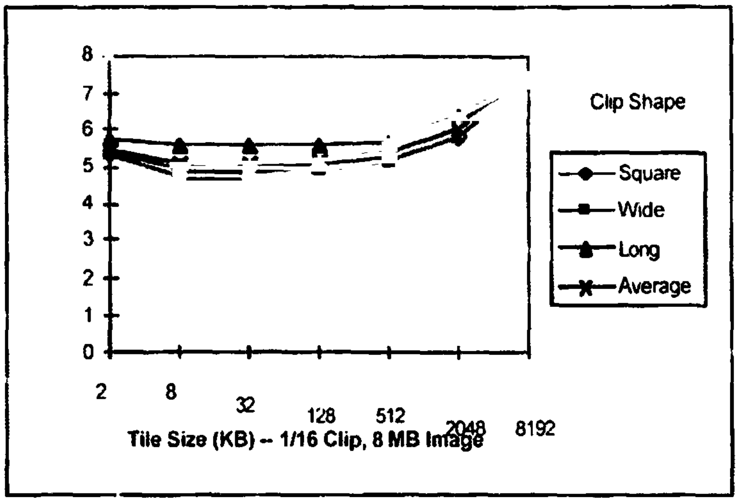


Figure 14: Alternative Clip Shape, Application Program

4.3 Comparison among all Models

Figure 15 shows the response times obtained for clipping 1/16th of a 32 MB Image under all four configurations (Analytical, Simulation, Application and Paradise)[vi]. Again, all the curves illustrate the same trend: as the tile size is increased beyond 32 KB the response time increases. As explained above, the difference between the simulation and application-level results is mainly due to the extra overhead of having to break large tape I/Os into multiple 32 KB chunks. In addition, it appears that there are additional fixed startup costs that are not captured by the analytical or simulation models.

There are a number of causes for the difference between Paradise and the application-level program. First, as a general DBMS engine, Paradise assumes that tape blocks requested by one query might be useful for other, concurrently executing queries. Thus, tape blocks are staged first on disk and then copied into Paradise's buffer pool in main memory on demand. Second, Paradise incurs some extra processing overhead while managing large objects like tiles. Nevertheless, the impact of tile size on performance is apparent.

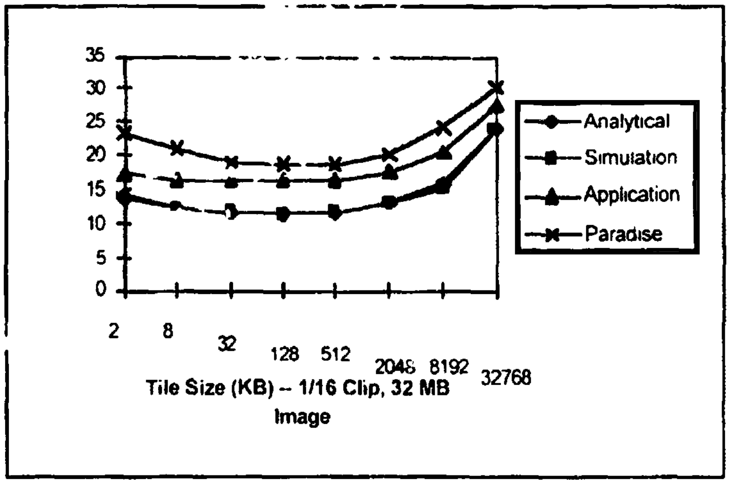


Figure 15 All Configurations

[vi] The curves for analytical and simulation models are almost the same as discussed in Section 3.2

4.5 Processing of Multiple Images

To determine the effect of clipping a large number of images, we next conducted an experiment in which fifty 32 MB images were clipped by a fixed region whose area was 1/16th the size of the image boundary. Figure 16 and 17 show the results from this test on the application-level program and Paradise. While both figures show a small advantage of using a smaller tile size, the most promising result is that Paradise's performance for this test is within 5-10% of the hand-coded application program.

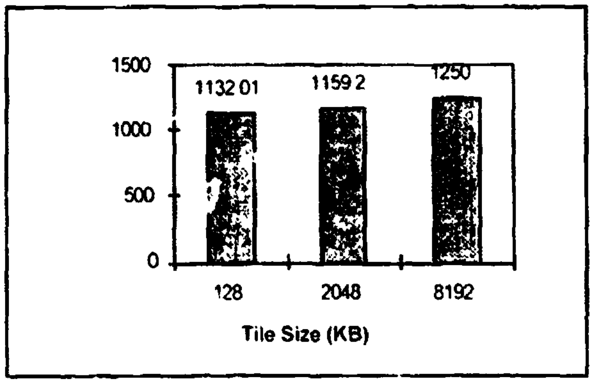


Figure 16: Time to Clip 1/16th of Fifty 32 MB Images – Application

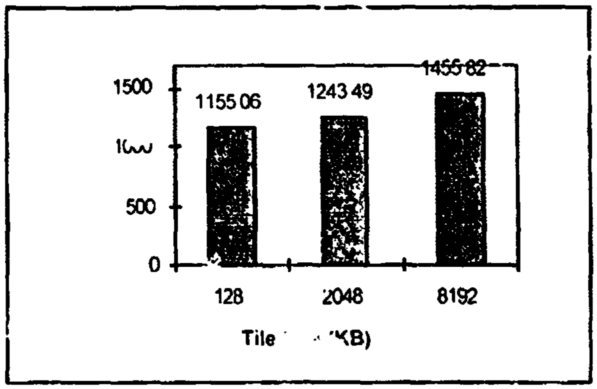


Figure 17: Time to Clip 1/16th of Fifty 32 MB Images -- Paradise

5. Conclusions and Future Work

In this paper we described a simple analytical model to study the impact of tile size on the performance of retrieving partial satellite images from tape. Using this analytical model as well as a simulation model and two actual implementations (Paradise and a hand-coded application program) we demonstrated that the tile size has a complex effect on the cost of executing queries involving clips of partial satellite images stored on tape. Our results indicate that, for the Quantum DLT 4000 tape drive, a tile size between 32 KB and 512 KB provides the best performance for a wide range of image and clip region sizes. These results are very encouraging as smaller tile sizes simplify space management on both the disk cache and buffer pool while providing good performance.

While this study assumed that the images being processed are stored sequentially on tape, in a companion study [6] we describe a set of new techniques that are capable of reordering tape accesses from complex object-relational queries in order to satisfy this constraint. Although this paper dealt only with 2D images, we believe that our results also apply to 3D images as well as arbitrary N-dimensional arrays stored on tape. As dimensionality is increased, the number of seeks and the seek distances can increase exponentially. We expect that in these cases the tile size will have even a larger impact.

References

[1] B. Kobler and J. Berbert, "NASA Earth Observing System Data Information System (EOSDIS)," Digest of Papers 11th IEEE Symposium on Mass Storage Systems, Los Alamitos, 1991.

[2] Quantum Corporation, "DLT-4000 Product Manual," 1995.

[3] M. Carey, L. Haas and M Livny, "Tapes Hold Data Too: Challenges of Tuples on Tertiary Store," Proceedings of the 1993 SIGMOD Conference, May, 1993

[4] J. Gray, "MOX, GOX and SCANS: The Way to Measure an Archive," EOSDIS V0 Experience, June, 1994.

[5] S. Sarawagi "Efficient Organization for Multidimensional Arrays," Proceedings of the 1994 IEEE Data Engineering Conference, February, 1994.

[6] J. Yu and D. DeWitt "Query Pre-Execution and Batching: A Two-Pronged Approach to the Efficient Processing of Tape-Resident Data Sets." Submitted for publication, February, 1996.

[7] W. Emery, T. Kelley, J. Dozier and P. Rotar. "On-line Access to Weather Satellite Imagery and Image Manipulation Software," Bulletin of the American Meteorological Society, January, 1995.

[8] R. Herring and L. Tefend. "Volume Serving and Media Management in a Networked, Distributed Client/Server Environment," Proceedings of the 3rd Goddard Conference on Mass Storage Systems and Technologies, NASA Con. Publication 3263. 1993.

[9] F. Davis, W. Farrel, J. Gray, et al. "The Sequoia Alternative Architecture Study for EOSDIS," NASA Study Report, October, 1994.

[10] D. DeWitt, N. Kabra, J. Luo, J. Patel and J. Yu. "Client Server Paradise," Proceedings of the 20th VLDB Conference, September, 1994.

[11] M. Stonebraker, J. Frew, K. Gardels and J. Meredith. "The SEQUOIA 2000 Storage Benchmark," Proceedings of the 1993 SIGMOD Conference, May, 1993.

[12] B. Hillyer and Avi Silbertschatz. "On the Modeling and Performance Characteristics of a Serpentine Tape Drive," Proceedings of the 1996 SIGMETRICS Conference, May, 1996.

NEXT
DOCUMENT

Evolving Requirements for Magnetic Tape Data Storage Systems

John J. Gniewek
IBM Corporation
9000 S. Rita Road
Tucson, Arizona 85744
jgniewek@vnet.ibm.com
+1-520-799-2390
Fax: +1-520-799-3665

Introduction

Magnetic tape data storage systems have evolved in an environment where the major applications have been back-up/restore, disaster recovery, and long term archive. Coincident with the rapidly improving price-performance of disk storage systems, the prime requirements for tape storage systems have remained: 1) low cost per MB, and 2) a data rate balanced to the remaining system components. Little emphasis was given to configuring the technology components to optimize retrieval of the stored data. Emerging new applications such as network-attached HSM, and digital libraries, place additional emphasis and requirements on the retrieval of the stored data. It is therefore desirable to consider the system to be defined by both STorage And Retrieval System (STARS) requirements. It is possible to provide comparative performance analyses of different STARS by incorporating parameters related to A) device characteristics, and B) application characteristics in combination with queuing theory analysis. Results of these analyses are presented here in the form of response time as a function of system configuration for two different types of devices and for a variety of applications.

STARS (STorage And Retrieval System)

A) Performance Model

Some simplifying assumptions will be necessary to be able to provide comparative performance analyses for two different tape devices. A list of the required input parameters for both A) Hardware characteristics and B) Application characteristics is given in Table 1. The output of the model will be given as an average response time for various combinations of these parameters. In order to directly compare device types only, the assumption is made that both devices are serviced by a robot with identical characteristics, i.e. a fixed robot average cycle time with no allowance for queuing delays at the robot level. Device queuing delays are calculated using the methodology in reference [1]. For this analysis* an M/M/C queue is used.

* The model and graphical output are developed using MathCad® 6.0 (®-MathSoft...)

Table 1
Input Parameters Required for Performance Model

Hardware Characteristics			Application Characteristics		
Device data rate	(MB/sec)	(D)	Service Request Rate	(#/Hr)	(I)
Cartridge Capacity	(MB)	(C)	Object Size	(MB)	(O)
Recording Density	(MB/m)	(K)	Library Size	(MB)	(L)
Search/Rewind Velocity	(M/sec)	(V)	Random Retrieval Factor	(#)	(A)
Load Time	(sec)	(LD)	Drive Number	(#)	(N)
Unload Time	(sec)	(ULD)			
Robot Cycle Time	(sec)	(AS)			

It is necessary to distinguish between: a) a base cycle time which is a necessary input for calculating queuing delay times, and b) a base service time which is defined as the response time in the absence of any queuing delays. Additionally, service times are defined for the cases where: 1) the required cartridge is already mounted in a drive, or 2) the cartridge is in a library bin and must be transported to and loaded into a drive. The assumed sequences of operations for both cycle time and service time are listed in Table 2. These sequences assume that there is no preemptive unloading of cartridges upon completion of a service request. It is additionally assumed that at the completion of each service request, there is a rewind to the starting point prior to servicing the next request.

Table 2
Sequence of Operations for Cycle Time and Service Time

Cycle Time		Service Time	
Unmounted	Mounted	Unmounted	Mounted
Unload Cartridge		Unload Cartridge	
Robot Put Cartridge		Robot Put Cartridge	
Robot Get Cartridge		Robot Get Cartridge	
Load Cartridge		Load Cartridge	
Search	Search	Search	Search
Read	Read	Read	Read
Rewind	Rewind		

In order to calculate an average system response time, it is necessary to be able to estimate the probability, P, that the next incoming request will be serviced by an already mounted cartridge as opposed to requiring a robotic Put and Get operation. The expression given in Equation (1) is used to estimate this probability as a function of the library size, the cartridge capacity, the number of drives in the robotic system, and an application dependent adjustable parameter, A, which characterizes the degree to which the requested objects are randomized within the library cartridges [2].

$$P = \frac{1}{100}\left\{100^A - \left(100 - 100\left(\frac{N \cdot C}{L}\right)\right)^A\right\}^{1/A} \tag{1}$$

where: A = application dependent random factor
C = cartridge capacity (MB)
L = library size (MB)
N = number of drives in library
P = probability of next request being serviced by a mounted volume

Figure 1 shows P as a function of the cumulative percent of volumes in the library ordered by activity from highest to lowest. Equation (1) is derived empirically, however its relevance to a statistical analysis is covered in reference [2].

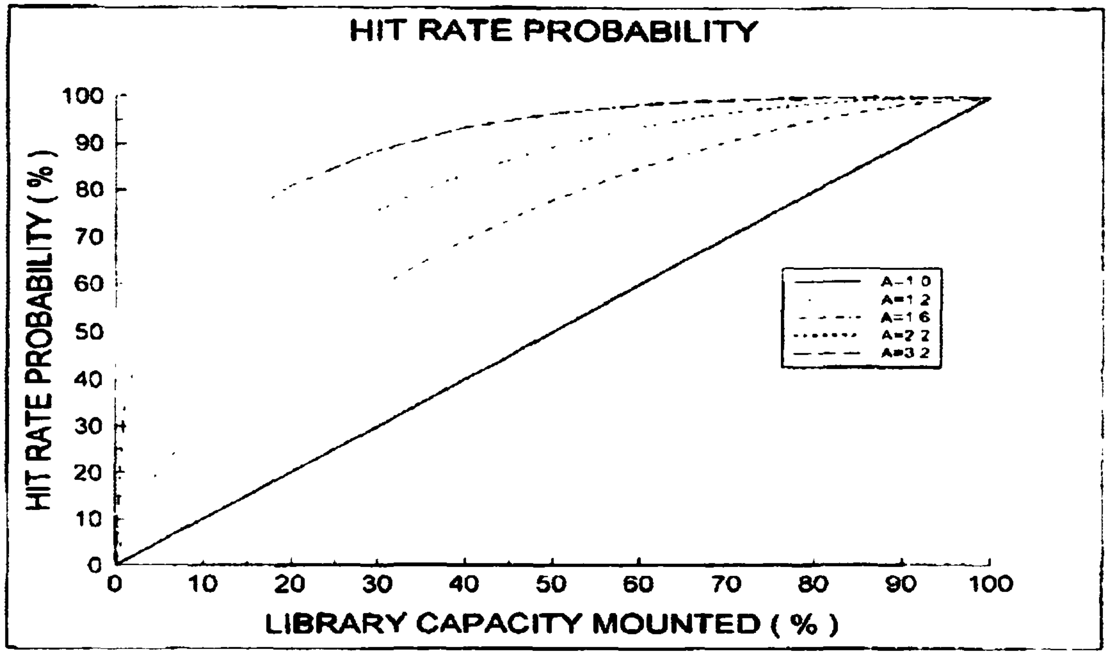


Figure 1. Probability, P, of next request being serviced by a mounted volume as a function of percent of library capacity mounted. Plotted from Equation (1) for A = 1.0, 1.2, 1.6, 2.2, and 3.2.

The output of the model is presented in terms of an average System Response Time (SRT). (to complete the service request) as a function of various combinations of the other input parameters. Intermediate output parameters include, in addition to P, an average cycle time, CT, (appropriately weighted by the fraction of requests serviced by mounted and unmounted volumes), a drive utilization factor, U, and an average device queuing delay, QD.

The expression for the queuing delay time for multiple servers in an M/M/C queue is adapted from reference [3] as:

$$QD = CT\left(\frac{(NU)^N \cdot P_o}{N!\ N(1-U)^2}\right) \tag{2}$$

$$\text{where } P_o = \left[\frac{(NU)^N}{N!(1-U)} + \sum_{i=0}^{N-1}\frac{(NU)^i}{i\ !}\right]^{-1} \tag{2a}$$

The M/M/C queue designates an expone.. ial interarrival time request distribution and an exponential service time distribution with means designated by the chosen values for the input parameters.

Average cycle times, utilization factors, and service times are calculated in a straight forward manner from the parameters designated in Tables 2 and 3. The average system response time is the sum of the average service time and the average queuing delay time.

Table 3
Hardware Characteristics of Two Prototype Devices

Paran.eter			Device I	Device II
Data Rate	(D)	(MB/sec)	9	2.2
Cartridge Capacity	(C)	(MB)	10000	5000
Recording Density	(K)	(MB/m)	34	34
Velocity	(V)	(m/sec)	5	10*
Load Time	(LD)	(sec)	15	2
Unload Time	(ULD)	(sec)	11	2
Robot Cycle Time	(AS)	(sec)	fixed**	fixed**

* Actual Search Velocity 5 m/sec. Midpoint cartridge load translates this parameter into an effective 10 m/sec consistent with the model formulation.

** Assumed common robotic device to highlight contrasts in device characteristics. See Figures for values used.

B) Hardware Characteristics

A description of two different types of prototype devices that could be developed from advanced technology recording components has previously been presented [4]. Their characteristics are designed to complement each other for different application requirements. For the purpose of this performance analysis, devices are assumed with characteristics similar to those previously described. The specific values used in the comparative performance analysis are given in Table 3. STARS are defined by specifying:

a) a robot syste n with a fixed cycle time[**], b) the type of device (I or II) defined in Table 3, c) the number of devices in the library system, and d) the library capacity (in MB).

Analysis

In addition to the large number of STARS hardware variables, the analysis must accommodate the application variables; I, the service request rate, O, the object size being retrieved, and A, the random retrieval factor. In the first analysis, the system response time is calculated as a function of the service request rate for both types of devices and with the number of drives as an independent parameter. All other parameters are fixed. This is done for both a large object size, 300 MB, and a small object size, 1 MB, to highlight the differences in device characteristics.

A helpful way to assess appropriate operating domains for the different types of devices is to plot the average system response time as isochrons over the domain space of object size, O, and service request rate, I, with the remaining parameters fixed. Most of the calculated results are presented in this format. System configuration variables such as the number of drives in the library represent another dimension and these data can be represented parametrically in separate, individual plots.

Results

A. System Response Time as a Function of Reques! Rate

Figures 2A and 2B show the system response time as a function of request rate for the two different types of devices in system configurations of 1, 2, or 4 drives, and for a library capacity of 10 TB, with a value of the randomness factor, A, equal to 3.2 Figure 2A plots results for a 1 MB object size and Figure 2B is calculated for 0 = 300 MB. Figures 2C and 2D repeat these calculations with all values the same except the library capacity is set to 0.5 TB. This has the effect of modifying the mounted/unmounted service request ratio via the probability function given in Figure 1. The effect this has on system respo: e time is a complex function of device characteristics, robotic cycle time and the specific application parameters. The data in Figure 2 illustrat one possible scenario. In general the response time is improved at smaller library capacities as a result of a higher probability of the request being satisfied by an already mounted volume, thereby eliminating the need to invoke a robotic move to satisfy that parameter request.

[**] The performance model presented here does not provide for robotic queuing delays in order to emphasize the different characteristics of the two types of devices

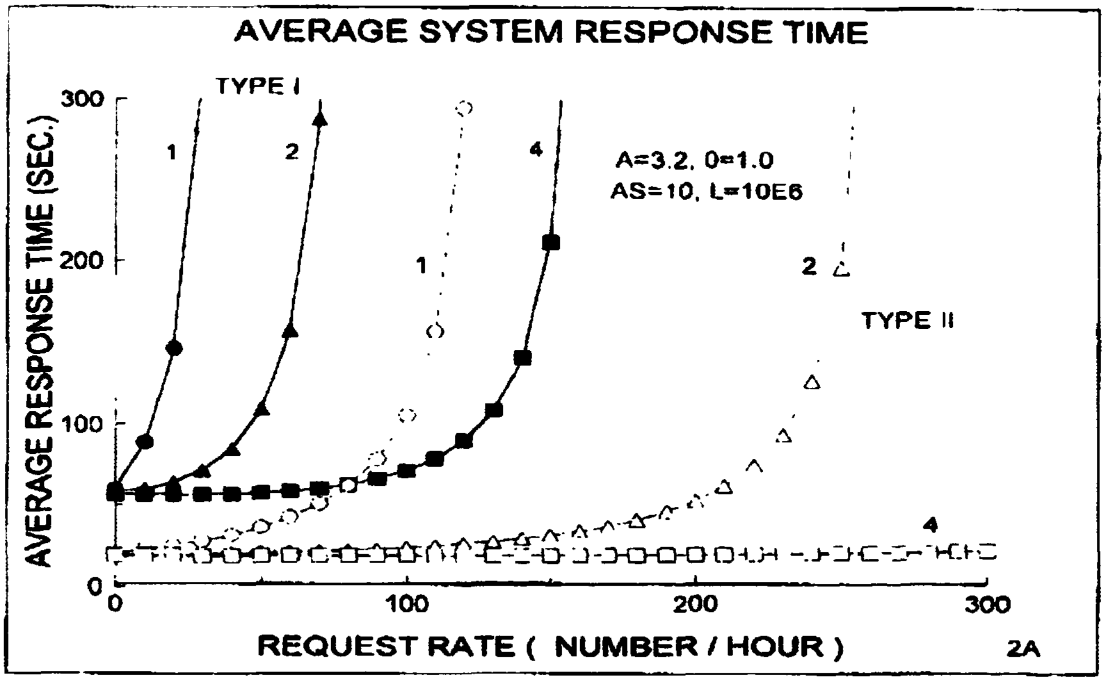


Figure 2A. Average system response time, SRT, as a function of service request rate (#/hour), for N = 1, 2, 4 drives of Device Type I and Device Type II. A = 3.2, AS = 10 secunds, L = 10 TB, O = 1 MB.

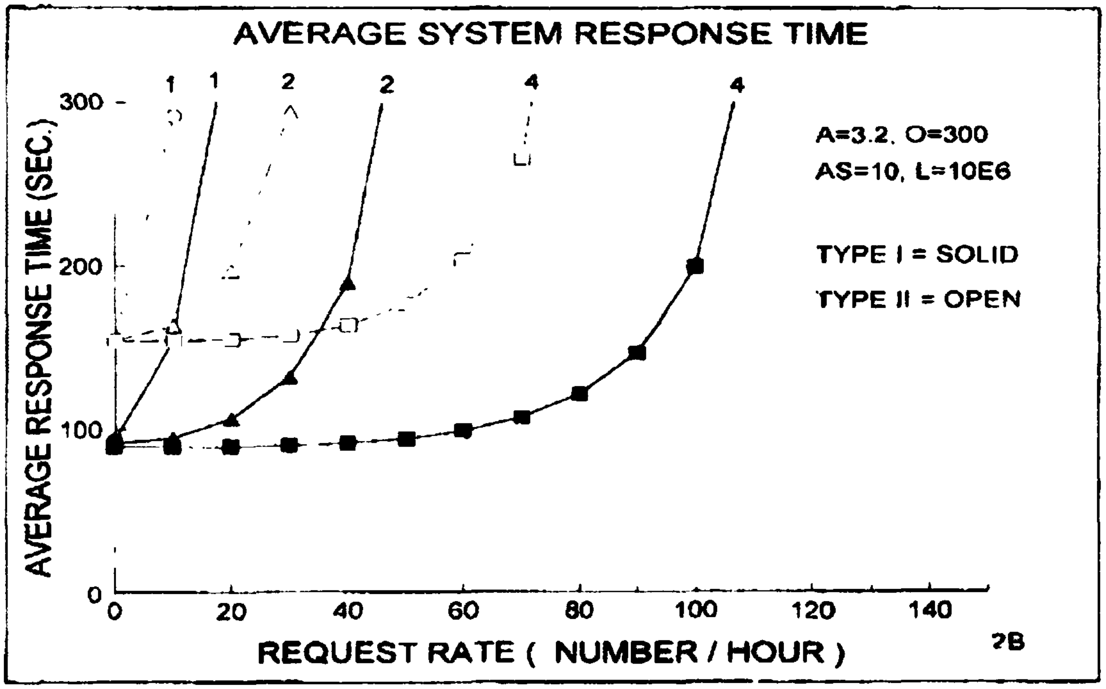


Figure 2B. Average system response time, SRT, as a functicn of service request rate (#/hour), for N = 1, 2, 4 drives of Device Type I and Device Type II. A = 3.2, AS = 10 seconds, L = 10 TB, O = 300 MB.

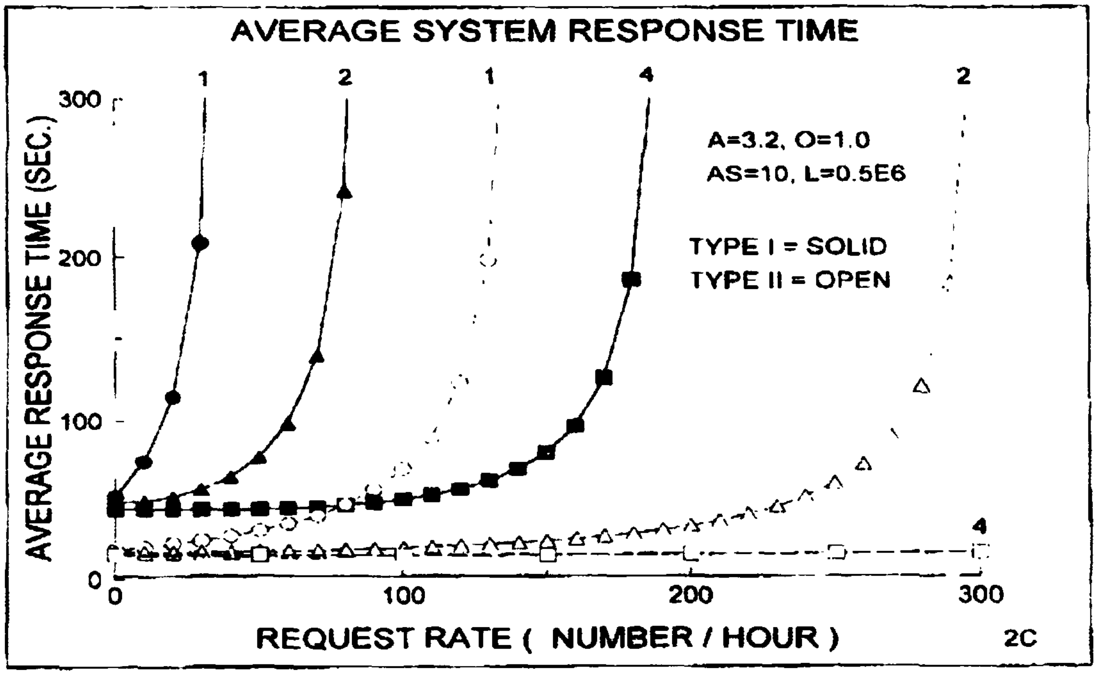


Figure 2C. Average system response time, SRT, as a function of service request rate (#/hour), for N = 1, 2, 4 drives of Device Type I and Device Type II. A = 3.2, AS = 10 seconds, L = 0.5 TB, O = 1 MB.

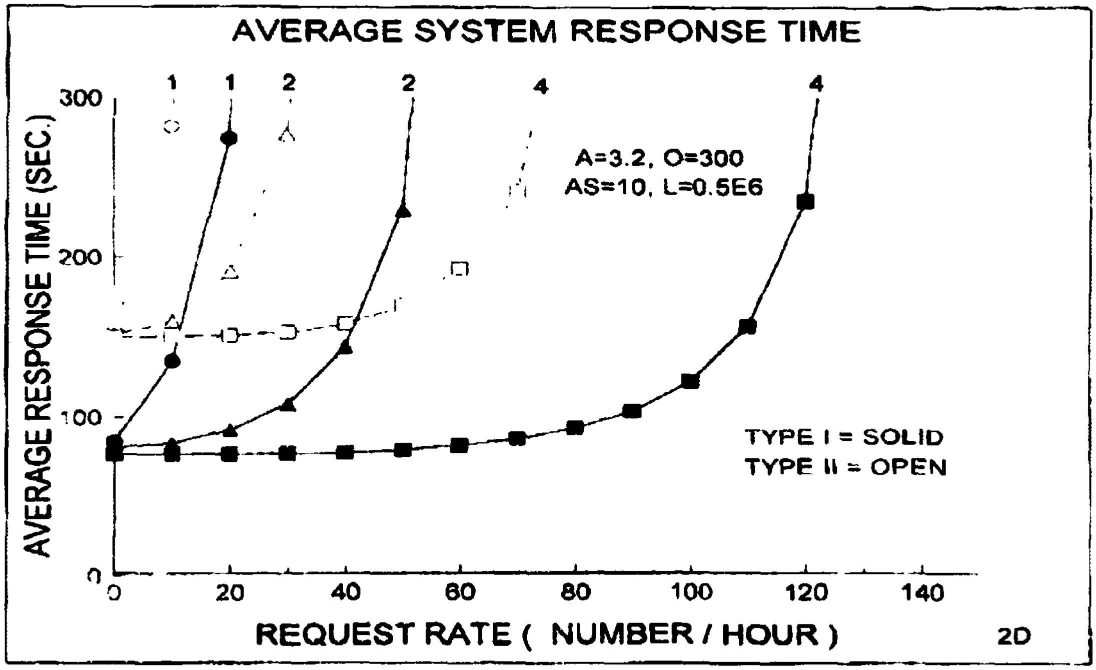


Figure 2D. Average system response time, SRT, as a function of service request rate (#/hour), for N = 1, 2, 4 drives of Device Type I and Device Type II. A = 3.2, AS = 10 seconds, L = 0.5 TB, O = 300 MB.

It is readily apparent from the data presented in Figure 2 that Device Type I provides superior performance (on a per drive basis) for those applications requiring large object sizes and modest request rates. Conversely, Device Type II excels for those applications characterized by modest object sizes and high request rates. In order to get a better perspective of the preferred operating domain for these two different types of devices, the data is next presented as a series of isochrons (lines of constant system response time) over the domain space of object size X request rate. The information is presented with number of drives as one of the parameters thus resulting in performance comparisons on a "per drive" basis.

B. System Response Time Isochrons

Figures 3A and 3B display isochrons of average system response time for Device Types I and II respectively. These figures display the operational range of object size (MB) X request rate (#/hour) that can be satisfied within 60 seconds, 90 seconds, or 120 seconds, for a system configuration of 4 drives, a library capacity of 10 TB, and a random retrieval factor, A = 3.2. In Figure 3B, as a result of the faster response times (for small to medium object sizes) of the Type II Device, an isochron at 30 seconds is also shown. Figure 4 shows only the 90 second isochron for both types of devices in overlay fashion to better illustrate the operational domain where each type of device excels, as well as the range of overlap.

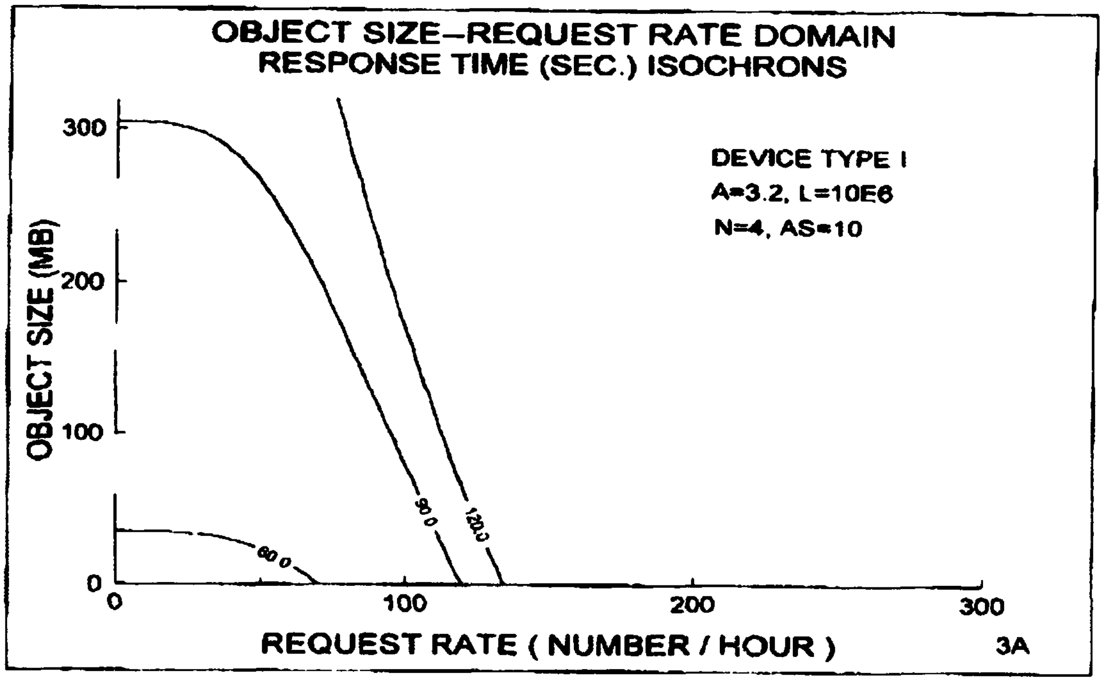


Figure 3A. Isochrons of average system response time, SRT, in seconds in the operating domain of object size (MB) X request rate (#/hour). A = 3.2, N = 4, L = 10 TB, AS = 10 seconds. Device Type I.

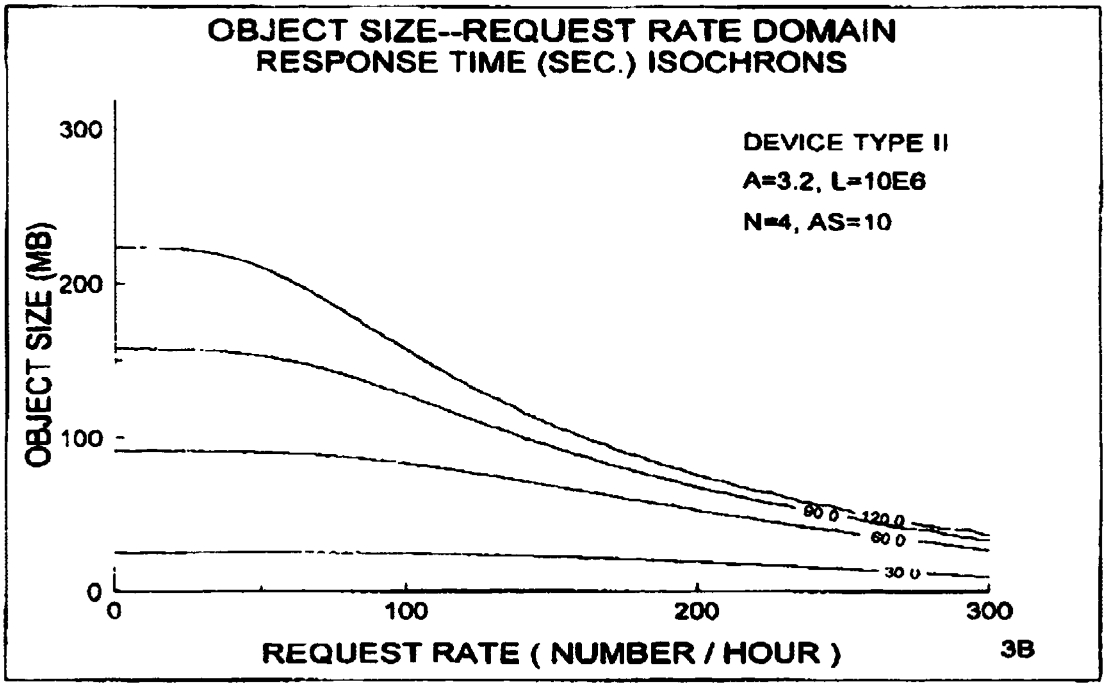


Figure 3B. Isochrons of average system response time, SRT, in seconds in the operating domain of object size (MB) X request rate (#/hour). A = 3.2, N = 4, L = 10 TB, AS = 10 seconds. Device Type II.

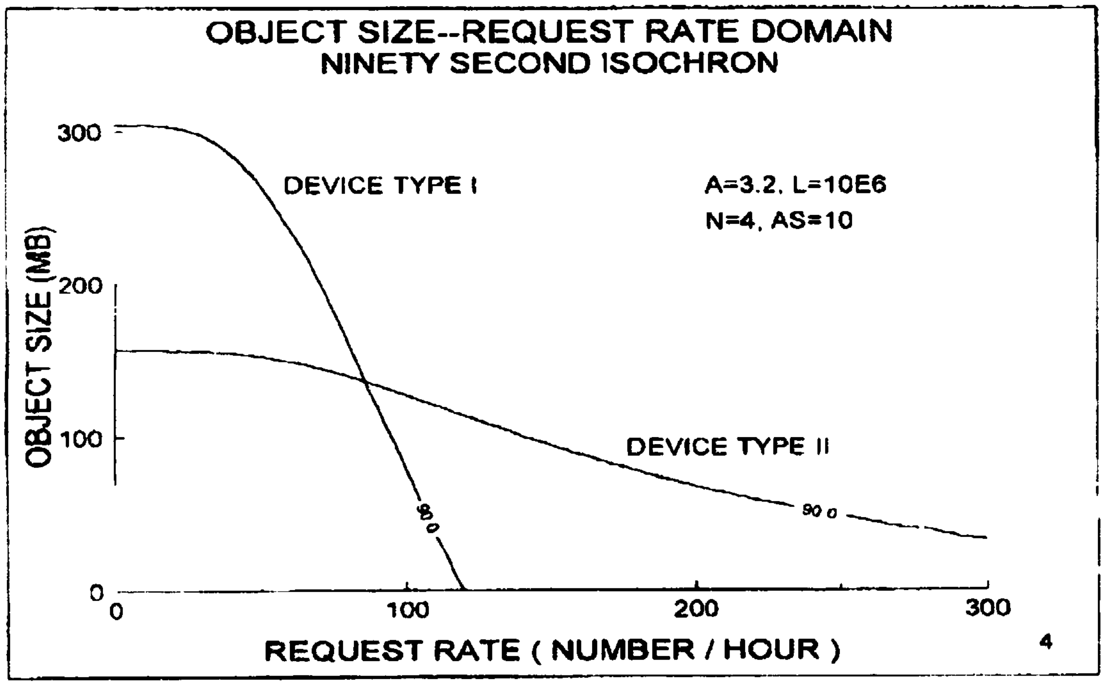


Figure 4. Ninety second isochron of average system response time in the operating domain of object size (MB) X request rate (#/hour). Overlay of Device Type I and Device Type II. Same conditions as defined in Figure 3.

Discussion

These analyses provide a means of quantifying expected performance for many variables characterizing different types of devices, different system configurations, and different application parameters. It is obvious that a higher data rate device will perform bettei than a lower data rate device for large object sizes. However, because of the non-linear nature of the queuing delays, it is not obvious where the crossovers occur in the operating space of object size X request rate. A comparison of the data in Figures 3A and 3B illustrates that for object sizes up to slightly in excess of 100 MB, the Type II device equals or exceeds the performance of the Type I device for all values of the request rate. At 150 MB, the Typ^ II device becomes superior only for request rates greater than approximately 100 per ho: for the configurations assumed. In order to convert this type of analysis to a price-performance analysis (rather than on a per drive basis) it would be necessary to first convert the configurations to equal dollar configurations and then compare performance over the operational space of interest for the given application requirements.

The performance results are specific for the defined assumptions made for the cycle time and service time components. Performance enhancements via software control algorithms are possible. For example, the algorithm used in this analysis assumes that following a read operation, the device rewinds the tape to the beginning of the tape and the tape remains mounted until a service request arrives that requires a new cartridge mount whereupon a drive is unloaded. If a request arrives for an object on a cartridge that is mounted, the drive searches from the beginning of tape , the new object location without invoking a robotic action. Depending upon the application, two possible alternative cycle sequences may provide better performance. In one situation it may be preferable to search for an incoming request from the stop point of reading the previous request rather than automatically rewinding to the beginning of tape. This could result in shorter average search distances. Another scenario could provide preemptive drive unloads [5] which might shorten robotic service times under some application conditions. An early pre-analysis of the specific use conditions would permit "tuning" the system for optimized performance.

Cartridge Capacity Considerations

In passive tape storage applications that very infrequently retrieve stored data objects, the emphasis has been on higher data rates and higher cartridge capacity. Increases in capacity can be achieved by either an increase in areal density or by way of a longer length, thinner tape. The K value, given in MB/M, is reflective of the areal density capability for a given tape width. For a fixed value of K, capacity is linearly dependent upon tape length, which in turn affects search and rewind times. An analysis of the average system response time as a function of cartridge capacity (i.e. length) is shown in Figures 5 and 6 as isochrons of average response time over the domain space of request rate X capacity for the fixed conditions listed.

Device Type II is unique in its design for type storage devices to be able to economically provide solutions for active applications such as HSM and digital libraries. However, for wide acceptance, it must also be capable of meeting the needs of the passive applications.

Hence, it was necessary to provide a carefully considered balance between capacity and response time under multi-user loaded conditions. The performance target was set to be in the range of 15-30 second average response time for request rates of a few hundred per hour and with an economical number of devices. There are many variables that affect the design space over which these objectives may be achieved. These include the randomization factor (A), the library size, the robot cycle time, and the size of the object being retrieved Using the technology recording density (K = 34) and the design of Device Type II, a capacity in the range of 5-10 GB provides a reasonable balance to meet the wide range of application characteristics. This is illustrated in Figures 5 and 6. These analyses are analogous to assessments of the trade-offs made between disk storage capacity and number of actuator arms. The result has been smaller physical disk sizes as the technology advanced to provide higher recording densities.

Figure 5A presents 20, 25, and 30 second isochrons over the domain space of request rate X cartridge capacity for the system parameters stated. Of note, randomization factor, A, is set at 2.2, the accessor cycle time = 15 seconds and the number of drives = 2. Foı a 5 GB cartridge capacity an average response time of ≤ 25 seconds is maintained up to approximately 100 requests per hour. Figure 5B illustrates the improved performance and the enlarged acceptable operating domain resulting from the addition of a third drive. Alternatively, improvements may be obtained by using a faster accessor. The results obtained with an accessor cycle time, AS of 10 seconds (and with 2 drives), is shown in Figure 5C. Figure 6A illustrates the effect (with 3 drives and AS = 15 seconds) of an application that has a highly non-random recall pattern (A = 3.2). This results in a high 'hit'

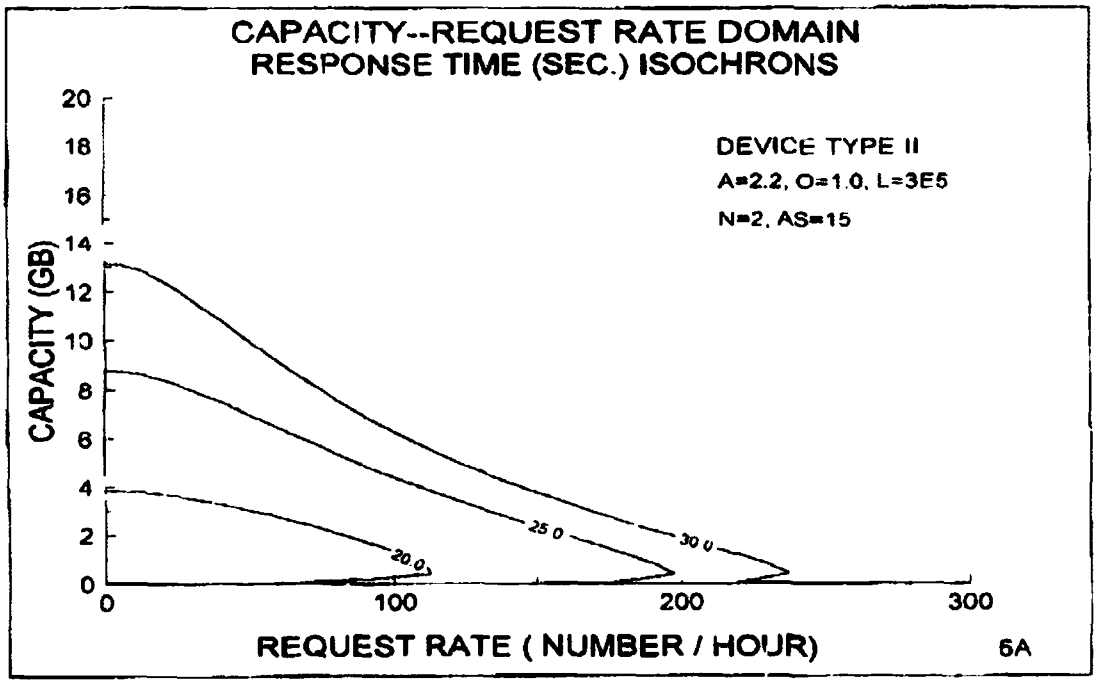


Figure 5A. **Isochrons of average system response time, SRT, (seconds) in the design space of request rate X capacity (as determined by tape length) for Device Type II. A=2.2, 0=1 MB, L=3x10^5 MB, N=2, AS=15 seconds.**

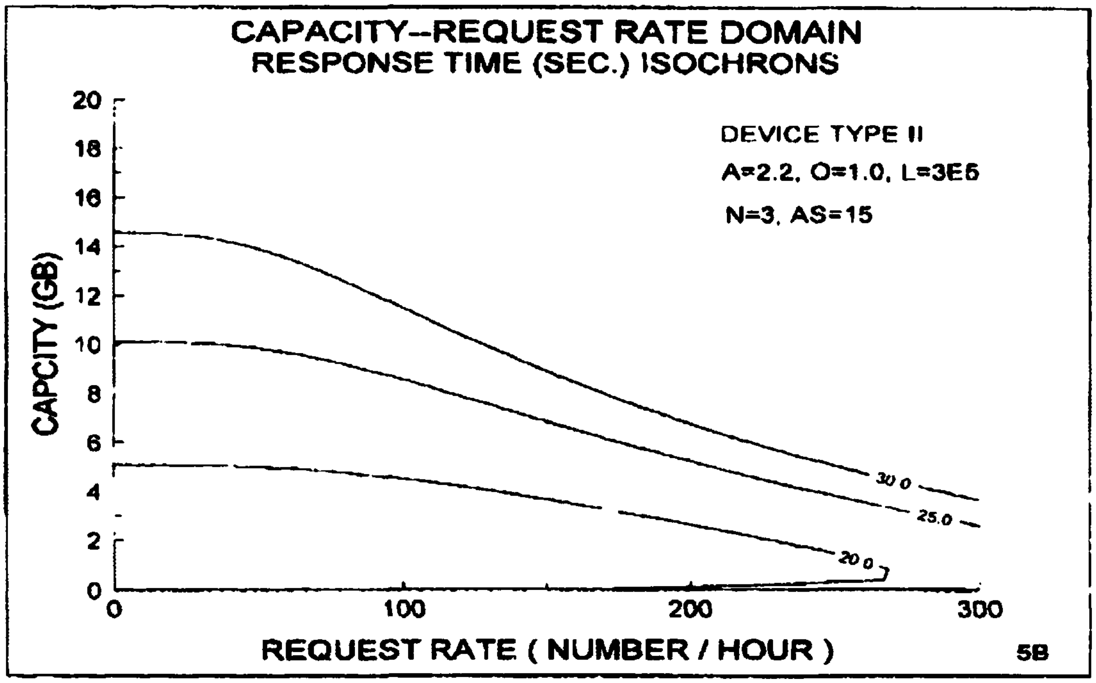


Figure 5B. Isochrons of average system response time, SRT, (seconds) in the design space of request rate X capacity (as determined by tape length) for Device Type II. A=2.2, 0=1 MB, L=3 x 10^5 MB, N=3, AS=15 seconds.

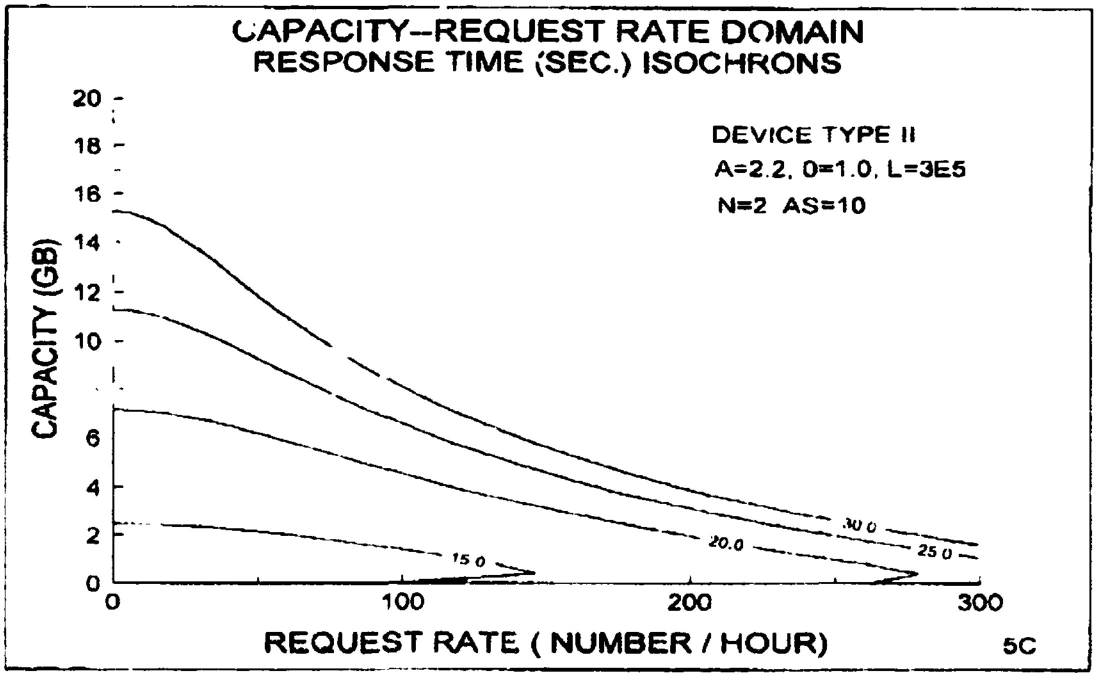


Figure 5C. Isochrons of average system response time, SRT, (seconds) in the design space of request rate X capacity (as determined by tape length) for Device Typ. II. A=2.2, 0=1 MB, L=3x10^5 MB, N=2, AS=10 seconds.

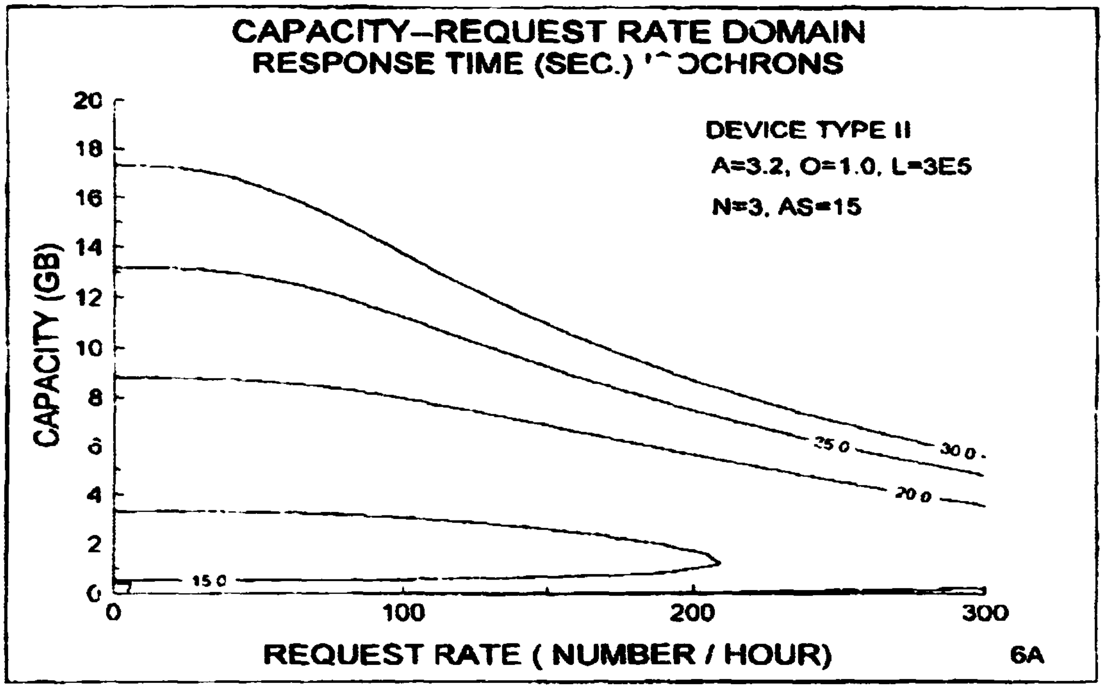


Figure 6A. Isochrons of average system response time (seconds) in the design space of request rate X capacity (as determined by increased tape length) for Device Type II. N=3, AS=15 seconds, O=1 MB, L=3x10^5 MB. A=3.2.

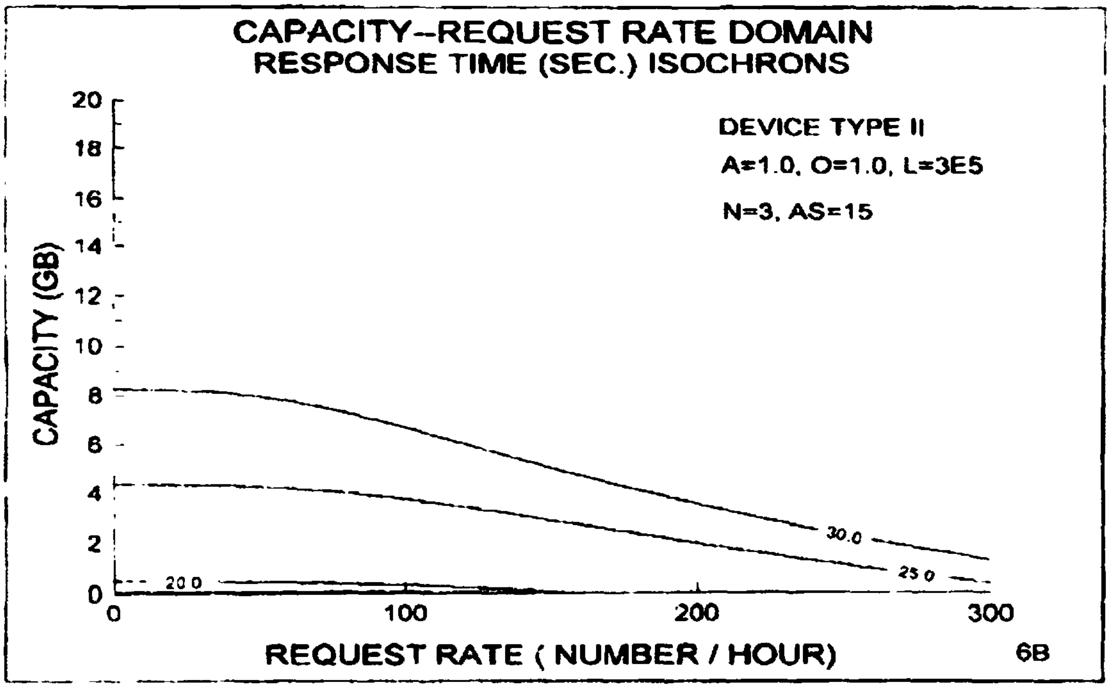


Figure 6B. Isochrons of average system response time (seconds) in the design space of request rate X capacity (as determined by increased tape length) for Device Type II. N=3, AS=15 seconds, O=1 MB, L=3x10^5 MB. A=1.0.

rate to mounted cartridges and hence performance improvement as a result of fewer required robot movements, and load-unload cycles. Figure 6B shows the other extreme of a totally random retrieval pattern (A=1) for the same system configuration parameters. Figures 6A and 6B should be compared to Figure 5B to see the effect of the application retrieval pattern on the response time for an otherwise fixed set of system parameters. Clearly, optimizing for one application will result in sub-optimization for other applications. The values chosen for the parameters for Device Type II in Table 3 provide a good balance, are achievable with current technology, and provide a basis for possible future technology enhancements.

Conclusions

The type of analysis presented here may be useful to the application engineer in comparing different STARS as to suitability for different application requirements. Likewise, this analysis has been used by developers to guide the development of device characteristics to meet existing or anticipated application requirements. The diversity of applications precludes the possibility of a single device doing all jobs equally well. A comparison of the preferred operating domains for two different types of devices which have been developed from a common advanced recording technology has been presented.

References

1. J.J. Gniewek, "Application of Queuing Theory to Performance Analysis of Automated Removable Media Storage Subsystem Response Time - Bounding the Problem," submitted for publication in IBM Journal of Research and Development.

2. J.J. Gniewek, "Factors Affecting Response Time Performance of Removable Media Storage Subsystems." Internal IBM Tucson Technical Report (1996). (This report is available from the author by e-mail request).

3. J. White, J. Schmidt, and G. Bennett, "Analysis of Queuing Systems," Academic Press, New York (1975).

4. J.J. Gniewek and S.M. Vogel, "Influence of Technology on Magnetic Tape Storage Device Characteristics," NASA Conference Publication 3295, Fourth NASA Goddard Conference on Mass Storage Subsystems and Technologies, 237-251 (March, 1995).

5. J.C. Hartung, et. al., "Preemptive Demount in a.. Automated Storage Library," U.S. Patent 5,239,650, issued 8/24/1993.

NEXT
DOCUMENT

Optimizing Input/Output Using Adaptive File System Policies[*]

Tara M. Madhyastha, Christopher L. Elford, Daniel A. Reed
Department of Computer Science
University of Illinois
Urbana, Illinois 61801
e-mail: tara@cs.ui.edu

Abstract

Parallel input/output characterization studies and experiments with flexible resource management algorithms indicate that adaptivity is crucial to file system performance. In this paper we propose an automatic technique for selecting and refining file system policies based on application access patterns and execution environment. An automatic classification framework allows the file system to select appropriate caching and prefetching policies, while performance sensors provide feedback used to tune policy parameters for the specific system environment. To illustrate the potential performance improvements possible using adaptive file system policies, we present results from experiments involving classification-based and performance-based steering.

1. Introduction

Input/output performance is the primary performance bottleneck of an important class of scientific applications (e.g., global climate modeling and satellite image processing). Moreover, input/output characterization studies such as Crandall [1] and Smirni [2] have revealed that parallel applications often have complex, irregular input/output access patterns for which existing file systems are not well optimized. Experience has shown that a few static file system policies are unlikely to bridge the growing gap between input/output and computation performance. In this paper we propose an automatic technique for selecting and refining file system policies based on application access patterns and execution environment. Knowledge of the input/output access pattern allows the file system to select appropriate caching and prefetching policies while the specific execution environment determines what policy refinements are necessary to further improve performance. For example, a sequential access pattern might benefit from sequential prefetching. The available memory and access latencies determine the quantity of data that should be prefetched. By being responsive to both application demands and system environment, this approach can provide better performance than a single static file system policy.

Adaptive file system policy controls rely on continuously monitoring access patterns and file system performance. We obtain a qualitative access pattern classification either through automatic analysis of the input/output request stream or via user-supplied hints. We also

[*] Supported in part by the National Science Foundation under grant NSF ASC 92-12369, by the National Aeronautics and Space Administration under NASA Contracts NGT-51023, NAG-1-613, and USRA 5555-22 and by the Advanced Research Projects Agency under ARPA contracts DAVT63-91-C-0029, DABT63-93-C-0040 and DABT63-94-C-0049

continuously monitor file system performance sensors (e.g., cache hit ratios, access latencies, and request queue lengths). The values of these sensors, together with the access pattern, are used to select and tune specific file system policies. For example, the file system can enable prefetching when the access pattern is sequential, using the interaccess delays determine how much data to prefetch. Updated performance sensor values or changing access pattern classification may result in additional refinements to file system policies.

The remainder of this paper is organized as follows. In ß2 we give a high-level overview of the adaptive file system infrastructure. Validation of these concepts requires an experimental framework; we have implemented adaptive file system policies within a portable, user-level file system called the Portable Parallel File System (PPFS) Huber [3], described in ß3. Our system has two major components; in ß4 we discuss how one automatically classifies user access patterns and uses this information to select file system policies. In ß5 we describe how to use an input/output performance summary generated from sensor values to select file system policies and parameters that should be modified to improve performance. Finally, ß6-ß7 place this work in context, summarize our results, and outline directions for future research.

2. Adaptive Steering

Given the natural variation in input/output access patterns, it is unlikely that one, static, system-wide set of file system policies will suffice to provide good performance for a reasonable range of applications. Even in a configurable environment, *a priori* identification of effective file system policies is difficult because application access patterns are sometimes data dependent or simply unknown. Furthermore, input/output requirements are a complex function of the interaction between system software and executing applications and may change unpredictably during program execution. We believe that integration of dynamic performance instrumentation and automatic access pattern classification with configurable, malleable resource management algorithms provides a solution to this performance optimization conundrum. Below, we describe the two major components of this approach.

2.1. Classification-Based Policy Selection

Parallel file system research such as Patterson [4], Kotz [5], Krieger [6], and Grimshaw [7] has demonstrated the importance of tuning file system policies (e.g., caching, prefetching, writeback) to application access patterns. For example, access pattern information can be used to guide prefetching, small input/output requests can be aggregated and large requests can be streamed.

One intuitive way to provide the file system with access pattern information is via user supplied hints, or qualitative access pattern descriptions, for each parallel file. Unfortunately, this approach requires ongoing programmer effort to reconcile the hints with code evolution. Inaccurate hints can cause performance problems if the file system selects policies that are unsuitable for the actual access pattern.

Our solution to this dilemma is to automatically classify access patterns during program execution. This approach requires no programmer intervention and is robust enough to handlc dynamically changing or data-dependent access patterns. A classifier module observes the application-level access stream and generates qualitative descriptions. These descriptions, combined with quantitative input/output statistics, are used to select and tune file system policies according to a system-dependent algorithm. Hints can be used in conjunction with this approach to provide access pattern information that cannot be intuited from the access stream (e.g., collective input/output).

2.2. Performance-Based Policy Selection

Although application access pattern information is a prerequisite for selecting appropriate file system policies, input/output performance ultimately determines the success of a particular policy choice. Extrinsic (external) input/output phases that occur when other applications compete for shared resources are equally important to file system policy selection, yet are not evident from application access patterns alone. Using a basic feedback system as a model, we can frame parallel file system policy optimization as a dynamic steering problem that tracks performance to refine file system policy selection. This type of computational steering framework has proven useful in other contexts (e.g. Vetter [8], Wood [9], Gergeleit [10], and Gu [11].)

In our dynamic steering framework, we monitor performance sensors that encapsulate the performance of critical file system features, consult access pattern dependent policy selectors that map changes in input/output performance to potential policy changes, and invoke system actuators to effect these policy changes. The resulting performance sensor metrics reflect the influence of our policy reconfiguration. When coupled with autrmatic access pattern detection, this closed loop steering infrastructure can adapt file system policies to match application access patterns and then tune these policies to the dynamic availability of system resources.

3. Portable Parallel File System (PPFS)

PPFS is a portable input/output library designed as an extensible testbed for file system policies [3]. A rich interface for application control of data placement and file system policies makes it exceptionally well-suited for our experiments. Below we describe the PPFS design and extensions that facilitate adaptive file system policy experiments.

3.1. PPFS Design

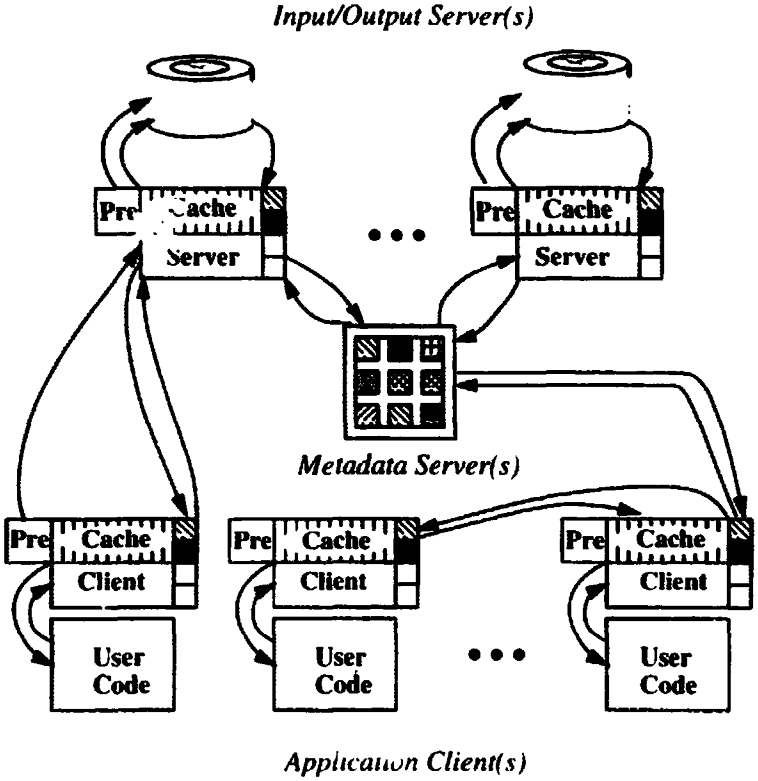


Figure 1: Basic PPFS Design

Figure 1 shows the PPFS components and their interactions. Application clients initiate input/output via invocation of PPFS interface functions. To open a file, the PPFS library first contacts the metadata server, which loads or creates information about the file layout on remote disk servers (input/output nodes). With this information, the application is able to issue input/output requests and specify caching and prefetching policies for all levels of the system. Clients either satisfy the requests or forward them to servers (abstractions of input/output devices). Clients and servers each have their own caches and prefetch engines. All "physical" input/output is performed through underlying UNIX file systems on each PPFS server.

In the PPFS input/output model, files are accessed by either fixed or variable length records, and the PPFS library has an extensible set of interfaces for specifying file distributions, expressing input/output parallelism, and tuning file system policies. For example, the user can specify how file records are distributed across input/output nodes, how and where they are cached, and when and where prefetch operations should be initiated.

3.2. PPFS Extensions

The original PPFS interface provides the application with a rich set of manual file system policy controls and structured data acce s functions, but the rules guiding their use are *ad hoc*. Ideally, the file system should automatically infer appropriate policies from low-level application access patterns, lessening the application programming burden and the likelihood of user misconfiguration. Dynamic performance data should be used to verify and refine these policy decisions. Through automatic access pattern classification, used to select file system policies, and performance-based policy refinement, we automate file system policy control. This has motivated two basic extensions to the base PPFS design: support for automatic access pattern classification and automatic policy refinement based on monitoring input/output performance.

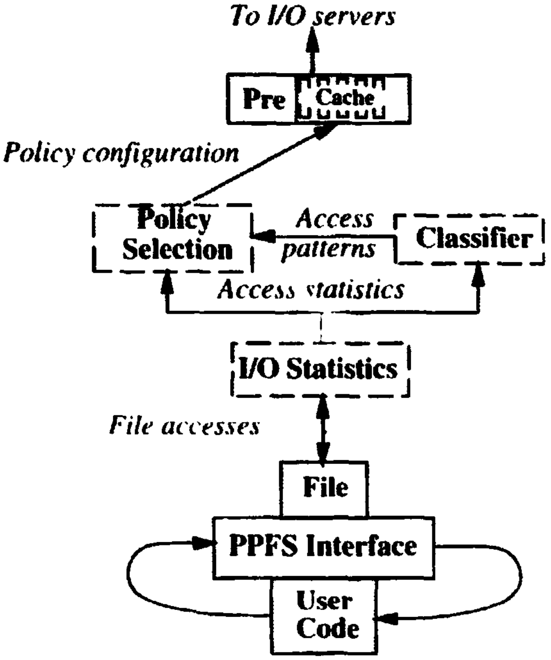


Figure 2: PPFS Classification and Policy Selection Extension

We have replaced manual PPFS file system controls in our extension by an adaptive access pattern classification and file system policy selection mechanism. During program execution, an input/output statistics module monitors the file access stream (each access is represented as a byte offset, read or write, and request size) and computes the statistics needed by the classifier module. PPFS uses the classification to select and tune prefetching and caching policies. Figure 2 illustrates the interaction of the classification extensions with the original PPFS components.

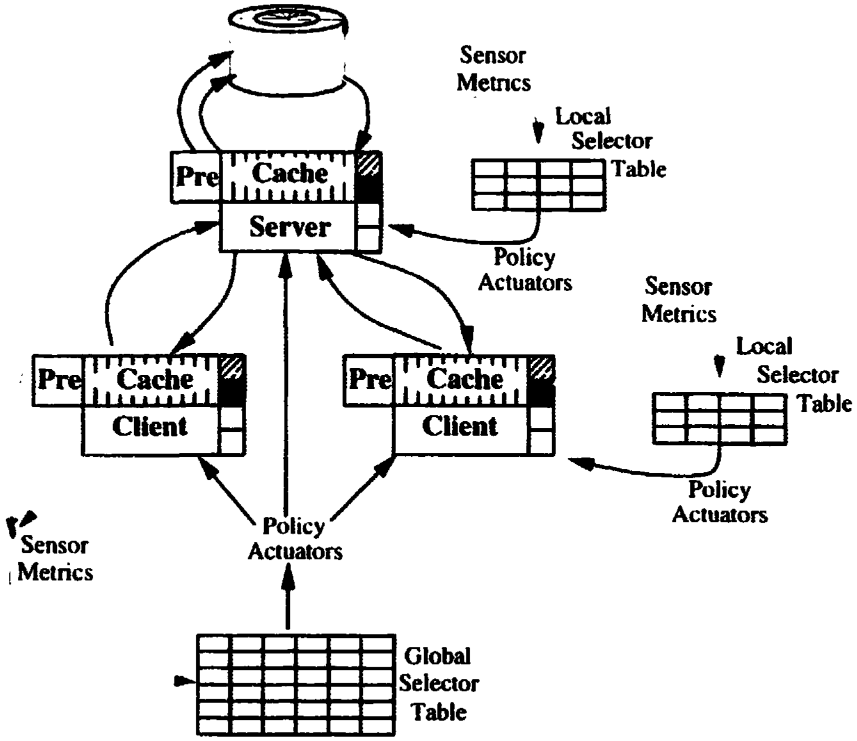


Figure 3: PPFS Performance Monitoring and Steering Extension

To refine policy selections using performance data, we instrumented the system components to periodically provide ensor metrics and created sensor-driven selector tables to automate invocation of the same native PPFS policy controls that a PPFS user could invoke manually. Figure 3 shows how our performance based policy selection extension interacts with the PPFS. Dynamically computed sensor metrics (e.g., input/output queue lengths, cache hit ratios, inter-request latencies) are routed to local and global policy selector tables, where they index appropriate file system policies and parameters for the system environment.

The local policy selector can only change local policies. For example, a client selector table may decide to increase the client file cache space and the number of records to prefetch ahead. It cannot change file system policies on other client nodes or on the PPFS servers. As shown in Figure 3, sensor metrics are also routed to a global selector mechanism that can select policy parameters for other nodes. For example, if the write throughput visible to client nodes for large writes drops below a certain threshold, the clients may elect to disable caching, and stream data directly to the PPFS servers. Rather than waiting for the individual server metrics and selector tables to disable server caching and stream data to disks, the global selector mechanism detects this input/output phase shift in the clients and invokes the policy change on the servers.

4. Automatic Classification and Policy Selection

As described in ß3, we have replaced the manual file system controls in PPFS with an adaptive access pattern classification and policy selection mechanism. Below we describe in greater detail our classification and policy control methodology.

A file access pattern classification is useful if it describes the input/output features that are most relevant to file system performance; it need not be perfectly accurate. For example, one might classify an input/output pattern as "sequential and write only" even if there are occasional small file seeks and reads -- this would suffice to correctly choose a sequential prefetching policy. Such a qualitative description is difficult to obtain based on heuristics alone. Instead, one needs a general classification methodology capable of learning from examples.

As a first step toward adaptive file system policies, we have implemented automatic access classification to select file system policies, adapting to application requirements. This is only half of the complete system; after making policy selections we rely upon performance sensor data to refine policy parameters, adapting to the total system environment. Performance-based steering is the subject of ß5.

4.1. Classification Methodology

Within a parallel application, file input/output access patterns can be observed at two levels. The first is at the local (e.g., per thread) level, and the second is at the global (e.g., per parallel program) level. For example, a parallel file might be distributed across the threads of a parallel program in such a way that each thread appears to be accessing the file locally in strides, but the interleaved access stream is globally sequential. Global classifications are formed from local classifications and input/output statistics. In ß4.1.1 we describe our access pattern classification approach. In ß4.1.2 we illustrate how global classification works in a parallel application.

4.1.1. Access Pattern Classification

To accommodate a variety of underlying file structures and layouts, we describe access pattern classifications assuming a byte stream file representation. File accesses are made using UNIX style read, write, and seek operations, and file access patterns are determined from this representation. Thus, an input/output trace of file accesses may be represented as a stream of tuples of the form

$$\left\langle \textit{byte offset}, \textit{request size}, \textit{read} \,/\, \textit{write} \right\rangle$$

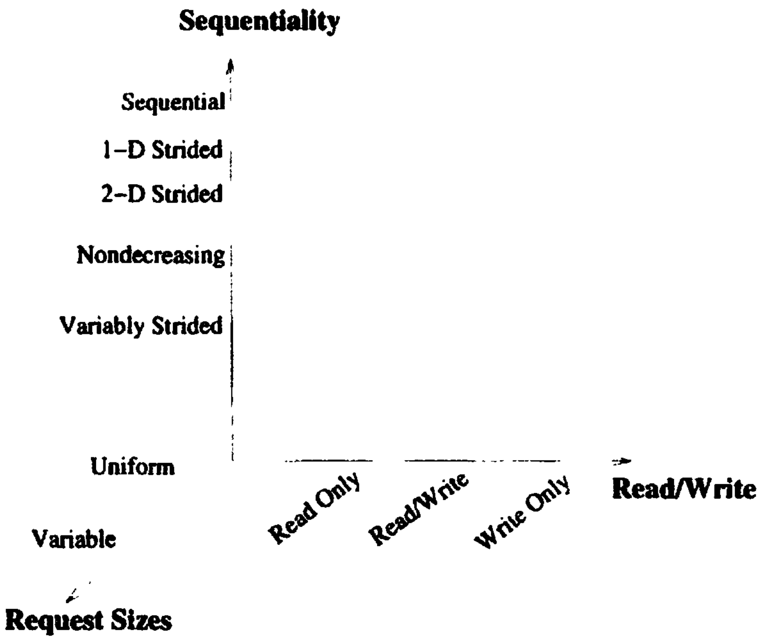


Figure 4: Access Pattern Space

Patterns observed in each of the time-varying values of the tuple components form a three dimensional access pattern space. Figure 4 shows certain categories along each axis that can be used to influence file system policy selection and label all points in the access space. Additional categories can be added as necessary to each axis to further refine the access pattern space.

Many techniques can be used to classify and identify observed access patterns within the space shown in Figure 4. Our approach is to train a feed-forward artificial neural network as in Hinton [12] to classify patterns. Although neural networks are expensive to train initially, once training is complete, classification is very efficient. To train the neural network, we represent the access pattern in a compact, normalized form by computing input/output statistics on a small fixed number of accesses, called the classification window. For example, representative statistics might be the number of unique read request sizes, or a transition matrix of the probabilities that one type of request (read/write) will follow the other.

Table 1: Input/Output Trace Features

Category	Category Features			
Read/Write	Read Only	Write Only	Read-Update-Write	Read/Write Nonupdate
Sequentiality	Sequential	1-D Strided	2-D Strided	Variably Strided
Request Sizes	Uniform		Variable	

Table 1 shows the features recognized by our trained neural network. These features correspond directly to planes or regions within the space shown in Figure 4. The neural network selects one and only one feature within each category; for example, a set of accesses cannot be both read only and write only. Neural networks ar^ inherently imprecise, allowing us to train a network to identify patterns that are "close" to a well-defined pattern in a more general way than specifying heuristics. For example, a pattern might be treated as read-only if there is only one small write among very large reads, but read/write if the single write is the same size as the reads. This allows us to train the file system to classify new access patterns.

4.1.2. Global Access Pattern Classification

Local access pattern classification is only part of a larger classification problem. Local classifications are made per parallel program thread; however, the local acc ss patterns within a parallel program merge during execution, creating a global access patte Global knowledge is especially important for tuning file system policies. For example, if all processors access a single file sequentially, one could potentially improve performance by employing a caching policy that does not evict a cached block until every processor has read it.

Our global classification infrastructure is based on an access pattern algebra. We combine local classifications and other local information to make global classifications. For example, if all local access patterns are read only, the global access pattern is read only. The number of processors contributing to the global access pattern is called the cardinality of the classification. Generally, we attempt to make global classifications with cardinality p, where p is the number of processors involved in the global input/output. However, a global classification involving a subset of the these processors is still useful for policy selection. A partial global classification may even be preferable, if it more accurately represents the temporal characteristics of the global access pattern.
Global access pattern classification cannot be useful for influencing file system policies unless we recognize common global access patterns in time to effect policy changes. To demonstrate that this is feasible, we have examined parallel applications from the Scalable Input/Output (SIO) application suite [1,2]. These applications exhibit a variety of global access patterns, including global sequential, partitioned sequential (processors sequentially access disjoint partitions), and interleaved sequential (individual strided access patterns are globally interleaved). The patterns are primarily read-only or write-only with regular and irregular request sizes. All of these patterns can be recognized by our classification infrastructure.

One specific application area we have examined is computational fluid dynamics. PRISM is a parallel implementation of a 3-D numerical simulation of the Navier-Stokes equations from Henderson [13,14]. The parallelization is implemented by apportioning slides of the periodic domain to the processors, with a combination of spectral elements and Fourier modes used to investigate the dynamics and transport properties of turbulent flow.

Figure 5 shows a file access timeline for PRISM on a 64 processor Intel Paragon XP/S running OSF/1 version 1.4. This code exhibits three distinct input/output phases. During the first phase, every processor reads three initialization files (m16.rst, m16.rea and m16.mor). Each file is accessed with a global sequential access pattern; m16.rst is also accessed with an interleaved sequential access pattern. In the second input/output phase, node zeio performs input/output on behalf of all the nodes, writing checkpoints and data (access to files m16.Rstat, m16.Qstat, m16.Vstat, m16.mea and m16.his). In the final phase, the result file is written to disk by all processors in an interleaved sequential access pattern m16.fld. Phases two and three occur iteratively throughout program execution.

When accesses are adjacent and very small, local classification windows (the time to make ten input/output accesses) are short, and we must observe more windows to detect overlap among processors and global behavior. For example, Figure 5a and Figure 5b show local classification times for a globally sequentially accessed initialization file (m16.rea). The reads are very small (most are less than 50 bytes) and we reclassify the pattern every ten accesses. We can make a global sequential classification when sequential access patterns with overlapping bytes have been detected on every processor. Despite initial startup asynchronicity, the slowest processor (number 31) completes its tenth access to this file at 7.79 seconds. Because this initialization input/output phase accounts for approximately !25 seconds of execution time, adapting file system policies to the access pattern is fundamental to improving performance.

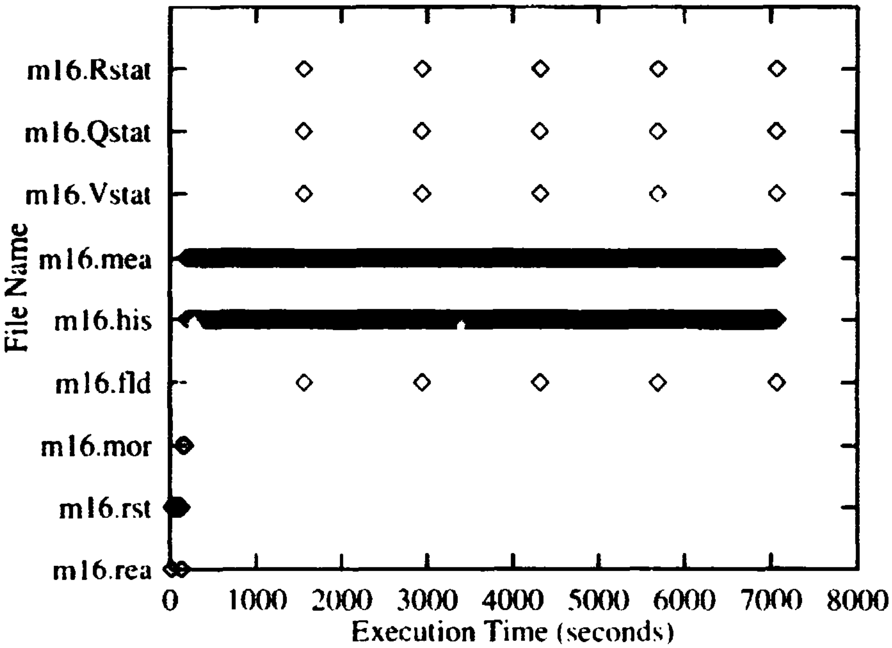
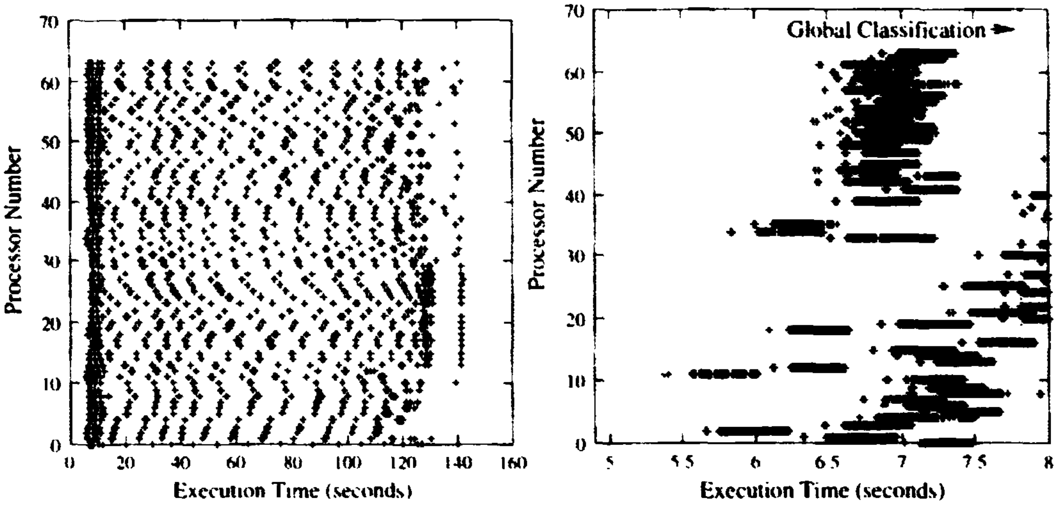


Figure 5: PRISM: File Access Timeline



(a) All Classifications

(b) Detail

Figure 6: PRISM: Local Processor Classification Points for Global Sequential Access Pattern

4.2. Intelligent Policy Selection

A file access pattern classification as described above is platform-independent and unique to a particular application execution. However, an optimal choice of file system policies for a particular access pattern is system-dependent. A file system uses the classification to tune file system policies for each input/output platform. By making policy decisions to suit the application requirements and the system architecture, not only is input/output performance portable over a variety of platforms, but the file system can provide better performance over a range of applications than it could by enforcing a single system-wide policy. This adaptivity should occur transparently, without application hints or user level optimizations.

Abstractly, PPFS continuously monitors and classifies the input/output request stream. This classification is passed to the file system policy suite for policy selection and configuration. For example, when the access pattern classification is sequential, the file system can assume that file access will continue to be sequential. If the classification is read only, the file system can prefetch aggressively; if it is write only, a write-behind policy might be efficient. When the classification is regularly (1-D or 2-D) strided, the file system can take advantage of this information to adjust the cache size and prefetch anticipated blocks according to the access and stride sizes.

As described in §4.1.2, we can combine local classifications to make global classifications, which we use to adjust policies at all system levels with global knowledge. For example, when all processors read the same file sequentially (global sequential) we can select a caching policy at input/output nodes that prefetches file blocks sequentially but does not flush cache blocks until every processor has accessed them. In contrast, if we detect an interleaved sequential global pattern, each input/output node could prefetch file blocks sequentially, retaining them only until each has been accessed in its entirety once.

Figure 7 shows a simple, parameterized example of a policy selection algorithm that selects PPFS policies for a uniprocessor UNIX workstation. Its default behavior is to favor small sequential reads, typical of UNIX workloads. However, when the classifier detects other access patterns, the algorithm adjusts policies to provide potential performance improvements. Quantitative values for the parameters of Figure 7 (e.g. LARGE_REQUEST) depend on the particular hardware configuration and must be determined experimentally.

The algorithm of Figure 7 is but one simple possibility for policy control. Richer control structures can be built upon more accurate models of input/output costs. However, in §4.3 we show that even this simple policy suite suffices to yield large performance increases over that possible with standard UNIX file policies. In §5 we describe our methodology for tuning automatically selected policies in response to overall system performance, closing the classification and performance feedback loop.

```
if (sequential) {
      if(write only) {
        enable caching
        use MRU replacement policy
      } else if (read only && average request size > LARGE_REQUEST) {
        disable caching
      } else {
         enable caching
         use LRU replacement policy
      }
}

if (variably strided || 1-D strided || 2-D strided (
     if (regular request sizes) {
         if (average request size > SMALL_REQUEST) {
              disable caching
         } else {
            enable caching
            increase cache size to MAX_CACHE_SIZE
            use LRU replacement policy
         }
     } else {
        enable caching
        use LRU replacement policy
     }
}
```

Figure 7: Dynamic File Policy Selection (Example)

4.3. Experimental Results

As a validation of automatic behavioral classification and dynamic adaptation, we used the enhanced PPFS to improve the input/output performance of Pathfinder, a single processor satellite data processing code. Pathfinder is from the NOAA/NASA Pathfinder AVHRR (Advanced Very High Resolution Radiometer) data processing project described in Agbu

[15]. Pathfinder processing is typical of low-level satellite data processing applications – fourteen large files of AVHRR orbital data are processed to produce a large output data set. It is an extremely input/output intensive application; over seventy percent of Pathfinder execution time is spent in UNIX input/output system calls.

4.3.1. Pathfinder

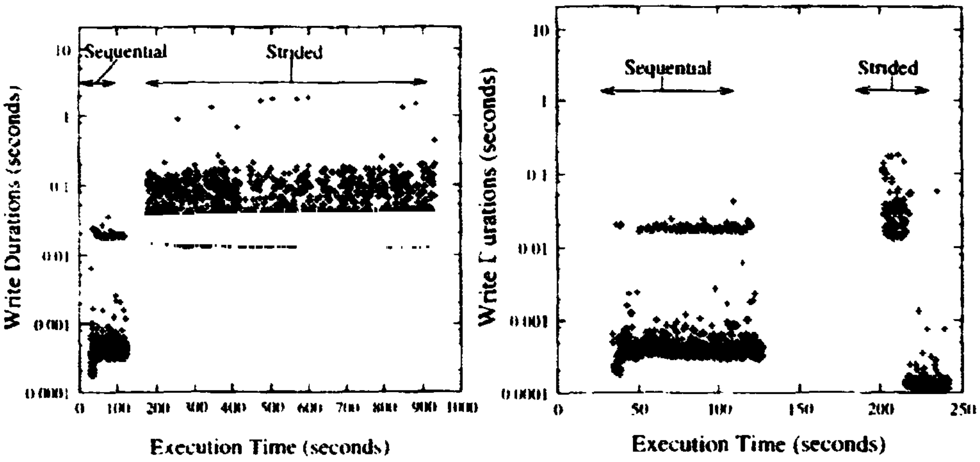
The goal of the Pathfinder project is to process existing data to create global, long-term time series remote-sensed data sets that can be used to study global climate change. There are four types of Pathfinder AVHRR Land data sets (daily, composite, climate, and browse images); we consider the creation of the daily data sets. Each day, fourteen files of AVHRR orbital data, approximately 42 megabytes each, in Pathfinder format are processed to produce an output data set that is approximately 228 megabytes in Hierarchical Data Format (HDF) from NCSA [16]. For simplicity, we examine the processing of a single orbital data file.

During Pathfinder execution, ancillary data files and the orbital data file are opened, and an orbit is processed 120 scans at a time. Although the orbit file is accessed sequentially, the access patterns for other ancillary data files range from sequential to irregularly strided. The result of this processing is written to a temporary output file using a combination of sequential and two-dimensionally strided accesses. Finally, the temporary file is re-written in HDF format to create three 8-bit and nine 16-bit layers.

Table 2 shows the relative execution times for Pathfinder using UNIX buffered input/output and PPFS with adaptive policies on a Sun SPARC 670. The dynamic adaptation of PPFS yields a speedup of approximately 1.87 with the policies Figure 7.[1] The PPFS automatic classifier could detect that the output file access pattern was initially write only and sequential, with large accesses, and that the pattern later changed to write only, strided, with very small accesses. Adapting to the first access pattern phase, PPFS selected an MRU cache block replacement policy. In the second phase it enlarged the cache, retaining the working set of blocks.

Figure 8a and Figure 8b illustrate the dramatic benefits of dynamic policy adaptation for Pathfinder's execution. Both graphs represent the same amount of input/output; however, in Figure 8a we use the same static policies for all access patterns. The first cluster of accesses in each graph is the write only sequential phase. Performance for the first phase is roughly equivalent using either MRU or the default, non-adaptive LRU replacement policy. However, enlarging the cache in the second phase substantially decreases the average write duration. PPFS successfully retains the working set of blocks (the overall cache hit ratio exceeds 0.99), while UNIX buffered input/output forces a write of 8 KB for every one or two byte access.

[1] However, due to limited physical memory, we disabled caching for small, variably strided reads.



(a) PPFS (Non-adaptive) PPFS (Adaptive)

Figure 8: Pathfinder Write Durations (Beginning Phase)

Table 2: Pathfinder Execution Times (seconds)

Experimental Environment	System Time	User Time	Total
UNIX	1578.2	1781.1	4299.3
PPFS	400.4	1270.4	2300.8

5. Performance-Based Steering

Although file system policy selection is partially a function of application input/output access patterns, system performance ultimately determines the success of a particular policy choice. Performance sensors provide feedback on file system behavior that can be used to optimize the parameters of policy decisions.

Below, we describe a complement to qualitative access pattern classification: sensor based, closed loop policy selection and configuration. As described in ß2.2 and shown in Figure 3, our framework partitions the steering problem into three components. The sensor metrics in ß5.1 provide input for policy selectors of ß5.2 which, based on system and application performance history, select policy parameters and activate them via the policy actuators of ß5.3.

5.1. Performance Sensors

Table 3: PPFS Sensor Metrics

Dimension	Description
Operation Count	Total number of input/output requests

Operation Time	Mean operation service time
Read Count Read Byte Count Read Time	Number of read requests Number of bytes read Mean read service time
Write Count Write Time	Number of write requests Mean write service time
Cache Hits Server Cache Hits Cache Check Time	Number of requests serviced by caches Number of requests serviced by offnode caches Time to check local cache
Server Time Server Queue Time Server Queue Lengths	Time spend on input/output servers Time spend in disk queue Length of disk queue
Prefetch Byte Count Prefetch Cache Check Time Prefetch Off Node Time Hit Miss Time	Number of bytes prefetched Time to scan cache on prefetch initiation Time spent offnode for prefetch operations Time spent waiting for overlapped prefetch to complete

To capture input/output performance data, we augmented PPFS with a set of performance sensors that are periodically sampled using the Pablo instrumentation library of Reed [17]. Table 3 shows the current PPFS sensor metrics. We chose these particular metrics because they are inexpensive to calculate, and we believe they are broad enough to reflect the performance of malleable file system policies within PPFS. In practice, many metrics are strongly correlated with others, magnifying or validating trends detected via other metrics.

5.2. Policy Selectors

Table 4: Sample Sequential Access Selectors

Sensor Conditions	Policy Options
(poor_read_service_times) & (many_read_requests) & (managable_byte_throughput) & (NOT high_hit_ratio)	Increase Cache Size Increase Prefetch Amount
(NOT managable_byte_throughput) & (low_hit_ratio)	Decrease Cache Size Disable Prefetch

Given detailed performance sensor metrics and an access pattern classification, our framework tunes file system policies using the sensor metrics as the indices to a selector table containing policy parameters for that set of sensor metrics. The dashed lines of Figure 1 show the flow of sensor data from PPFS modules to the policy selectors. Table 4 shows some sample selectors that a system might provide, given a sequential access pattern classification. For example, if the sensor metrics indicate that relatively small read requests take a long time and the cache hit ratio is low, we might increase the cache size and the number of blocks prefetched to anticipate the request stream. If the sensors indicate that too much data is being requested to effectively cache and prefetch, we may disable caching and prefetching altogether to avoid thrashing the cache.

The sensor rules shown in Table 4 are qualitative rather than quantitative. We quantify the selector table rules when we calibrate them with the specific sensor metrics for a given platform. For example, on an IBM SP/2 with 128 MB of memory per node manageable_byte_thruput may calibrate to (Read_Byte_Count[2] < 100 MB/second). Similarly on an Intel Paragon with only 32 MB of memory per input/output node, the calibration may be (Read_Byte_Count < 25 MB/second).

To create selector tables for a given access pattern, we need to know how different file system policies perform for this access pattern. By executing access pattern benchmarks with a variety of policies and under a variety of load conditions, we can develop a set of selector rules such as those shown in Table 4. We calibrate the qualitative rules on a given platform by storing the quantitative performance sensors with the qualitative rules. Our portable, dynamic steering infrastructure can then adapt to a system's resource constraints by simply loading selector tables calibrated for that system.

5.3. Policy Actuators

After the policy selector mechanism determines what file system policy parameters should be used, actuators provide the mechanism to instantiate policies and configure parameters. Currently, PPFS supports actuators that allow dynamic reconfiguration of cache sizes, replacement policies, and prefetch and write behind parameters on each client and server node. These actuators provide a rich variety of controls to our dynamic steering infrastructure. We have tested these controls by interactively steering application behavior based on a virtual reality display of the sensor metrics as in Reed [18].

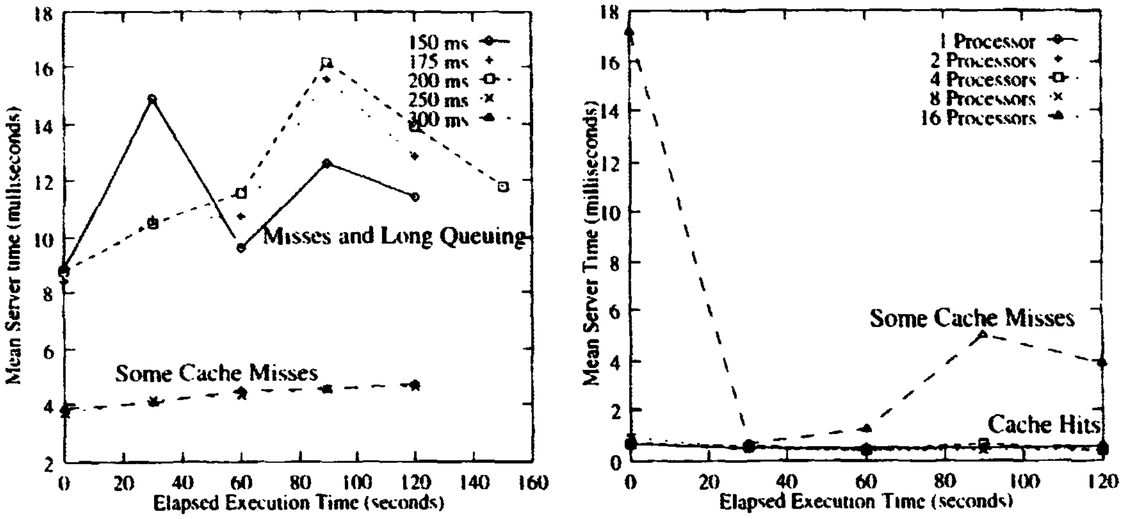
5.4. Experimental Results

To demonstrate the efficacy of sensor-based adaptive control when coupled with behavioral assertions, we used an input/output benchmark to conduct a set of simple experiments on several parallel architectures. We had several fundamental goals for the benchmark study. First, we wanted to verify that sensor metrics help us make improved PPFS policy decisions. We also wanted to determine how long we have to wait between policy changes to allow the sensor metrics to settle to their new steady state values.

In our benchmark, a group of tasks reads disjoint interleaved portions of a shared file. Task i reads all blocks, i modulo the number of tasks (e.g., task 0 of *p* tasks reads file

[2] Note that Read_Byte_Count is a sensor metric from Table 3

blocks 0, $2p$, p, ...) Between accesses, a processor computes for a uniform random interval with a parametric mean. We executed this benchmark on several parallel architectures with a variety of request sizes, prefetching options, and computation overheads for varying numbers of reader tasks.



a) Changing Interaccess Latency

b) Changing Processor Count

Figure 9: Sensor Variation for Different Workloads

Figure 9 shows the effect on server request overhead[3] of varying the inter-access computation interval and the number of nodes reading a file. This experiment was performed on an Intel Paragon XP/S using a single input/output server controlling a RAID-3 disk array with a throughput of 3.1 MB/second. In Figure 9a, eight processors read the file and the PPFS server prefetches only sixteen KB ahead of the access stream. In Figure 9b, on the other hand, the PPFS server prefetches 256 KB ahead and clients wait on average 175 milliseconds in between each access. The PPFS server performance depends on the number of requests arriving at the server each second. In Figure 9a, the arrival rate varies from 27 to 54 requests per second. Similarly, in Figure 9b, the request arrival rate varies from 6 to 92 requests per second.

The sensors values in Figure 9 fall into three basic categories. As shown in the top of Figure 9a, most of the requests could result in cache misses coupled with long queuing delays where the server time exceeds ten milliseconds. A substantial increase in the amount of prefetching is required to alleviate this problem. When some of the requests result in cache misses, we see that the server time is between four and six milliseconds.[4] A

[3] Server request overhead is the time that a request spends on the PPFS server node. It includes cache check time, buffer copy overhead, and disk queuing times if the request is not in the server cache

[4] In Figure 9b, the startup transient lasts about sixty seconds before these cache misses occur regularly

moderate increase in the number of blocks prefetched should result in improved performance. Finally, at the bottom of Figure 9b, we see that when all of the requests can be serviced from the cache, the mean time spent on the PPFS server is less than one millisecond.

Table 5: Benchmark Selector Rules

Sensor Conditions	Policy Options
Quantitative Rules	
(large_server_times) & (many_read_requests)	Substantially Increase Prefetch Amount
(moderate_server_times) & (many_read_requests)	Moderately Increase Prefetch Amount
Quantitative Calibration	
(MEAN_SERVER_TIME > 8 MS) & (READ_REQUEST_COUNT >40)	Substantially Increase Prefetch Amount
(READ_REQUEST_COUNT > 40 & (MEAN_SERVER_TIME > 2 MS) & (MEAN_SERVER_TIME < 8 MS)	Moderately Increase Prefetch Amount

Based on the figure, we can develop the two simple selector rules shown in Table 5 for this benchmark access pattern. One rule detects when the prefetch parameters should be increased considerably while the other detects when the prefetch parameters should be increased slightly. To calibrate these rules for the Intel Paragon with a single RAID-3 disk array, we simply augment the selector table with the appropriate sensor values as shown at the bottom of the Table 5. When the calibrated selector table is used for an application that exhibits this access pattern, the steering infrastructure can detect poor PPFS server performance and increase the prefetch parameters appropriately.[5]

6. Related Work

Current work in parallel file systems centers on understanding application input/output requirements and determining how to consistently deliver close to peak input/output performance. This challenge necessitates re-examining the traditional interface between the file system and application.

[5] The rules in Table 5 are examples of a subset of the needed rules for this benchmark. A complete set of rules could also reduce the amount of prefetching performed when the sensors indicate that resources were being wasted.

Characterization studies have revealed a large natural variation in input/output access patterns. During the past two years, our group and others have used the Pablo input/output analysis software to study the behavior of a wide variety of parallel applications on the Intel Paragon XP/S [1,2] and IBM SP/2. We have determined from these application studies that high performance applications exhibit a wide variety of input/output request patterns, with both very small and very large request sizes, reads, and writes, sequential and non-sequential access, and a variety of temporal variations.

Given the natural variation in parallel input/output patterns, tailoring file system policies to application requirements can provide better performance than a uniformly imposed set of strategies. Many studies have shown this under different workloads and environments [5,6,7]. Small input/output requests are best managed by aggregation, prefetching, caching, and write-behind, though large requests are better served by streaming data directly to or from storage devices and application buffers. There are several approaches to application policy control; these can be grouped into systems that offer explicit policy control (e.g. SPIN from Bershad [19], exokernel from Engler [20], the Hurricane File System from Krieger [21], and Galley from Nieuwejaar [22]), and implicit policy control, via hints [4], expressive user interfaces (e.g., ELFS [7] and collective input/output as in del Rosario [23] and Kotz [24]), or intelligent modeling of file access (e.g., Fido from Palmer [25] and knowlege based caching from Korner [26]). Fido is an example of a predictive cache that prefetches by using an associative memory to recognize access patterns over time. Knowledge based caching has been proposed to enhance cache performance of remote file servers.

The second component of our research, dynamic performance based steering, has been used successfully in many contexts. A natural analog to explicit policy control is interactive steering, where the steering infrastructure extracts run time sensor information from an application, presents this information to the user who selects system or application policies, and actuates these policies to change application behavior. Falcon as in Gu [27] and SciChem from Parker [28] are two representative examples of this interactive approach.

In contrast to interactive steering environments, automatic steering environments do not require continuing user involvement. Instead, steering decisions are made automatically without user intervention. DIRECT [10], Falcon [29,30] and the Meta Toolkit [9] all provide automatic steering interfaces. DIRECT targets real time applications, a domain where the primary concern is validating that the system meets real-time constraints. This goal is different from run-time performance improvement, but the steering infrastructure is similar. Automated run-time steering is used in Falcon to select different mutual exclusion lock configurations based on the number of threads blocked on the lock [30]. The Meta Toolkit provides a framework for performing dynamic steering and provides special guards that help to maintain mutual exclusion of critical state variables [9] that may be changed during actuator execution. When an actuator is invoked, the appropriate guards are executed before the system module is modified.

7. Conclusions

The wide variety of irregular access patterns displayed by important input/output boսnd scientific applications suggests that optimizing application performance requires a judicious match of resource management policies to resource request patterns. Because the interactions between dynamic, irregular applications and system software change during application execution, we believe that the solution to this performance problem is adaptive file system policies that are controlled by user-level access patterns and by system-level performance metrics.

In this paper, we described a prototype of an adaptive file system and presented the results of experiments demonstrating the viability of this approach. This prototype, built upon on our PPFS user-level parallel file system. selects and configures file caching and prefetching policies using both qualitative classifications of access patterns and performance sensor data on file system responses.

In the coming months, we plan to more tightly couple automatic access pattern classification with performance steering. We are currently rounding out the prototype by extending PPFS to perform run time global access pattern classification and enhancing the performance-driven steering infrastructure.

References

[1] CRANDALL, P.E., AYDT, R.A, CHIEN, A.A., AND REED, D.A. Characterization of a Suite of Input/Output Intensive Applications. In *Proceedings of Supercomputing '95* (Dec. 1995).

[2] SMIRNI, E., AYDT, R.A., CHIEN, A.A. AND REED, D.A. I/O Requirements of Scientific Applications: An Evolutionary View. *In Fifth International Symposium on High Performance Distributed Computing* (1996).

[3] HUBER, J., ELFORD, C.L., REED, D.A., CHIEN, A.A., AND BLUMENTHAL, D.S. PPFS: A High Performance Portabie Parallel File System. *In Proceedings of the 9th ACM International Conference on Supercomputing* (Barcelona, July 1995), pp. 385-394.

[4] PATTERSON, R.H., GIBSON, G.A., GINTING, E., STODOLSKY, D., AND ZELENKA, J. Informed Prefetching and Caching. *In Proceedings of the Fifteenth ACM Symposium on Operating Systems Principles* (December 1995), pp. 79-95.

[5] KOTZ, D., AND ELLIS, C.S. Practical Prefetching Techniques for Multiprocessor File Systems. *Journal of Distributed and Parallel Databases* 1, 1 (January 1993), 33-51.

[6] KRIEGER, O., AND STUMM, M. HFS: A Flexible File System for Large-Scale Multiprocessors. In *Proceedings of the 1993 DAGS/PC Symposium* (Hanover, NH, June 1993), Dartmouth Institute for Advanced Graduate Studies, pp. 6-14.

[7] GRIMSHAW, A.S., AND LOYOT, JR., E.C. ELFS: Object-Oriented Extensible File Systems. In *Proceedings of the First International Conference on Parallel and Distributed Information Systems* (December 1991), p. 177.

[8] VETTER, J., AND SCHWAN, K. Models for Computational Steering. *In Proceedings International Conference on Configurable Distributed Systems* Annapolis (May 1996).

[9] WOOD, M.D. *Fault-Tolerant Management of Distributed Applications Using the Reactive System Architecture.* PhD thesis, Cornell University, January 1992. Available as technical report TR91-1252.

[10] GERGELEIT, M., KAISER, J., AND STREICH, H. Direct: Towards a Distributed Object-Oriented Real-Time Control System. In *Workshop on Concurrent Object-based Systems* (Oct. 1994).

[11] GU, W., VETTER, J., AND SCHWAN, K. An Annotated Bibliography of Interactive Program Steering. Tech. Rep. GIT-CC-94-15, College of Computing, Georgia Institute of Technology, 1994.

[12] HINTON, G.E. Connectionist Learning Procedures. *Artificial Intelligence 40* (1989), 185-234.

[13] HENDERSON, R.D. *Unstructured Spectral Element Methods: Parallel Algorithms and Simulations* PhD thesis, June 1994.

[14] HENDERSON, R.D., AND KARNIADAKIS, G.E. Unstructured Spectral Element Methods For Simulation of Turbulent Flows. *Journal of Computational Physics* 122, 2 (1995), 191-217.

[15] AGBU, P.A., AND JAMES, M.E. *The NOAA/NASA Pathfinder AVHRR Land Data Set User's Manual.* Goddard Distributed Active Archive Center, NASA, Goddard Space Flight Center, Greenbelt, 1994.

[16] NCSA. NCSA HDF, Version 2.0. University of Illinois at Urbana-Champaign, National Center for Supercomputing Applications, Feb. 1989.

[17] REED, D.A., AYDT, R.A., NOE, R.J., ROTH, P.C., SHIELDS, K.A., SCHWARTZ, B.W., AND TAVERA, L.F. Scalable Performance Analysis: The Pablo Performance Analysis Environment. In *Proceedings of the Scalable Parallel Libraries Conference*, A. Skjellum, Ed. IEEE Computer Society, 1993, pp. 104-113.

[18] REED, D.A., SHIELDS, K.A., TAVERA, L.F., SCULLIN, W.H., AND ELFORD, C.L. Virtual Reality and Parallel Systems Performance Analysis. *IEEE Computer* (Nov. 1995), 57-67.

[19] BERSHAD, B.N., SAVAGE, S., PARDYAK, P., SIRER, E.G., FIUCZYNSKI, M.E., BECKER, D., EGGERS, S., AND CHAMBERS, C. Extensibility, Safety and Performance in the SPIN Operating System. In *Proceedings of the Fifteenth ACM Symposium on Operating Systems Principles* (December 1995).

[20] ENGLER, D.R., KAASHOEK, M.F., AND JR., J.O. Exokernel: An Operating System Architecture for Application-Level Resource Management. *In Proceedings of the Fifteenth ACM Symposium on Operating Systems Principles* (December 1995).

[21] KRIEGER, O. *HFS: A Flexible File System for Shared-Memory Multiprocessors.* PhD thesis, University of Toronto, October 1994.

[22] NIEUWEJAAR, N., AND KOTZ, D. The Galley Parallel File System. In *Proceedings of the 10th ACM International Conference on Supercomputing* (May 1996). To appear.

[23] DEL ROSARIO, J.M., BORDAWEKAR, R., AND CHOUDHARY, A. Improved Parallel I/O via a Two-Phase Run-Time Access Strategy. *In IPPS '93 Workshop on Input/Output in Parallel Computer Systems* (1993), pp. 56-70. Also published in Computer Architecture News 21(5), December 1993, pages 31-38

[24] Kotz, D. Disk-directed I/O for MIMD Multiprocessors. *In Proceedings of the 1994 Symposium on Operating Systems Design and Implementation* (November 1994), pp. 61-74. Updated as Dartmouth TR PCS-TR94-226 on November 8, 1994.

[25] Palmer, M., and Zdonik, S.B. Fido: A Cache That Learns to Fetch. *In Proceedings of the 17th International Conference on Very Large Data Bases* (Barcelona, September 1991), pp. 255-262.

[26] Korner, K. Intelligent Caching for Remote File Service. *In Proceedings of the 10th International Conference on Distributed Computing Systems* (May 1990), pp. 220-226.

[27] Gu, W., Eisenhauer, G., Kraemer, E., Schwan, K., Stasko, J., and Vetter, J. Falcon: On-line Monitoring and Steering of Large-Scale Parallel Programs. Tech. Rep. GIT-CC-94-21, College of Computing, Georgia Institute of Technology, 1994.

[28] Parker, S.G., and Johnson, C.R. SciRun: A Scientific Programming Environment for Computational Steering. In *Proceedings of Supercomputing '95* (December 1995).

[29] Gheith, A., Mukherjee, B., Silva, D., and Schwan, K. Ktk: Kernel Support for Configurable Objects and Invocations. Tech. Rep. GIT-CC-94-11, College of Computing, Georgia Institute of Technology, Feb. 1994.

[30] Mukherjee, B., and Schwan, K. Improving Performance by Use of Adaptive Objects: Experimentation with a Configurable Multiprocessor Thread Package. Tech. Rep. GIT-CC-93-17, College of Computing, Georgia Institute of Technology, Feb. 1993.

NEXT
DOCUMENT

Towards Scalable Benchmarks for Mass Storage Systems

Ethan L. Miller
Computer Science and Electrical Engineering Department
University of Maryland Baltimore County
1000 Hilltop Drive
Baltimore, MD 21228*
elm@cs.umbc.edu
Tel: 410-455-3972
Fax: 410-455-3969

Abstract

While mass storage systems have been used for several decades to store large quantities of scientific data, there has been little work on devising standard ways of measuring them. Each system is hand-tuned using parameters that seem to work best, but it is difficult to gauge the potential effect of similar changes on other systems. The proliferation of storage management software and policies has made it difficult for users to make the best choices for their own systems. The introduction of benchmarks will make it possible to gather standard performance measurements across disparate systems, allowing users to make intelligent choices of hardware, software, and algorithms for their mass storage system.

This paper presents guidelines for the design of a mass storage system benchmark suite, along with preliminary suggestions for programs to be included. The benchmarks will measure both peak and sustained performance of the system as well as predicting both short-term and long-term behavior. These benchmarks should be both portable and scalable so they may be used on storage systems from tens of gigabytes to petabytes or more. By developing a standard set of benchmarks that reflect real user workload, we hope to encourage system designers and users to publish performance figures that can be compared with those of other systems. This will allow users to choose the system that best meets their needs best and give designers a tool with which they can measure the performance effects of improvements to their systems.

1. Introduction

Mass storage systems are used by many data centers around the world to store and manage terabytes of data. These systems are composed of both hardware from many vendors and storage management software, often from a different vendor. Each data center builds its own system, and no two are alike. How can two different mass storage systems be compared? Additionally, how can users gauge performance of planned systems?

We believe the introduction of a standard benchmark suite for mass storage systems will enable storage users to plan their systems in the same way that the SPEC and Perfect benchmarks allow users to compare different computing systems. In such suites, one or more of the benchmarks should sufficiently resemble a user's needs so that she can predict the performance on her own application. Similarly, data center personnel should be able to pick the metrics that most closely model their mass storage workloads, allowing some prediction of system performance without the need to experimentally configure multiple systems.

Mass storage system benchmarks must be portable, scalable, and reflective of real system workloads. Achieving portability will require limiting the scope of changes that must be made to the tests for different systems. Scalability is necessary because a mass storage system can hold from tens or hundreds of gigabytes to petabytes, and access patterns and file sizes will both vary greatly across this range of sizes. Finally, benchmarks must reflect real system workloads. Rather than rely on a single metric, a mass storage system benchmark suite must test both burst and sustained transfer rates and gauge the effectiveness of migration algorithms using several "typical workloads.

This paper proposes several candidate benchmarks for a scalable mass storage system benchmark suite. These programs are synthetic; they do not include code from actual user applications, but instead are based on access patterns observed on real mass storage systems. Some of the benchmarks generate access patterns similar to those of individual programs, typically requiring less than a day to run. Others model long-term access by many users to mass storage over periods of many days. Both types of benchmarks include programs that mimic "real world" access patterns as well as others that stress the system to find its limits, since both factors are important to mass storage system users.

While this paper contains concrete suggestions for mass storage systems benchmarks, there is still much work to be done. Using feedback from users of mass storage systems as well as vendors, it is our hope that this benchmark suite will become a standard that will ease the process of comparing many different options in storage system design.

2. Background

Research into mass storage system benchmarks builds on work in two different areas: the analysis of mass storage system usage patterns and the development of benchmark suites for different areas of computer systems. There are many papers that discuss benchmarks for areas ranging from processors to file systems to disks, providing a good foundation for deciding what a benchmark should (and shouldn't) do. However, there are relatively few quantitative papers on the usage patterns seen by mass storage systems; while many organizations study their systems to help plan for the future, these studies are rarely published.

Of the many papers that have been published about benchmarking, the most relevant to this research are those on file system benchmarks. These benchmarks fall into two broad categories: those that consist of a single program [1,2] and those that are built from many programs and are meant to model longer-term usage [3]. Additionally, many papers use ad hoc benchmarks to compare research file systems to already-existing file systems.

The few file system benchmarks that do exist are designed to test workstation-class file systems, both standalone and in a networked environment. Several of the benchmarks consist of a single program sending many read and write requests to the file system; such programs include IOStone [1], iozone [4], and bonnie [5]. These benchmarks are designed to gauge the maximum file system or disk system performance available to a single application over a short period of time. However, constant improvements in memory size and disk performance require scalable benchmarks; Chen's scalable disk benchmark [2] addresses this problem by scaling the workload to the system used. Still, this scaling is restricted to a single program.

NFSstone and the Laddis benchmark used by SPEC, on the other hand, are designed to model the activity of several programs and their effects on the file server. Rather than present a workload from a single client, these benchmarks can mimic an entire network of workstations. These benchmarks may be scaled by increasing the file request rate or the file size or both. Unfortunately, they are very specific in that they test the ability of a file server to respond to NFS requests. While NFS is a commonly-used file system, it is not clear that good performance for an NFS workload is necessarily the hallmark of a high-performance file system.

An even more complex benchmark, the Andrew file system benchmark [3] tests the entire file system by including operations such as file creation and deletion. However, the Andrew benchmark is not directly scalable, and still runs for only a few minutes or less. Clearly, a mass storage system benchmark must measure performance over longer periods of time as well as gauging the burst rates that the system can attain.

Many researchers gauging the performance of their new file systems create their own "benchmarks" that involve reading and writing many files. While such ad hoc benchmarks can provide comparisons between different file systems they require that the authors of such benchmarks run them on all of the systems being compared. This burden is not excessive for researchers because they often compare their research file system to one or two "real-world" systems that are already running at their site. However, this approach creates problems for "normal" users because most of them do not have access to the systems whose performance they wish to measure. While this approach is infeasible for standardized comparisons of many mass storage systems, the idea behind it is a good one: use a typical workload to measure performance. This method can be varied to find both performance under a normal load and the maximum load a system can handle.

Since synthetic benchmarks must mimic actual usage, knowing the access patterns exhibited by users of real systems is crucial. The system at the National Center for Atmospheric Research was studied in [6], and the National Center for Supercomputing Applications was studied in [7]. Both of these studies suggest that mass storage system performance must be measured over a period of days or weeks because that is the time scale over which file migration algorithms operate. Examining shorter periods is similar to running file system benchmarks that access a file that fits in memory — it provides a measure of peak bandwidth but does not give an indication of long-term performance.

These papers, along with other studies done for internal use at various organizations. provide a basis for designing benchmarks that test long-term mass storage system performance.

Short-term performance of large storage systems is also an important metric. Benchmarks that stress the file system as a single program would can model their I/O after the single program access patterns such as those reported in [8] and [9], which detail usage patterns exhibited by supercomputer and parallel computer applications, respectively. Short-term benchmarks might also include programs that stress the storage hierarchy. such as one that searches the relatively short file headers of hundreds of multi-megabyte files for a certain pattern.

3. Benchmark Characteristics

In order for a collection of mass storage system benchmarks to be useful, the benchmarks must have several features and characteristics. First and foremost, they must provide a good basis for comparing two systems that may be very different. They must also be portable and scalable, and should reflect real system workloads.

3.1 Suitability

Perhaps the most important quality for a benchmark suite is suitability. A benchmark must do two things to be useful. First, its results must bear some relation to the real use of a system. Typically, this is a predictive relation — the performance of a benchmark should be directly related to the performance of a real workload that the user will eventually run. Second, a benchmark suite should allow the comparison of two different systems in a manner more meaningful than "A is faster than B." While this is a good observation, it is almost always necessary to know *how much* faster A is relative to B.

Benchmark suites such as SPECint95, SPECfp95 and Perfect [10] are successful in large part because they use real programs (or fragments of them) to predict the performance of a computer system. A combination of several of the benchmark programs from these suites that closely mirrors the intended use of a system can usually be found, and the performance of the system on the real workload can be approximated by combining the system's scores on each individual benchmark program. Thus, benchmark reporting usually includes both a suite-wide average and a listing of the components' individual scores. The average is useful for gauging overall performance, while the individual listings allow the prediction of performance for a specific workload.

A relatively small suite of benchmarks works well for CPU benchmarks, but how will it work for mass storage systems? A benchmark suite may contain dozens of programs, but they are of no use if a user cannot assemble some of them into a workload that resembles her usage patterns. Fortunately, there are some general access patterns for mass storage systems that may be generated by a benchmark suite. These patterns will be discussed in Section 4.

3.2 Portability

The portability of a benchmark suite is another major concern for mass storage system benchmarks. CPU benchmarks are often portable because the interface to the system is at a high-level — programs are simply written in a high-level language such as C or FORTRAN. Running the benchmark on a new system requires is largely dependent on the existence of a compiler for the appropriate language being available for the system being tested. While there are may be other requirements for a portable CPU benchmark such as environment or operating system dependencies, building portable benchmark suits for CPUs is relatively well understood.

Portability of mass storage system benchmarks is another matter altogether. While mass storage systems tend to have the same functionality, they often have very different interfaces. Some systems require a user to explicitly request transfers to and from tertiary storage, while others do so automatically. Worse, the commands to effect such transfers are often different from system to system. As a result, mass storage system benchmarks will likely need to be customized to run on each individual system. To preserve portability, this customization should be limited to a few small pieces of code so that porting the benchmarks to new systems is not a difficult task. Nonetheless, there may need to be large changes in the benchmarks between systems. While it is straightforward to make a small change to read and write files via system calls or ftp, it may be more difficult to adapt a benchmark that assumes explicit transfers of files between tertiary storage and disk to a system that uses implicit transfers. These tradeoffs will be discussed in Section 4.

A second difficulty with portability of mass storage system benchmarks is the existence of different features on various mass storage systems. This issue is not present in CPU benchmarks — while an individual processor may not have a vector coprocessor or floating point unit, it *can* emulate those features using other instructions, albeit at a loss of speed. However, mass storage systems may have features that are simply not present elsewhere and that greatly improve performance. For example, one mass storage system might have the ability to compress files before storing them tape, while another lacks this feature. Should the two systems be compared without compression? The use of compression is likely to slow down the system that uses it, but it will also free up additional space. The decision of whether to include such features will be left to the benchmarker; as long as the settings of such relevant features are reported, a user can choose the appropriate benchmark results.

3.3 Scalability

The second goal of the benchmark suite is scalability. The suite must permit comparisons of two systems of roughly equivalent size, regardless of whether their capacity is 50 GB or 500 TB. On the other hand, comparing the performance of two mass storage systems of very different sizes makes little sense since the two systems will almost certainly have different workloads — a 50 GB storage system would not experience many repeated reads and writes of 1 GB files, though a 50 TB system certainly might.

Scaling the benchmarks can be done by a combination of two methods: increasing the request rate, and making individual requests larger. Increasing the request size reflects the larger data sets that necessitate larger storage systems. However, more storage space can also correspond to a larger user community or faster computers, both of which can increase request rate as well as allowing larger data sets. The TPC database benchmarks [10] follow this model, increasing request rate as the capacity of the storage system increases while keeping request size relatively constant.

Not all of the benchmarks need be scalable in order to provide a scalable benchmark suite, though. Clearly, some of the benchmarks must place a higher demand on the system as it becomes larger, but individual benchmarks need not. For example, a benchmark that mimics the behavior of a single program requesting a single gigabyte file might not change from a 50 GB system to a 50 TB system. Since this benchmark measures peak transfer bandwidth and nothing else, it does not have to scale up as the system becomes larger. However, other benchmarks must measure the performance of the system as a whole instead of focusing on short-term performance issues such as peak transfer rate. It is these benchmarks that must take parameters governing their behavior to allow them to model various workload levels. A benchmark measuring a storage system's ability to service clients, for example, must take the number of users as an input. For small systems, this number might be three or four. For larger systems, though, it could be several hundred. Similarly, average request size and an individual user's request rate will be different for different systems; these parameters must be customizable between benchmarks.

3.4 Feasibility

While mass storage system benchmarks share many characteristics with CPU and disk benchmarks, they also have limitations not suffered by CPU and file system benchmarks. CPU benchmarks usually have running times of a few hours or less, with many needing only a few hundred seconds to complete. Disk benchmarks may take longer, but still complete in well less a day for even the longest benchmarks. These time scales are too short for mass storage system benchmarks, however. Individual programs using a mass storage system may complete in a few hours or less, but long-term performance is just as important, and much more difficult to measure. The effects of a poorly-chosen file migration algorithm may not be apparent until several weeks have passed because the storage system's disk is not filled until then. Worse, policies governing file placement on tape may have little effect on overall performance until files are migrated from disk to tape and back, a process which could take several months before a significant number of files have taken the path.

Additionally, long-running benchmarks are difficult to use for tuning purposes. Seeing the effect of a faster CPU on a benchmark suite requires only an hour or two, while adding one more tape drive may not show performance improvement on a benchmark suite for days. This lack of responsiveness makes it likely that mass storage system benchmarks will be run on simulators rather than on real equipment at least some of the time; this requires the development of good simulators that model software systems and their quirks as well a:, hardware.

A second issue for mass storage system benchmarks is the existence of a system on which the benchmarks can be run. This is typically a simple matter for CPU benchmarks because the manufacturer usually has a system on which the benchmarks can be run. For expensive supercomputer systems, the manufacturer need only run the suite as part of the development process or even during the testing period for a new system. Since the benchmark suites take less than a day, the equipment cost is minimal. For mass storage systems, however, equipment cost is not as low. A system is usually built from components from several vendors, and the installation of the software to manage the storage is hardly trivial. The difficulty of assembling a storage system for benchmarks is another factor that makes it likely that a benchmark suite used for its predictive ability will be run on simulated rather than real hardware.

4. Proposed Benchmark Programs

Based on the analyses presented in several studies of mass storage systems [6,7] and the behavior of individual programs [8,9], we propose a collection of mass storage system benchmark programs. To assure their portability, the benchmarks use few file system features beyond reading, writing, file creation, file deletion and directory listings. Rather, they focus on the ability of the mass storage system to supply and store data. They are not restricted to reading and writing whole files, however; some of the benchmarks perform operations that model workstation file usage of large scientific files including partial file reads. Although such operations may not be supported efficiently by many mass storage systems today, our experience has shown that users viewing large data files often do not view the entire file.

The benchmarks in this collection fall into two broad categories: short-running benchmarks that highly stress the system to gauge its maximum performance, and long-running benchmarks that model long-term user behavior, allowing the testing of file migration algorithms and other long-term processes that cannot be measured by a single program that only runs for a few hours. It is our expectation that the long-running benchmarks will be primarily run on a simulation model of the mass storage system rather than on an actual system because of the time and expense involved in dedicating a storage system to a benchmark suite for more than a month.

4.1 Short-Running Benchmarks

One aim of the benchmark suite is to measure short-term performance of mass storage systems. Since these systems consist of both disks and tertiary storage devices such as tapes and optical disks, any benchmark suite must be capable of measuring the sustained performance of each of these parts of the system. Measuring the peak performance of the disk is straightforward, but measurements of tertiary storage device performance may be more difficult, particularly in systems that do not require explicit commands to move files between disk and tertiary storage.

The first program in the benchmark suite merely writes several large files and then reads them back. The number of files to be written and the size of the files is configurable, allowing users to scale up the benchmark to larger systems. This benchmark only tests peak sequential read and write performance; it does not attempt to gather any other file system metrics. Nonetheless, the peak performance of a file system on large sequential reads is of great interest to many users, necessitating the inclusion of such a benchmark.

A similar program can be used to measure the ability of a mass storage system to create and delete small files. As with the first program, the number and size of files are specified as parameters. Rather than merely create all of the files, though, this benchmark creates the files, lists the directory in which they were created, reads them in, and then deletes them. These operations stress other aspects of the mass storage system software, showing its performance on small file operations.

Another variation on the first program creates several large files and then reads only the first few blocks of each file, "searching" for a particular piece of data. This benchmark is similar to the access pattern exhibited by a user when she is looking through the headers of large data files.

The remaining "micro-benchmarks" model two types of real user behavior: workstation users accessing the mass storage system, and scientific programs using the storage system for input and output. Human users typically read a group of files over the span of several hours, perhaps performing a few writes during that time. While some files are read in their entirety, many are partially read as users look at slices through their data. Since this program is designed for measuring short-term performance, it only models a user's access to a single set of data over a relatively short period of time. Longer-term benchmarks that model user behavior are mentioned in Section 4.2. While this program only generates the workload for a single user, it is possible to run multiple copies of the program, generating a workload resembling that from multiple users.

Batch jobs using the storage system behave quite differently from human users. They almost always read entire files and perform more and larger writes than do humans [6], stressing the storage system in a different way. Programs such as out-of-core matrix decomposition and global climate modeling make excellent benchmarks because their I/O access patterns can easily be captured without the need to actually perform the computations called for in the programs [12]. Rather than actually factor a large matrix, the benchmark simply reads and writes the files in the same pattern as the real application . Similarly, the benchmark simulating global climate modeling does not do any actual modeling, but rather follows the same access pattern as the real program. This allows the benchmarking of a high-performance storage system without the need for a high-powered CPU to run applications. This is particularly important for planning purposes, since there may not yet be a computer that can run the program sufficiently fast — given the rate with which computers increase in processing power, a storage system that will become operational in eighteen months must deal with programs twice as fast as those running today.

The benchmarks listed in this section are generally useful for determining peak performance for bandwidth, request rate, or both. Combining the various benchmarks and running several copies of each allows users to customize the benchmark to their needs, matching the presented workload to what their installation will eventually support. However, these benchmarks are only good for measuring peak rates such as maximum bandwidth for reading files from tertiary storage or disk or the maximum rate at which a user may create small files. They do not measure any long-term statistics such as the efficiency of the file migration algorithms or the efficacy of tertiary storage media allocation.

4.2 Long-Running Benchmarks

A second class of benchmarks are those that generate multi-week workloads. Unlike CPUs and disks, mass storage systems exhibit activity with cycles considerably longer than a day. To measure the effects of file migration and differing sizes of disk cache for tertiary storage, benchmarks must run sufficiently long so that the disk fills up. Merely filling the disks is not sufficient, though, since the benchmark must also exhibit other user behaviors such as occasional file reuse after a long period of inactivity.

Fortunately, long-term benchmarks can be built from the short-term benchmarks mentioned in Section 4.1. Rather than running the benchmark programs alone or in small groups, though, long-term benchmarks run hundreds or thousands of instances of the same programs, possibly supplying different parameters for each run. This is done by a "master" program that controls the execution of all of the micro-benchmarks.

In addition to the usual issues of setting parameters appropriately, the master program may also need to throttle the execution of the benchmark suite. For example, a batch job that normally takes 200 minutes might take only 180 minutes because of improvements in the mass storage system. Rather than leave the machine idle for that period of time, the master benchmark coordinator should run the next benchmark "job." Of course, not all benchmarks need such throttling — it is unlikely that a human being will want to come to work earlier just because she finished a few minutes early the night before. Thus, the benchmark coordinator only throttles batch jobs, leaving the programs modeling human behavior unaffected. While this may not accurately reflect reality (people may actually *do* more work with a more responsive system), the question of gauging the changes in human response time are beyond the scope of this work.

Because of the length of time necessary to run a long-term benchmark and the expense of setting up and maintaining a system for the weeks necessary to complete its run, it is likely that most long-term benchmarks will be run on simulations of a mass storage system rather than on real hardware, as will be discussed in Section 4.3.

4.3 Running the Benchmarks

A major concern with a benchmark suite is the method used to run it. CPU benchmarks are simply run as programs, either individually in or in a group. The results of running the benchmark are the time taken to complete it and the amount of work the benchmark program did. A similar principle applies to file system and disk benchmarks because their behavior can be encapsulated in either one or a small group of programs.

Mass storage system benchmarks follow the same general guidelines but on a different time scale. Certainly, some benchmarks will consist of a single program or a small group of programs that finishes within a few hours. Since these benchmarks will model individual programs, they must intersperse "computation" with requests for file data. This presents a problem, however — a mass storage system's performance should not be dependent on the computation speed of its clients. To address this problem, benchmarks will avoid computation as much as possible, focusing on file I/O. Benchmarks will thus often be of the form "transfer all of this data, and then do nothing with it." While this format removes the effect of a slower CPU, it allows the file system to perform "optimizations" by not actually fetching or storing the requested data. This can be prevented by writing files with pseudo-randomly generated data, reading them back in, and checking the results by either using the same generator or computing the digital signature for the file and ensuring that it matches that computed for the original.

Workload generators that may run for many days present a different set of problems. If a system crashes in the middle of a one hour benchmark, the program can just be rerun from the start. This is not practical for benchmarks that may run for more than a month (though it may encourage mass storage system software vendors to improve the quality of their code...). Instead, the workload generator may be restarted so it begins with the next request after the last one that completed successfully. Of course, such an outage will adversely affect overall performance, since the time spent fixing the system counts towards the total time necessary to run the benchmark.

4.4 Benchmark Customization

Running the benchmark programs may require customization in the form of providing the appropriate calls to open, read, write, close, and perform other operations on files and directories. To facilitate customization, the benchmark suite uses a standard library across all programs to access the mass storage system. This library can contain real calls to a storage manager, as would be required for short-running benchmarks, or it can contain calls to a model of the storage system that returns appropriate delays. Since the interface to the storage system is localized to a single file, the benchmark suite can easily be ported to new architectures by modifying that library file.

Localizing the interface to a single file allows benchmarks to be widely distributed, and lessens the ability of manufacturers to "cheat" on the benchmarks by reducing the changes they may make to the benchmarks. It also facilitates the development of new benchmarks, since the programs may call a standard interface rather than requiring a custom interface for each system. It also encourages the development of a standard set of capabilities for mass storage systems because "special" functions are not exercised by the benchmarks and will not improve their performance. While this may sound restrictive, it will actually benefit users by ensuring that they will not need to modify their programs to run efficiently on different mass storage systems.

5. Evaluating the Benchmark Programs

The true test of benchmarks is their ability to predict system behavior: thus, we plan to test our benchmark suite on several systems to gauge how well its results match the actual performance of working systems. Because the designs presented in this paper are very preliminary, we expect that several rounds of benchmark tuning will be necessary before the suite is ready for wider distribution.

The basic testing method is similar to that of benchmarks in other areas: obtain performance measures from the benchmark by running it on several systems, and compare the results with the actual performance of the systems. This exercise is not as simple as it may seem, however, because no two mass storage systems have the same workload pattern. For a fair test, it will be necessary to select the most appropriate benchmark mix for a system without knowing in advance what performance to expect. Thus, our final test will be to run the benchmark on one or more systems before measuring performance and looking for correlation between predicted performance and real performance.

6. Future Work

The work on mass storage system benchmarks presented in this paper is still in its very early stages. By presenting these ideas to the mass storage system community at this point, we hope to get valuable feedback and direction for the construction of this benchmark suite. In particular, we hope that mass storage system users will contribute representative codes to be added to the collection.

Our first goal is to produce source code for several of the benchmarks mentioned in the paper and run them on different storage systems including workstation file servers as well as multi-terabyte tertiary storage-backed storage systems. Using the results, we plan to refine our benchmarks, producing a set of a dozen or fewer programs that generate workloads representative of those seen in production mass storage systems.

We are also building a simulation model of mass storage systems to allow the running of long-term benchmarks. When this model is complete, we will be able to examine long-term effects such as the tradeoffs between different file migration algorithms and performance gains from different sizes of disk cache for tertiary storage. Using the benchmark suite rather than a particular workload will allow us to come up with general guidelines for building mass storage systems rather than the site-specific advice common in the field today.

7. Conclusions

This paper presented design principles for building a benchmark suite for a mass storage systems with capacities ranging from tens of gigabytes to petabytes. The benchmark programs will be synthetic; while they will be based on access patterns observed on real mass storage systems, they will not include real code from actual user. Some of the benchmarks will generate access patterns similar to those of individual programs, typically requiring less than a day to run. Others will model long-term access by many users to mass storage over periods of many days. Both types of benchmarks will include programs that mimic "real world" access patterns as well as others that stress the system to find its limits, since both factors are important to mass storage system users. Using feedback from users of mass storage systems as well as vendors, it is our hope that this benchmark suite will become a standard that will ease the process of comparing many different options in storage system design.

References

1. A. Park and J. C. Becker, "IOStone: A synthetic file system benchmark," *Computer Architecture News* 18([illegible]) June 1990, pages 45-52.

2. P. M. Chen, and D. A. Patterson, "A New Approach to I/O Performance Evaluation - Self-Scaling I/O Benchmarks, Predicted I/O Performance," *Proceedings of the 1993 SIGMETRICS Conference on Measurement and Modeling of Computer Systems*, Santa Clara, CA, May 1993, pages 1-12.

3. J. H. Howard, M. L. Kazar, S. G. Menes, D. A. Nichols, M. Satyanarayanan, R. N. Sidebotham and M. J. West, "Scale and Performance in a Distributed File System," *ACM Transactions on Computer Systems* 6(1), February 1988, pages 51-81.

4. W. Norcott, IOzone benchmark source code, version 2.01, posted to comp.sources.misc, October 30, 1994. Available from `ftp://www.cs.umbc.edu/pub/elm/iobenchmarks/iozone01.`

5. T. Bray, Bonnie benchmark source code, 1990. Available from `ftp://www.cs.umbc.edu/pub/elm/iobenchmarks/bonnie.sh`.

6. E. L. Miller and R. H. Katz, "An Analysis of File Migration in a UNIX Supercomputing Environment," *USENIX - Winter 1993*, San Diego, CA, January 1993, pages 421-434.

7. D. W. Jensen and D. A. Reed, "File Archive Activity in a Supercomputer Environment," Technical Report UIUCDCS-R-91-1672, University of Illinois at Urbana-Champaign, April 1991.

8. E. L. Miller and R. H. Katz, "Input/Output Behavior of Supercomputer Applications," *Proceedings of Supercomputing '91*, Albuquerque, NM, November 1991, pages 567-576.

9. D. Kotz and N Nieuwejaar, "File-System Workload on a Scientific Multiprocessor," *IEEE Parallel and Distributed Technology* 3(1), Spring 1995, pages 51-60.

10. M. Berry, et. al, "The PERFECT Club Benchmarks: Effective Performance Evaluation of Supercomputers," *International Journal of Supercomputer Applications* 3(3), 1989, pages 5 - 40..

11. Transaction Processing Perf. Council. Details available from ***http://www.tpc.org***

12. E. L. Miller, *Storage Hierarchy Management for Scientific Computing*, Ph.D. dissertation, University of California at Berkeley, April 1995. Also available as technical report UCB/CSD 95/872.

NEXT
DOCUMENT

Queuing Models of Tertiary Storage[1]

Theodore Johnson
Dept. of CISE, University of Florida
AT&T Research
ted@cis.ufl.edu

Abstract

Large scale scientific projects generate and use huge amounts of data. For example, the NASA EOSDIS project is expected to archive one petabyte per year of raw satellite data. This data is made automatically available for processing into higher level data products and for dissemination to the scientific community. Such large volumes of data can only be stored in robotic storage libraries (RSLs) for near-line access. A characteristic of RSLs is the use of a robot arm that transfers media between a storage rack and the read/write drives, thus multiplying the capacity of the system.

The performance of the RSLs can be a critical limiting factor of the performance of the archive system. However, the many interacting components of an RSL make a performance analysis difficult. In addition, different RSL components can have widely varying performance characteristics. This paper describes our work to develop performance models of a RSL. We first develop a performance model of a RSL in isolation. Next, we show how the RSL model can be incorporated into a queuing network model. We use the models to make some example performance studies of archive systems.

The models described in this paper, developed for the NASA EOSDIS project, are implemented in C with a well-defined interface. The source code, accompanying documentation, and also sample JAVA applets, are available at:

http://www.cis.ufl.edu/~ted/

Introduction

Large scale scientific projects generate and use huge amounts of data. For example, the NASA EOSDIS project is expected to archive one petabyte per year of raw satellite data [KBCH94]. This data is made automatically available for processing into higher level data products and for dissemination to the scientific community (see, for example, the reports in [ESDIS]). Automatic management of such large data sets requires the use of tertiary storage, typically implemented using *robotic storage libraries* (RSLs). In addition to EOSDIS and related projects, many organizations and scientific disciplines make use of mass storage archives (for example high energy physics [Lu95] and digital libraries [CoHu93]).

[1] This research was funded by a grant from NASA #10-77556

The database community has also become interested in the use of RSLs [DSF94,CHL93,Sequoia2k,Sa95]. This interest is motivated in part by scientific database problems such as EOSDIS. Another motivation for integrating RSLs with on-line database systems is to facilitate data warehousing.

Tertiary storage is required when the managed data set becomes too large to store economically with conventional magnetic disk devices. The point at which tertiary storage becomes necessary is an economic tradeoff. Currently, it seems that tertiary storage is needed to manage more than a terabyte of data. A RSL is much slower than magnetic disk storage, and data access latencies can run into minutes even on unloaded systems. However, RSL-resident data can be accessed automatically. Hierarchical storage management systems, such as Unitree, Filestore, and Amass, provide the illusion that the RSL is an extension of the file system. Access to archived data incurs a short delay. The storage capacity of a data system can also be increased by using off-line storage -- i.e. tape racks with human operators. Access latencies with off-line storage can be very large, ranging into hours or days, but the data storage capacity is limited only by the size of the warehouse that one can afford to rent. Since RSL provides data volumes and access latencies between those provided by on-line and off-line storage, it is often referred to as *near-line storage*. A cost analysis of on-line, near-line, and off-line archives can be found in [KGT90].

A characteristic of RSLs is the use of removable media and a robot arm. The removable media (e.g. magnetic tape, optical disk, etc. are normally located in a storage rack. To service a request for a file, the robot arm fetches the proper media from the storage rack and delivers it to a read/write drive. The media is accessed in the normal way to fetch the file. Finally, the media is returned to the storage rack. The capacity of RSL is the product of the capacity of the media and the size of the storage rack. Recent magnetic tapes have a data capacity on the order of 10 Gbytes, and storage rack sizes range from 10 to 1000 media (approximately). The time to fetch and mount the media which holds the requested file can be a large component of the access latency.

The performance of the RSLs can be a critical limiting factor of the performance of the archive system. Given the high data request rates expected for EOSDIS, attention to handling these requests efficiently is critical [KBCH94,ESDIS]. However, the many interacting components of a RSL make a performance analysis difficult. In addition, different RSL components can have widely varying performance characteristics.

This paper describes our work to develop performance models of tertiary storage. We first develop a performance model of a RSL in isolation. Next, we show how the RSL model can be incorporated into a queuing network model. Finally, we model fork-join jobs to study the tradeoffs of using multiple devices. We use the models to make some example performance studies of archive systems.

The models described in this paper, developed for the NASA EOSDIS project, are implemented in C with a well-defined interface. The source code, accompanying documentation, and also example JAVA applets, are available through:

http://www.cis.ufl.edu/~ted/

Previous Work

Considerable work has been done to develop performance models of mass storage. Rahm [Rh92] presents a simulation study of a database system with a hierarchy of storage devices. Ramakrishnan and Emer [RE89] present a queuing model of a client/server file system. Drakopoulos and Merges [DM92] present a closed queuing model of a client/server storage system with hierarchical storage. Kelly, Haynes, and Ernest [KHE91] discuss a benchmark for network storage systems. Hauser, Rivera, and Thoma [HRT91] discuss the performance of their networked WORM server.

Some work has been done to characterize the performance of mass storage devices. Waters [Wa74] presents a validated model of seek times in hard disk drives. More recently, Ruemmler and Wilkes [ReWi94] present a detailed model of a modern disk drive, and discuss the difficulties inherent in I/O modeling. Christodoulakis and Ford [CF88] and Christodoulakis [Ch87] present analytical performance models of optical drives. Chinnaswamy [Ch92] presents performance models of a streaming tape drive to investigate the benefit of a cache.

Models of disk arrays resemble the models presented in this paper in several aspects. Burkhard, Claffy, and Schwarz [BCS91] present a simulation study of a disk array scheme. Lee and Katz [LK93] and Yang, Hu and Yang [YHY94] present analytical models of disk arrays. Chen et al. [CLGKP94] and Thomasian [Th95] present surveys of research in RAID modeling.

Several authors have modeled a RSL. Butturini [Bu88] presents the results of a simulation study of an optical disk jukebox system. Hevner [He85] presents a model of an optical jukebox that is used for a database application. Howard [Ho92] gives a performance model for data duplication from an archive. Finestead and Yeager [FY92] give performance measurements of a Unitree file server at the National Center for Supercomputer Applications. Hull and Ranade [HR93] present measurements of tape loading and unloading, and of data throughput, in a tape silo. Bedet et al. [Be93] discuss the results of a detailed simulation model of the Goddard DAAC. Pentakalos, Menasce, Halem, and Yesha [PMHY95] develop a queuing network model that incorporates a RSL. Daigle, Kuehl, and Langford [DKL90] present a queuing model of an optical disk jukebox. Golubchik, Muntz and Watson [GMW95] analyze tape striping on an RSL.

The analyses most closely related to the one in this paper are [PMHY95,DKL90,GMW95]. The analysis in [DKL90] gives a detailed model of access times to data on an optical platter. However, only one drive is permitted and contention for the robotic arm is not modeled. In [PMHY95], the authors present a detailed model of a data center, incorporating RAID disk caches and user computation. However, the authors assume that contention for the drives in the

RSL is negligible, and model the RSL as a delay server. Contention for the robotic arm due to batch arrivals is modeled in [GMW95], but contention between jobs is not modeled.

The contribution of this work is to present a validated model of a RSL that accounts for batch arrivals, multiple drives, contention for the robotic arm, and realistic operation. We show how the model can be used to make a variety of data layout and device comparison studies. Finally, we show how to incorporate the RSL model into a queuing network model.

Model of a Robotic Storage Library

Our model of a RSL is illustrated in Figure 1. Previous studies of mass storage archive log files (see, for example, [Jo95a,DKL90]) indicate that requests to a mass storage device come in batches. This study has been corroborated by our studies of access to preliminary versions of the EOSDIS archives (the V0 archives) [Bedet96, DunhamNorth96]. As a result, our RSL model uses batch arrivals.

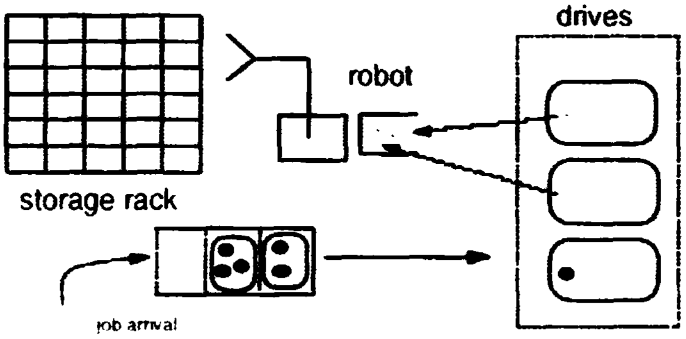


Figure 1

A user requests that f files be loaded into on-line storage, and these files are distributed over m media in the RSL. The request is satisfied when every file has been loaded into on-line storage. So, a user *request* consists of *m jobs*, each of which must be completed before the request in finished. A RSL consists of n_d drives, each of which can read or write any of the media in the RSL[2], a storage rack containing the removable media, and a robot arm for transferring the media between the drives and the storage rack. The model of a RSL is illustrated in Figure 1.

[2] In some installations, a subset of the drives are designated as read-only or write-only We will address this complication in a later version of the model.

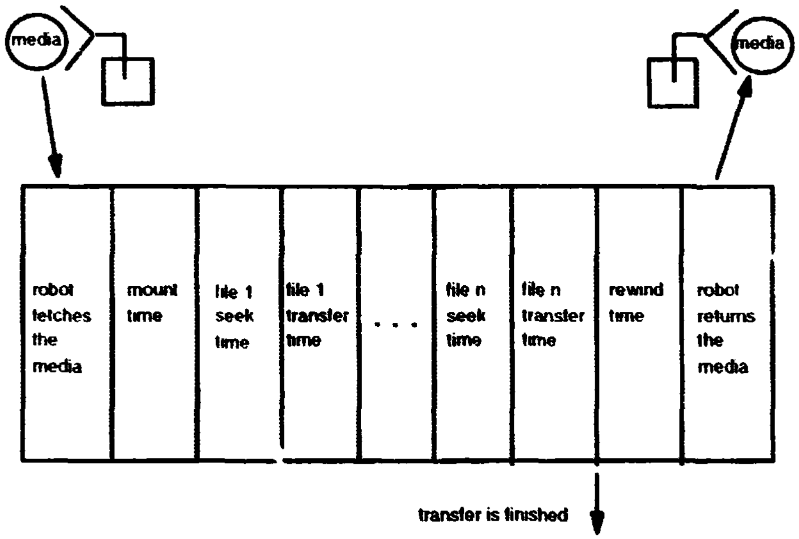


Figure 2

The steps taken by a drive in retrieving files from a media is illustrated in Figure 2. When a request arrives, its jobs are placed in the job queue. If there are jobs in the RSL queue and a drive is idle, the drive allocates one of the jobs for execution. First, the robot arm fetches the appropriate media from the storage area and loads it into the drive. If the robot arm is busy serving other drives, the drive must wait for service. After the media is brought to the drive, it must be mounted. For every file of interest on the media, the drive must seek to the start of the file, spend a settling time for precise positioning and opening communications channels, and then transfer the file to on-line storage. After all files have been transferred, the media is rewound and returned to the storage rack by the robot arm. However, the job is finished once all of the files have been transferred.

In the next section, we briefly discuss our analytical performance model of a RSL system (a more detailed discussion can be found in [Johnson96]). In this preliminary model, we make the following assumptions:

- Requests arrive in a Poisson process.

- The distribution of the number of media per request and the number of files per request must be specified. In the model discussion, we assume that the number of media per request and the number of files per request have geometric distributions. These can be replaced by user-specified distributions (e.g. empirically determined), but at the cost of requiring the user to specify more parameters.

- The RSL can contains one robot arm. The robot arm can access every media, and every drive.

- Every drive can read and write every media.

- Requests (i.e., jobs) are serviced first-come-first-serve[3].

- The service for a request is completed when the last file of the batch has been read (written).

- Network or communication channel contention is not significant[4].

- Service times at the drives are independent.

Analytical model

A RSL presents many difficulties for performance modeling, including batch arrivals, multiple servers, derived parameters, and interacting components. The primary component of the RSL model, the M^x/G/c queue, has been studied and solved in the literature [Tijms94]. Solving the actual M^x/G/c queue is intractable, so the solution technique is to interpolate between the results for the M^x/M/c queue and the M^x/D/c queue using the coefficient of variation of the service time as the interpolation parameter. Because of the potential complexity of the batch arrival distributions, we do not use explicit (i.e., generating function) formulas. Instead, we numerically solve the recurrence equation that defines the state occupancy probabilities. If the occupancy probabilities of the first N states must be computed for an error bound of e, then solving the M^x/M/c queue requires $O(N^2)$ time and solving the M^x/D/c queue requires $O(N^3)$ time.

Fortunately, we can take advantage of the nature of the problem to speed up the solution times. The state occupancy distributions eventually converge to a geometric distribution (i.e., $p_N = t_0 \, p_{N-1}$). Therefore the recurrence equations only need to be solved up the first N_0 states, and the remainder can be computed using the t_0 ratio (or perhaps the performance metrics can be computed directly). N_0 depends primarily on the distribution of the size of the batch arrival. Fortunately, the batch arrival distribution will have a short tail -- one cannot request that more media than exist in the storage rack be mounted, and usually only a few media are required to

[3] A simple optimization is to load files for all requests once a media has been mounted. We assume this situation has a negligible impact on performance in this model.

[4] Potential model users indicated that communication channel contention is not a problem for their systems. Communication contention can be incorporated into the seek times or mount times using standard techniques [Ka92].

satisfy a request. By using these tricks, we implemented batch queue solvers that are fast enough to be incorporated into a higher level model which calls them many times.

We model the RSL as an $M^X/G/c$ queue -- that is, a queue with Poisson batch arrivals, general service time distribution, and c servers. The parameters of a $M^X/G/c$ queue are:

- Arrival rate
- Mean service time
- Coefficient of variation of service time
- Batch size distribution
- Number of servers

All but the service time distribution are input parameters, so our analysis is focused on how to compute the expected service time, E_d and the coefficient of variation cv_d. To compute queue length distributions and expected waiting times properly, we need to compute the time that a drive is unable to serve other jobs per media that it serves. This period includes the time to fetch the media, mount it, seek to each file, transfer each file, rewind and eject the media, and return it to the storage rack. We will incorporate the time to return the media as part of the media fetch time, so we have:

drive service = (robot fetch)+(mount time)+(seek time)+(transfer time)+(rewind time)

Since we are interested in the response time of the last job in the batch to finish (i.e., instead of the average job), we need to modify the response time computation. An efficient algorithm for computing the response time of the last job in the batch is given in [Ka92]. The modified $M^X/G/c$ queue provides the batch response time R_{batch}, the drive utilization r_d, and $p_d(0),\ldots,p_d(n_d\text{-}1)$, the probability that $0,\ldots,n_d\text{-}1$ servers are busy on a request arrival (where n_d is the number of drives in the RSL). The effective drive service time (and $p_d(.)$) depends on the robot response time, which in turn depends on $p_d(.)$. Finally, the job is finished when the last file has been transferred, there is no need to wait for the tape rewind. Therefore:

$$R_{\text{request}} = R_{\text{batch}} - E_{\text{rwd}}$$

For more information about the derivations in the model, please see our other report [Johnson96].

Interfaces

The RSL solver consists of a number of modules, mostly consisting of procedures to solve $M^X/G/c$ queue. Figure 3 shows a map of the procedure calls. The functions rslsolve and rslsolve_f are the entries to the RSL solver.

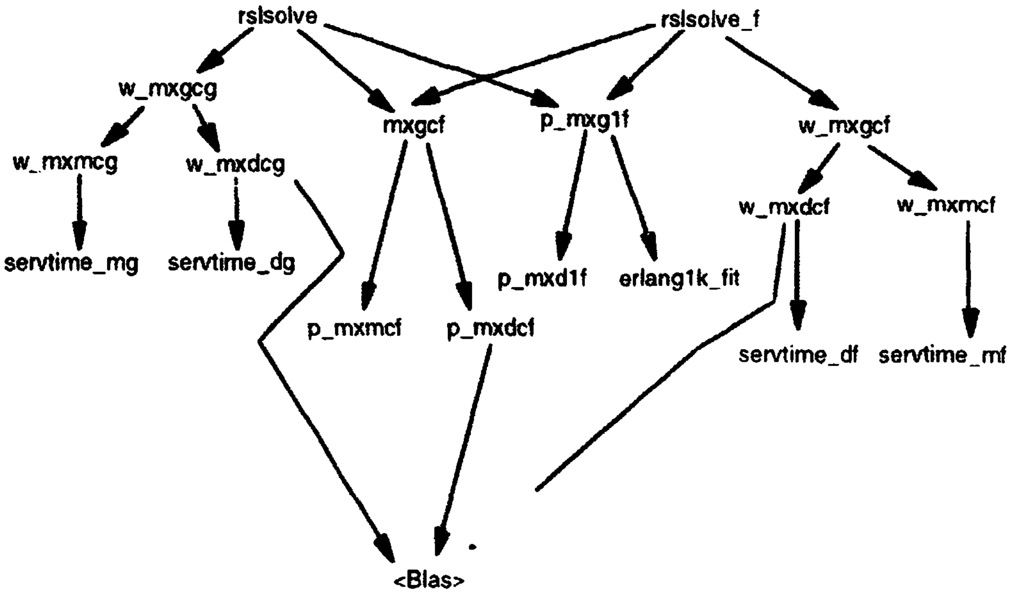


Figure 3

The prototype for the rslsolve function is

```
void rslsolve(float l,float fr,float mr,int nd,int nr,float,Etr,
 float Vtr,float tmt,float Xb,float Esz,float Vsz,float Erwd,
 float Vrwd,float (* seekfun)(int,float*,float*),int ncust,
  float *drho, float *dR, float *dW, float* dRv, float *dWv,
  float *rrho,float *rR, float *rW,float *dmu, float *dV,
  float *basemu,float *drbusy,int DEBUG)
```

where the input parameters are:

- l is the arrival rate.
- fr is the average number of files per request.
- mr is the average number of media per request
- nd is the number of drives in the RSL.
- nr is the number of robot arms.
- Etr is the is the average robot fetch time.
- Vtr is the variance of the robot fetch time.
- tmt is the media mount time.
- Xb is the transfer rate.
- Esz is the average file size.
- Vsz is the variance in the file size.
- Erwd is the average tape rewind and unmount time
- Vrwd is the variance in the average tape rewind and unmount time.
- moments of the seek time, given that nfiles are loaded.
- DEBUG is set true to print a trace.

And the output parameters are:

- drho is the drive utilization
- dR is the batch job response time
- dW is the batch job waiting time
- dRv is the variance in job response time
- dWv is the variance in job waiting time
- rrho is the robot utilization
- rR is the avg. robot response time
- rW is the avg robot waiting time
- dmu is the drive service time
- dV is the variance of drive service
- basemu is the base drive service time
- drbusy is an array where drbusy[i] is the long-term probability that *I* drives are busy. *0* <= *I* <= *nd-1* (drbu. ; must point to the storage location for the array when the call is made).

The rslsolve function assumes that the number of media per request and the number of files per media have a geometric distribution. In the rslsolve_f function, the user supplies the distribution of the number of media per request, but the number of files per media has a geometric distribution. The prototype for the rslsolve_f function is:

```
void rslsolve_f(float l,float fr, int nd,int nr,float,Etr,
 float Vtr,float tmt,float Xb,float Esz,float Vsz,float Erwd,
 float Vrwd, float *bd, int bmax,
 float (* seekfun)(int,float*,float*),int ncust,
  float *drho, float *dR, float *dW, float* dRv, float *dWv,
  float *rrho,float *rR, float *rW,float *dmu, float *dV,
  float *basemu,float *drbusy,int DEBUG)
```

where

- bd is an array where bd[i] is the probability that a request requires *I* media.
- bmax is the largest number of media required to service a request.

Validation Study

We wrote a simple RSL simulator. The simulation accepts batch arrivals, requires that a robot unload and fetch a media before a drive can service a job, handles multiple drives, and accounts for media rewind times. The drive service time, except for the robot arm component, is sampled from an Erlang distribution.

We used the following values of the parameters in the validation study:

- fr = 20.
- bd[.] : Geometric distribution.
- nd = 4.
- Etr = 10.0 seconds.
- Vtr = 10.0.
- tmt = 10.0 seconds.
- X_b = 1.0 Mbyte/sec.

We ran four sets of experiments to test the model. In the ``large files" experiments, Esz=50, Vsz=100, Tfs=50, and Tsl=1. In the ``small files" experiments, Esz=5, Vsz=10, Tfs=20, and Tsl=2. We tested the model with mr=2 and mr=6.

The results of the validation study are shown in Figures 4 through 7. In each case there is close agreement between the analytical and the simulation models. The most difficult case is when the files are small and distributed over an average of six media, because the robot fetch times constitute a large portion of the drive service times (about 22% of the total drive service time when the robotic arm waiting time is added). However, the analytical model is accurate enough to predict response times and drive utilizations. Charts comparing analytical and simulation drive utilizations are shown in Figures 8 and 9.

Performance Study

A performance model is useful for studying implementation alternatives. In this section, we present two sample performance studies based on the RSL model.

Clustering

Conventional wisdom holds that striping or declustering is necessary for obtaining high transfer rates from tertiary storage (by making use of parallel I/O). So, one should spread the files of a typical request around as many media as possible. Conventional wisdom also holds that swapping media is a source of great inefficiency in RSL access, so that one should try to ensure that the files of a typical request are placed on as few media as possible.

Neither argument is convincing, unless one has a predictive performance model. We ran the "small files" experiment with mr ranging between 1.2 and 8. In Figure 10, we plot the response time of a request against the number of media per request for varying arrival rates (A similar chart can be found in an analysis of tape striping [GMW95]). For low arrival rates, setting mr to approximately nd produces the best results. When lambda=.0001, setting mr =3 results in a 22% lower response time than setting mr=1.2. For high arrival rates, setting mr=2 gives lower response times than other choices.

In Figure 11, we plot the drive utilization against mr for varying arrival rates. Increasing mr causes a linear increase in the drive utilization. As the arrival rate increases, it becomes less likely that all nd drives are available to service the request. So, distributing the files over a smaller number of media reduces queuing delays. If the demand on the RSL is expected to be close to the device's capacity, then mr should be small to increase the maximum throughput of the device.

The question of whether to cluster or decluster the files on the media can be summarized as:

- If the expected drive utilization is low [PMHY95] and fast response is important, then declustering can be a good strategy. However, the decrease in transfer times must be larger than the increase in queuing delays.

- If high throughput is important, clustering is a good strategy.

Number of Drives

Many RSLs allow the user to install a ranging number of drives. Adding drives to a RSL can improve the performance of the device. But after a threshold, adding drives does not significantly improve performance.

We ran a sample study using the ``small files" parameters and four media per request. In Figure 12, we plot the response time versus the arrival rate for a number of drives varying between 2 and 8. Adding a drive significantly improves performance up to four drives, but gives less benefit after four drives. In Figure 13, we plot the drive utilization against the arrival rate. Adding a drive to the RSL increases the capacity of the device. However, the robot arm will start to become a bottleneck. This can be seen in the non-linear increase in utilization of some of the curves, for example for nd=8.

Computing Response Times for Particular Jobs

The rslsolve function computes the response time for an average request. However, it is often necessary to compute the expected response time for a particular request (with a particular number of media to be accessed, etc.). The servtime* routines use information computed by rslsolve to compute response times for particular requests. The prototype for the servtime_mf function is:

```
void servtime_mf(float* bd,float bmax,float* pr,float m, int c,
 float* Eserv, float* M2serv)
```

where the input parameters are:

- bd - batch arrival distribution.
- bmax - largest batch arrival.
- pr - pr[k] is the probability that k servers are busy. $0 <= k < nd$
- m -service rate.
- c - number of servers.

And the output parameters are:

- Eserv - average request service time.

- M2serv - 2nd moment of request service time.

The function servtime_mf assumes that the service time has an exponential distribution. A similar routine, servtime_df, assumes that the service time is deterministic. Res:.lts for general service time distributions are obtained through interpolation.

We consider the following application. A large scale data center is likely to have multiple RSLs. The devices might be acquired to handle data center growth, or multipie small devices might be less expensive than a single large device. In this section, we discuss an approximation to the request response time when the request is served by multiple RSLs.

If a request is services by two different RSLs, the request is finished when both devices have completed their part of the request. Since we assume that requests are independent, we need to analyze a fork-join queue with interfering traffic. Thomasian and Tantawi [ThTa94,Th96] have found that a good approximation to the response time of the fork-join job is to take the maximum of the response times of each device (we assume an Erlang distribution on the response time when computing order statistics).

For an experiment, we applied the ``large files" workload to two RSLs, with both receiving the same arrival rate. We considered a request that required files from six media. In Figure 14, we plot the response time of this request against the arrival rate, and varied the number of media that must be loaded from each device. The results show that when the load on the tertiary storage devices is low, it is better to divide the request evenly between the two devices. But, when the load is high it is better to use one device only. The reason for this result is that splitting the request between the two devices provides parallel I/O, but if the request load is high, then the variance in the response times becomes large. Thus, the decision to allocate files so that most requests use only one device or that most requests use both devices depends on the expected load placed on the devices.

Queuing Network Model

A mass storage data system consists of many components in addition to the RSLs. Typical hierarchical storage management systems use a database to track file to media location mappings, and maintain a sizeable staging and caching area. The computing centers that use tertiary storage often have large scale computing tasks. For example, EOSDIS archives must perform *product generation* to filter, correct, remap, and fuse satellite images (see the reports in [ESDIS], and also the discussion in [PMHY95]).

To capture the effects of RSLs in computing systems, we need to integrate the RSL model into a queuing network model. The typical approach for incorporating devices with unusual response time characteristics into a queuing network model is to use mean value analysis (MVA), and develop a MVA recurrence for the device in question [Ka92]. However, it is difficult to develop such a recurrence even for multiple server devices. Therefore, we take the approach of integrating the open RSL queue into a MVA model.

Although the RSL model solver is fast (about 2 seconds of execution time), an exact MVA solver requires an iteration over every possible population vector. If the population is large and there are many job classes, solution times become intolerably large. We instead used an approximate MVA solver, making use of Schweitzer's approximation on queue lengths and Bard's approximation for the load dependent servers [Ka92]. The approximate MVA solver built using these approximations compute the throughput at each iteration, which we use as the arrival rate at the RSL (after scaling by the visit ratio).

The function that solves the queuing network model is closedqn_rsl. Its prototype is:

```
int closedqn_rsl( int M, int K, double D[MaxM][MaxClass], double* N,
 int* servtype, double alpha[MaxLD][MaxPop], struct rslparamstr* rslparam,
 double visit[MaxM][MaxClass],
   double lam_out[MaxClass], double Rloc[MaxM][MaxClass], double* U)
```

where the input parameters are:

- M - number of servers.
- K - number of classes.
- D - D[k][r] is the service demand of a class r job at server k.
- N - N[r] is the number of customers of class r.
- servtype - servtype[k] encodes server k's type, and possibly points to additional parameters.
- alpha - alpha[l][*] are the service rate multipliers of load dependent server l.
- rslparam rslparam[l] is a structure containing the service parameters of RSL l.
- visit - visit[k][r] is the number of times a class r job visits server k.

And the output parameters are:

- lam_out - lam_out[r] is the throughput of class r jobs.
- Rloc - Rloc[k][r] is the residence time of a class r job at server k, per visit.
- U - U[k] is the utilization of server k.

Incorporating an open queuing model into a closed queuing network can introduce inaccuracies (we implemented some heuristic corrections). To test the accuracy of the approximate MVA model, we simulated a computer system with a RSL and three other queuing devices. The requests to the RSL used the ``large file'' workload, and every customer submits a single request to the RSL per task execution. There are three queuing devices, with per-task service demands of 250, 400, and 350 units of work, respectively.

We plot the response time of the RSL against the number of customers in the system in Figure 15 for a sleep time of 5000 and 9000. The model is accurate even for a small number of customers. However, the accuracy declines when the number of customers is large and the sleep time is small. This problem is occurring because one of the queuing devices is saturated, and the approximate MVA solver becomes inaccurate in these situations.

Conclusions

We have developed an analytical model of a robotic storage library and validated the model by comparison to simulations. The RSL consists of a storage rack for removable media, a set of drives that read and write the media, and a robotic arm that transfers the media between the storage rack and the drives.

The RSL model can be used for many useful studies. We provide examples of data layout and device selection studies. A RSL is used as a part of a larger computing system. We incorporated the RSL solver into an approximate MVA queuing network model, and validated the model by comparison to a simulation.

We have developed this model to support NASA's EOSDIS on-line archiving efforts. Future work will be directed towards refining the model and providing studies useful to archive sites. This work includes further model refinements, and encapsulating the queuing model solvers into Java applets that perform particular analyses.

Acknowledgments

We'd like to thank Ben Kobler and Chris Daly of NASA GSFC, and Bob Howard of Hughes for their comments, and Alex Thomasian for his advice regarding fork-join jobs.

Bibliography

[Bedet96] T. Johnson and J.J. Bedet. Analysis of the access patterns at the GSFC Distributed Active Archive Center. In *Proc. Fifth Goddard Conf. on Mass Storage Systems and Technologies, 1996*

[Be93] J.J. Bedet et al. Simulation of a data archival and distribution system at GSFC. In *Goddard Conference on Mass Storage Systems and Technology*, NASA CP 3262, pages 139-160, 1993.

[BCS91] W.A. Burkhard, K.C. Claffy, and T.J.E. Schwarz. Performance of balanced array schemes. In *Mass Storage Systems Symposium*, pages 45-50, 1991.

[Bu88] R.S. Butturini. Performance simulation of a high capacity optical disk system. In *Mass Storage Systems Symposium*, pages 147-153, 1988.

[CHL93] M.J. Carey, L.M. Haas, and M. Linvy. Tapes hold data too: Challenges of tuples on tertiary store. In *Proc. ACM SIGMOD*, pages 413-418, 1993.

[CLGKP94] M.P. Chen, E.K. Lee, G.A. Gibson, R.H. Katz, and D.A. Patterson. RAID: High-performance reliable secondary memory. *Computing Surveys*, 26(2):145-185, 1994.

[Ch92] V. Chinnaswamy. Analysis of cache for streaming tape drive. In *Proc. NASA Goddard Conference on Mass Storage Systems and Technology*, NASA CP-3198, Vol. I, pages 299-310, 1992.

[Ch87] S. Christodoulakis. Analysis of retrieval performance for records and objects using optical disk technology. *ACM Transactions on Database Systems*, 12(2):137-169, 1987.

[CF88] S. Christodoulakis and D.A. Ford. Performance analysis and fundamental performance tradeoffs for clv optical disks. In *ACM SIGMOD*, pages 286-294, 1988.

[CoHu93] R.A. 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*, NASA CP-3262, pages 15-18, 1993.

[DKL90] J.N. Daigle, R.B. Kuehl, and J.D. Langford. Queuing analysis of an optical disk jukebox-based office system. *IEEE Trans. on Computers*, 39(6):819-828, 1990.

[DSF94] J. Dozier, M. Stonebraker, and J. Frew. Sequoia 2000: A next-generation information system for the study of global change. In *Proc. 13th IEEE Mass Storage Systems Symposium*, pages 47-53, 1994.

[DM92] E. Drakopoulos and M.J. Merges. Performance analysis of client-server storage systems. *IEEE Transactions on Computers*, 41(11):1442-1452, 1992.

[DunhamNorth96] J. Dunham and B. North. EOSDIS statistics collection and reporting system, 1996. Available by anonymous FTP at eos.nasa.gov:/EosDis/Daacs/Statistics.

[ESDIS] ESDIS document catalog. http://spsosun.gsfc.nasa.gov/ESDIS_Docs.html.

[FY92] A. Finestead and N. Yeager. Performance of a distributed superscaler storage server. In *Proc. NASA Goddard Conference on Mass Storage Systems and Technology*, NASA CP- 3198, Vol. II, pages 573-580, 1992.

[GMW95] L. Golubchik, R.R. Muntz, and R.W. Watson. Analysis of striping techniques in robotic storage libraries. In *Proc. 14th IEEE Mass Storage Systems Symposium*, pages 225-238, 1995.

[HRT91] S.E. Hauser, C. Rivera, and G.R. Thoma. Factors affecting the performance of a dos-based WORM file server. In *Mass Storage Systems Symposium*, pages 33-37, 1991.

[He85] A.R. Hevner. Evaluation of optical disk systems for very large database applications. In *ACM SIGMETRICS* conference, pages 166-172, 1985.

[Ho92] K. Howard. High speed data duplication/data distribution - an adjunct to the mass storage equation. In *Proc. NASA Goddard Conference on Mass Storage Systems and Technology*, pages 123-133, 1992.

[HR93] G. Hull and S. Ranade. Performance measurements and operational characteristics of the Storage Tek ACS 4400 tape library with the Cray Y-MP EL. In *Proc. NASA Goddard Conference on Mass Storage Systems and Technology*, pages 111-122, 1993.

[Jo95a] T. Johnson. Analysis of the request patterns to the nssdc on-line archive. In Proc. 4th *NASA Goddard Conf. on Mass Storage Systems and Technologies*, 1995.

[Johnson96] T. Johnson. An Analytical Performance Model of Robotic Storage Libraries. In *Performance '96*, 1996.

[Ka92] K. Kant. *Introduction to Computer System Performance Evaluation*. McGraw Hill, 1992.

[KHE91] S.M. Kelly, R.A. Haynes, and M.J. Ernest. Benchmarking a network storage service. In *Mass Storage Systems Symposium*, pages 38-44, 1991.

[KGT90] K.F. Klenk, J.L. Green, and L.A. Treinish. A Cost Model for NASA Data Archiving. Technical Report 90-08, National Space Science Data Center, NASA Goddard Space Flight Center, 1990.

[KBCH94] B. Kobler, J. Berbert, P. Caulk, and P.C. Hariharan. Architecture and design of storage and data management for the NASA Earth Observing System Data and Information System (EOSDIS). In *Proc. 14th IEEE Mass Storage Systems Symposium*, pages 65-78, 1995.

[LK93] E.K. Lee and R.H. Katz. An analytic performance model of disk arrays. In *ACM SIGMETRICS*, pages 98-109, 1993.

[Lu95] L. Lueking. Managing and serving a multi-terabyte data set at the Fermilab D0 experiment. In *Proc. 14th IEEE Mass Storage Systems Symposium*, pages 200-208, 1995.

[PMHY95] O.I. Pentakalos, D.A. Menasce, M. Halem, and Y. Yesha. Analytical performance modeling of hierarchical mass storage systems. Technical Report TR-CS-96-01, Dept. of Computer Science, University of Maryland, 1995. A short version appears in the *14th IEEE Mass Storage Symposium* proceedings.

[Rh92] E. Rahm. Performance evaluation of extended storage architectures for transaction processing. In *ACM SIGMOD*, pages 308-317, 1992.

[RE89] K.K. Ramakrishnan and J.S. Emer. Performance analysis of mass storage service alternatives for distributed systems. *IEEE Trans. on Software Engineering*, 15(2):120-133, 1989.

[ReWi94] C. Ruemmler and J. Wilkes. An introduction to disk drive modeling. *IEEE Computer*, 27:17-28, 1994.

[Sa95] S. Sarawagi. Database Systems for Efficient Access to Tertiary Memory. In *Proc. 14th IEEE Mass Storage Systems Symposium*, pages 120-126, 1995.

[Sequoia2k] Sequoia 2000 home page. http://s2k-ftp.cs.berkeley.edu:8000/.

[Th95] A. Thomasian. Surveyor's forum: High performance secondary memory. *Computing Surveys*, 27(2):292-295, 1995.

[Th96] A. Thomasian. Approximate analyses for fork/join synchronization in RAID 5. *Computer Systems: Science and Engineering*, 1996. To appear.

[ThTa94] A. Thomasian and A.N. Tantawi. Approximate solutions for M/G/1 fork/join synchronization. In *Proc. 1994 Winter Simulation Conference*, 1994.

[Tijms94] H.C. Tijms. *Stochastic Models: An Algorithmic Approach*. Wiley, 1994.

[Wa74] S.J. Waters. Estimating disk seeks. *The Computer Journal*, 18(1):12-17, 1974.

[YHY94] T. Yang, S. Hu, and Q. Yang. A closed-form formula for queuing delays in disk arrays. In *Proc. Intl. Conf. on Parallel Processing*, pages II:189-192, 1994.

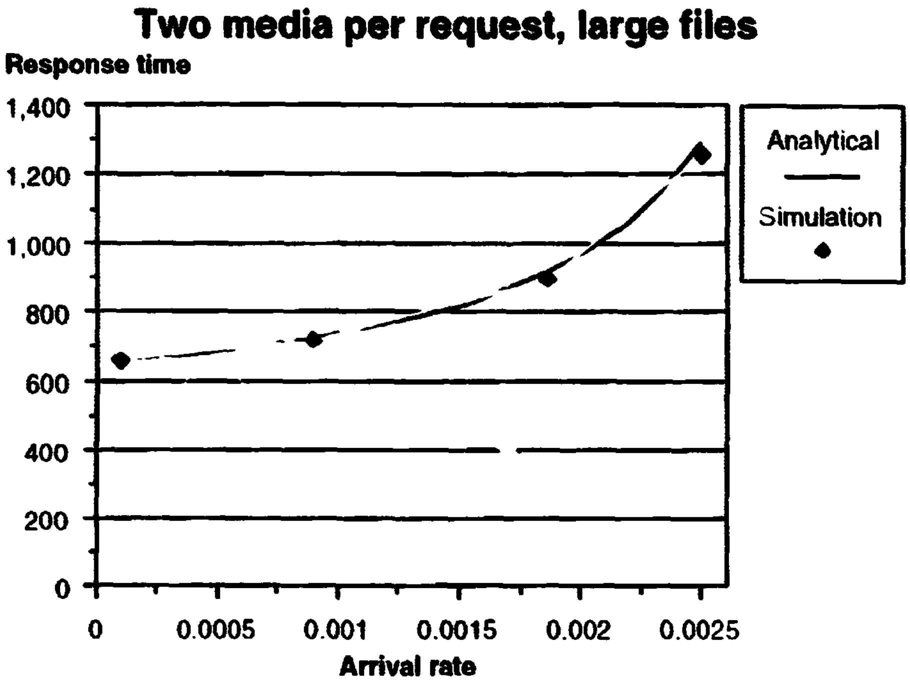


Figure 4

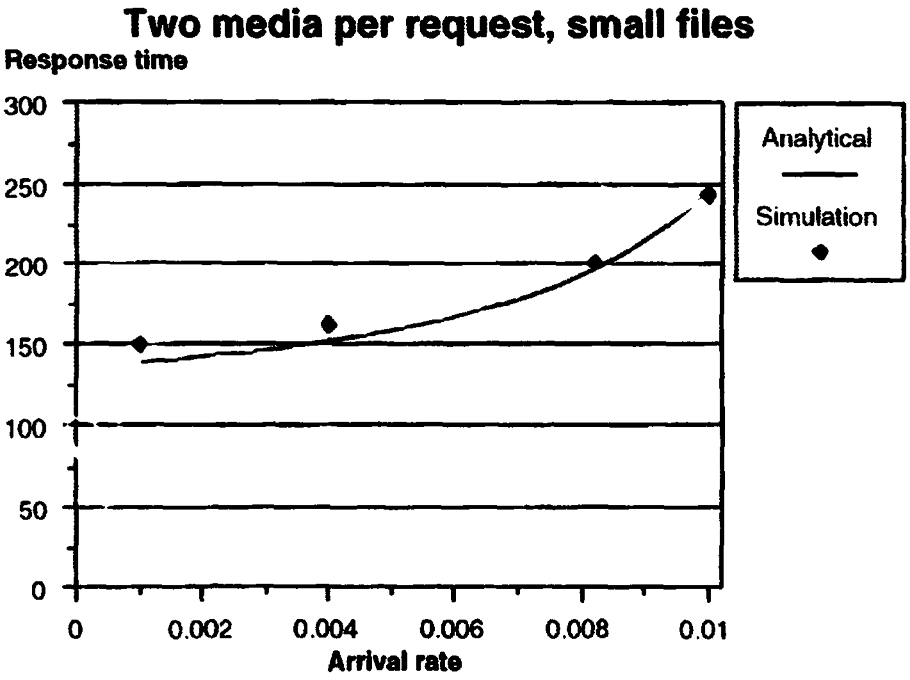


Figure 5

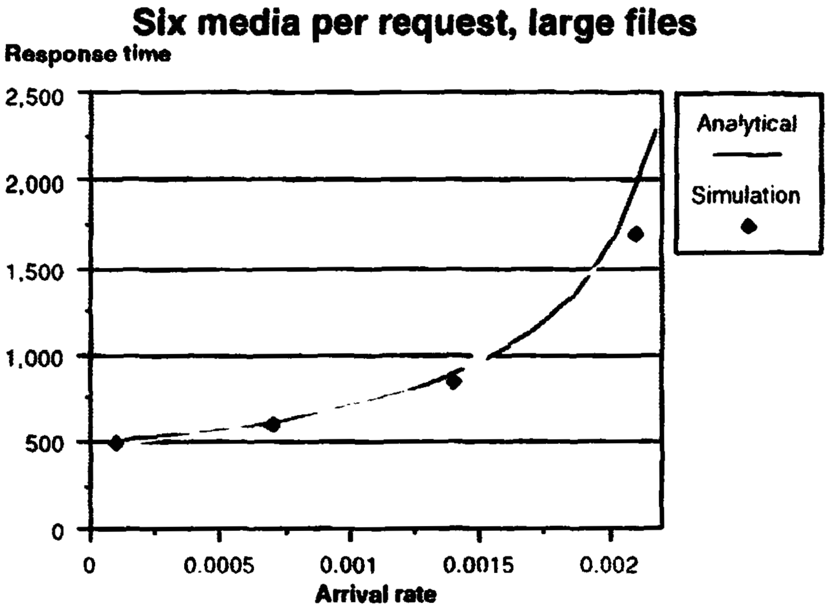


Figure 6

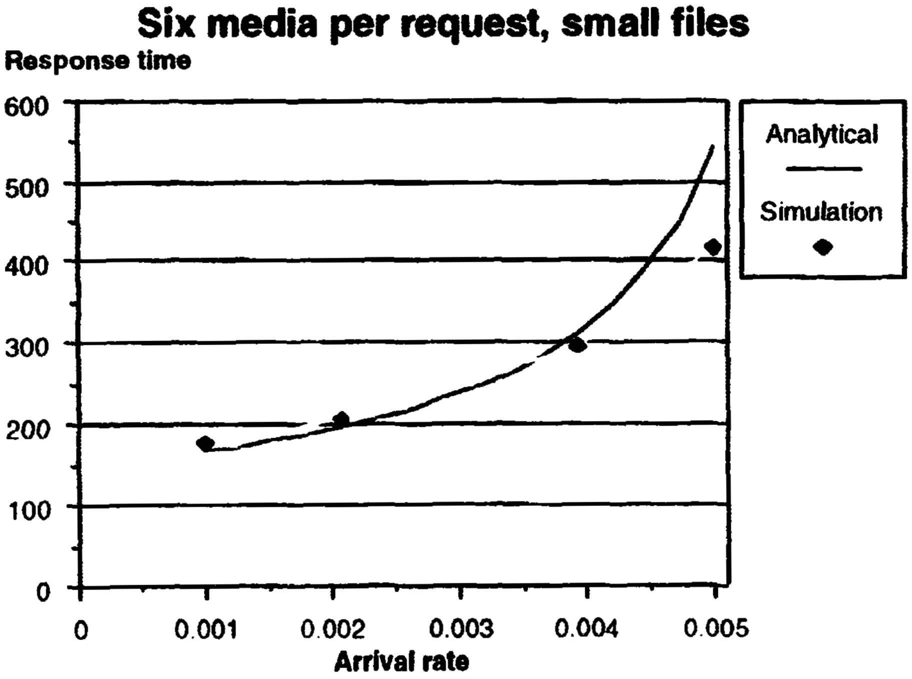


Figure 7

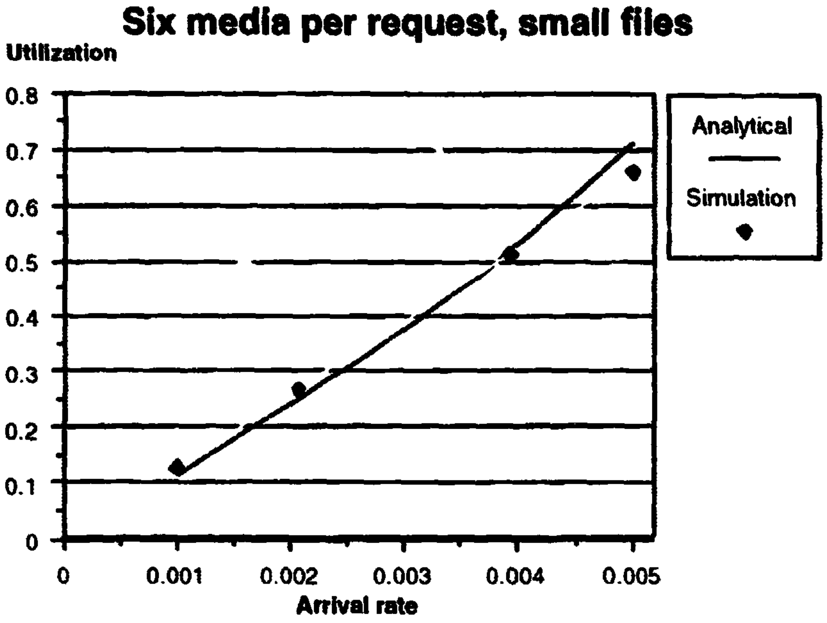


Figure 8

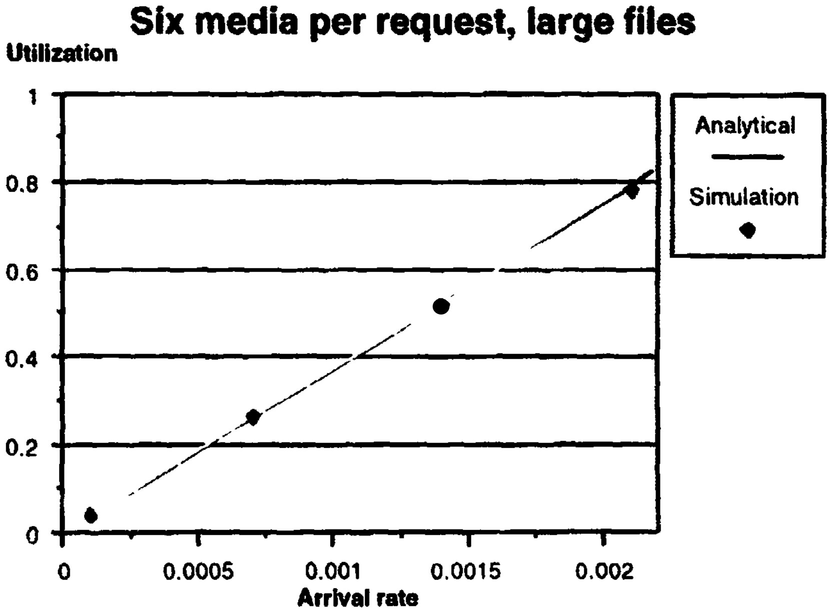


Figure 9

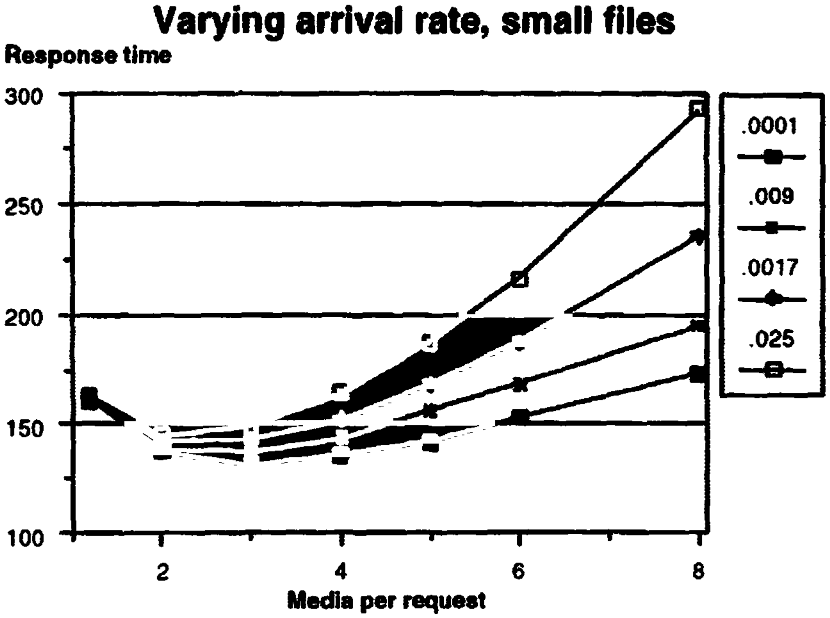


Figure 10

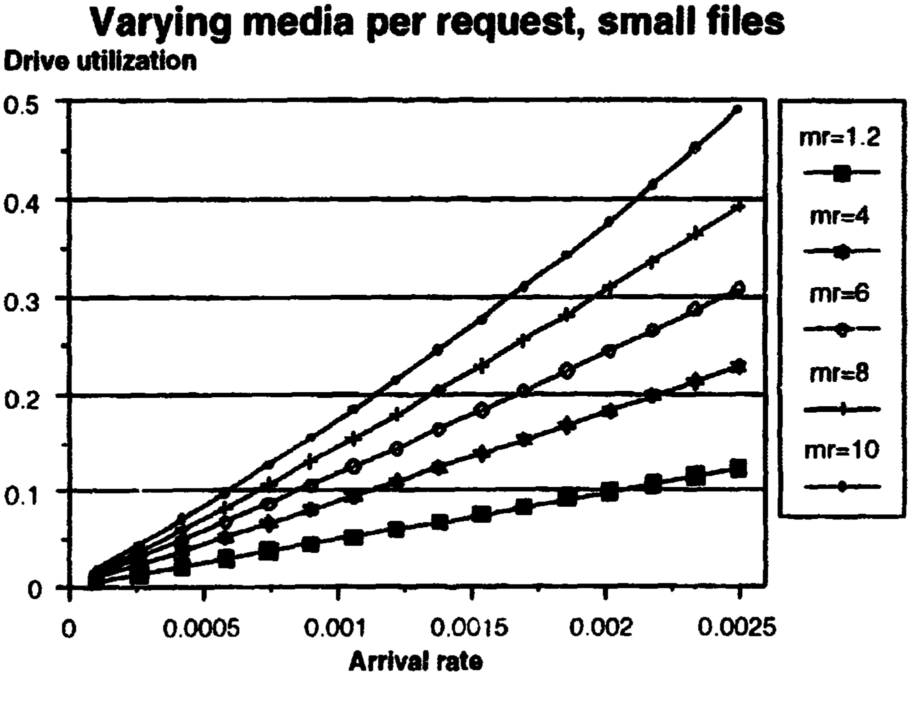


Figure 11

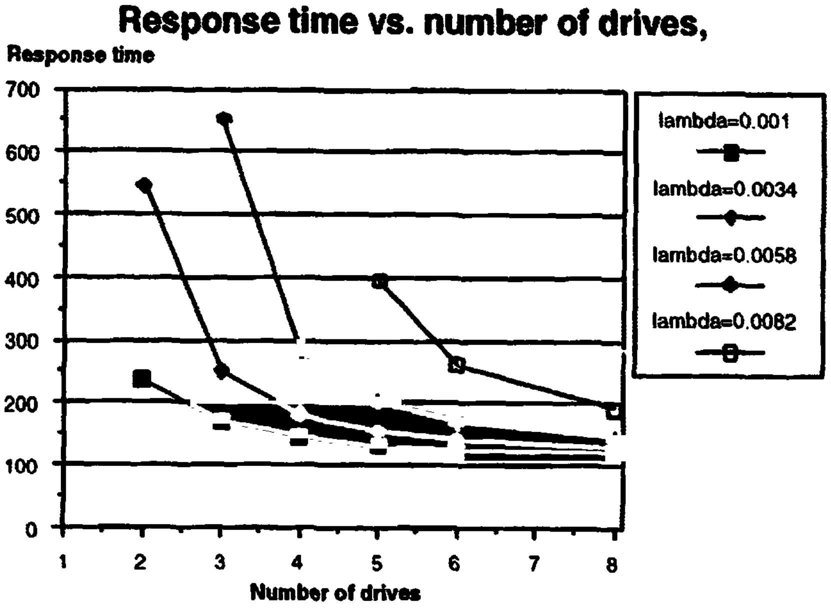


Figure 12

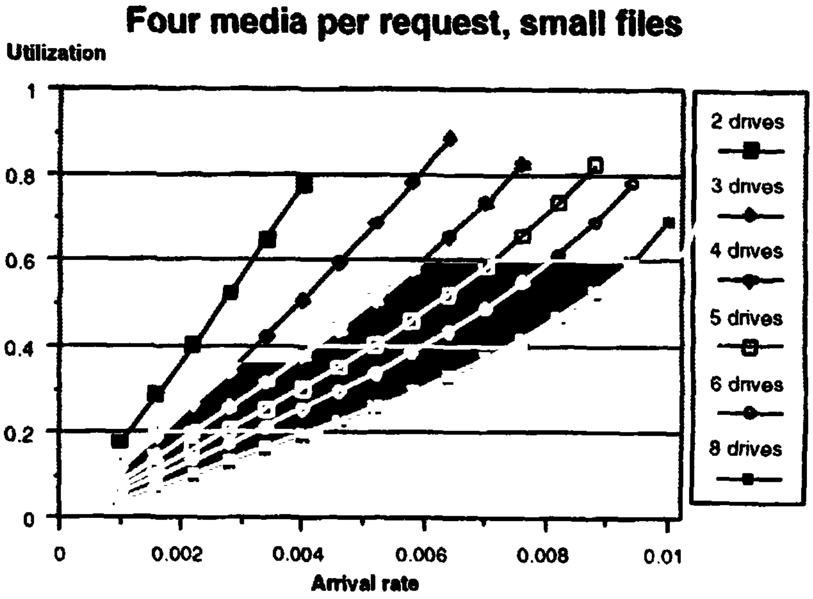


Figure 13

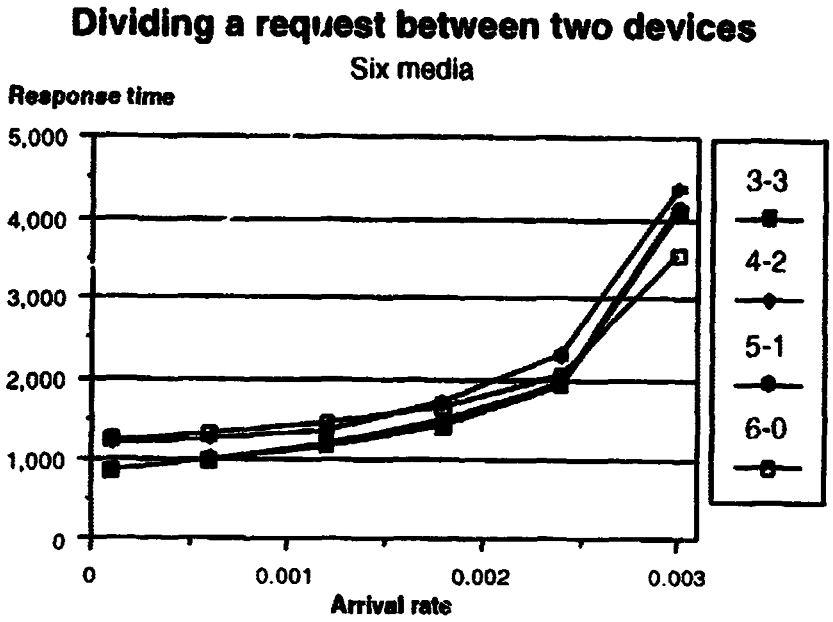


Figure 14

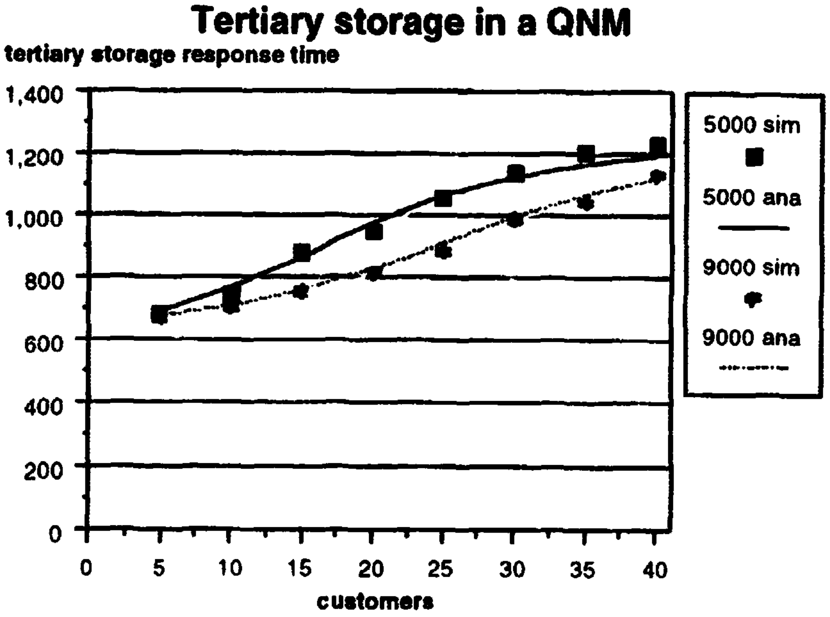


Figure 15

NEXT
DOCUMENT

I/O-Efficient Scientific Computation Using TPIE

Darren Erik Vengroff[*]
Dept. of Electrical Engineering
Evans Hall
Univ. of Delaware
Newark, DE 19716
voice: (302) 831-2405
FAX: (302) 831-4316
email: vengroff@ee.udel.edu

Jeffrey Scott Vitter[†]
Dep. of Computer Science
Box 90129
Duke University
Durham, NC 27708-0129
voice: (919) 660-6548
FAX: (919) 660-6502
email: jsv@cs.duke.edu

July 25, 1996

Abstract

In recent years, I/O-efficient algorithms for a wide variety of problems have appeared in the literature. Thus far, however, systems specifically designed to assist programmers in implementing such algorithms have remained scarce. TPIE is a system designed to fill this void. It supports I/O-efficient paradigms for problems from a variety of domains, including computational geometry, graph algorithms, and scientific computation. The TPIE interface frees programmers from having to deal not only of explicit read and write calls, but also the complex memory management that must be performed for I/O-efficient computation.

In this paper, we discuss applications of TPIE to problems in scientific computation. We discuss algorithmic issues underlying the design and implementation of the relevant components of TPIE and present performance results of programs written to solve a series of benchmark problems using our current TPIE prototype. Some of the benchmarks we present are based on the NAS parallel benchmarks [5], while others are of our own creation.

We demonstrate that the CPU overhead required to manage I/O is small and that even with just a single disk the I/O overhead of I/O-efficient computation ranges from negligible to the same order of magnitude as CPU time. We conjecture that if we use a number of disks in parallel this overhead can be all but eliminated.

[*] Supported in part by the U.S. Army Research Office under grant DAAH04-93-G-0076 and by the National Science Foundation under grant DMR-9217290. Portions of this work were conducted while visiting the University of Michigan and Duke University

[†] Supported in part by the National Science Foundation under grant CCR-9007851 and by the U.S. Army Research Office under grants DAAL03-91-G-0035 and DAAH04-93-G-0076.

1 Introduction

The Input/Output communication between fast internal memory and slower secondary storage is the bottleneck in many large-scale applications. The significance of this bottleneck is increasing as internal computation gets faster and parallel computing gains popularity [17]. CPU bandwidth is currently increasing at a rate of 40–60% per year, versus an annual increase in bandwidth of 7–10% for disk drives [18]. Main memory sizes are also increasing, but not fast enough to meet the needs of many large-scale applications. Additionally, main memory is roughly two orders of magnitude more expensive than disks. Thus, if I/O-efficient code can be written so as to provide performance near that obtained by solving the same problem on a machine with a much larger RAM, a great deal of money can be saved.

Up to this point, a great many I/O-efficient algorithms have been developed. The problems that have been considered include sorting and permutation-related problems [1, 2, 14, 15, 22], computational geometry [3, 4, 11, 23] and graph problems [7]. Until recently, there had been virtually no work directed at implementing these algorithms. Some work has now begun to appear [6, 19], but to the best of our knowledge no comprehensive package designed to support I/O-efficient programming across multiple platforms and problem domains has appeared. One goal of our ongoing research is to remedy this problem. Towards that end, we are developing TPIE, a transparent parallel I/O environment designed to facilitate the implementation of I/O-efficient programs.

In this work, we describe a series of experiments we have run using a prototype implementation of the TPIE interface. The experiments were chosen as models of common operations in scientific codes. Several of the experiments are on NAS parallel benchmarks designed to model large-scale scientific computation [5]. The results of our experiments demonstrate that I/O-efficient programs can be written using a high-level, portable, abstract interface, yet run efficiently.

In Section 2, we introduce the parallel I/O model of computation on which the algorithms that TPIE implements are based. In Section 3, we describe the TPIE system itself and the structure of our current prototype. In Section 4, we discuss the benchmarks we implemented, the algorithms that TPIE uses, and the performance of our implementations. Finally, we list a number of open problems worthy of examination in the continued pursuit of I/O-efficient computation.

2 The Parallel I/O Model of Computation

The algorithms TPIE uses are typically based on those designed for the parallel I/O model of computation [22]. This model abstractly represents a system having one or more processors, some fixed amount of main memory, and one or more independent disk drives. It is described by the following parameters:

$$\begin{aligned} N &= \text{\# of items in the problem instance} \\ M &= \text{\# of items that can fit into main memory} \end{aligned}$$

$$\begin{aligned} B &= \text{\# of items per disk block} \\ D &= \text{\# of disks.} \end{aligned}$$

We define an I/O operation, or simply an I/O for short, to the be process of transferring exactly one block of data to or from each disk. The I/O complexity of an algorithm is simply the number of I/Os it performs.

In discussing the I/O complexity of algorithms, we make every effort to avoid the use of big-Oh asymptotic notation. Instead, we are interested in as exact a figure as possible for the number of I/Os an algorithm will use for a given problem size and run-time environment. In some cases, the relative effect of rounding up certain quantities to the their integral ceilings can be significant, for example, when quantities round to small natural numbers. This effect is typically ignored when big-Oh notation is used. We are careful in this work to consider ceilings explicitly when their effect is significant; we use the ceiling notation $\lceil x \rceil$ to denote the smallest integer that is $\geq x$.

3 TPIE: A Transparent Parallel I/O Environment

TPIE [20, 21] is a system designed to assist programmers in the development of I/O-efficient programs for large-scale computation. TPIE is designed to be portable across a variety of platforms, including both sequential and parallel machines with both single and multiple I/O devices. Applications written using TPIE should continue to run unmodified [illegible] moved from one system that supports TPIE to another. In order to facilitate [illegible] level of portability, TPIE implements a moderately sized set of high-level access methods. The access methods were chosen based on their paradigmatic importance in the design of I/O-efficient algorithms. Using these access methods, we can implement I/O-efficient algorithms for many problems, including sorting [2, 15, [illegible], 22], permuting [8, 9, 10, 22], computational geometry problems [3, 4, 11, 23], graph-theoretic problems [7], and scientific problems [8, 13, 22].

Because such a large number of problems can be solved using a relatively [illegible] number of paradigms, it is important that the access method implementations remain flexible enough to allow application programs a great deal of control over the functional details of the computation taking place within the fixed set of paradigms. To accomplish this, TPIE takes a somewhat non-traditional approach to I/O. Instead of viewing computation as an enterprise in which code reads data, operates on it and then writes results, we view it as a continuous process in which program objects are fed streams of data from an outside source and leave trails of results behind them. The distinction is subtle, but significant. In the TPIE model, programmers don't have to worry about making explicit calls to I/O subroutines or managing internal memory data structures in a run-time dependent environment. Instead, they merely specify the functional details of the computation they wish to perform within a given paradigm. TPIE then choreographs an appropriate sequence of data movements to keep the computation fed.

TPIE is implemented in C++ as a set of templated classes and functions and a run-time library. Currently, a prototype implementation supports access to data stored on one or more disks attached to a workstation.[1] In the future, we plan to port the interface to larger multiprocessors and/or collections of workstations connected to a high-speed LAN. From the programmer's perspective, very little will change when the system moves to parallel hardware. All the same access methods will continue to exist, and applications will still be written with a single logical thread of control, though they will be executed in a data parallel manner.

The current TPIE prototype is a modular system with three components. The Access Method Interface (AMI) provides the high-level interface to the programmer. This is the only component with which most programmers will need to directly interact. The Block Transfer Engine (BTE) is responsible for moving blocks of data to and from the disk. It is also responsible for scheduling asynchronous read-ahead and write-behind when necessary to allow computation and I/O to overlap. Finally, the Memory Manager (MM) is responsible for managing main memory resources. All memory allocated by application programs or other components of TPIE is handled by the MM. In the case of application programs, this is facilitated through the use of a global `operator new()` in the TPIE library.

The AMI supports access methods including scanning, distribution, merging, sorting, permuting, and matrix arithmetic. In order to specify the functional details of a particular operation, the programmer defines a particular class of object called a scan management object. This object is then passed to the AMI, which coordinates I/O and calls member functions of the scan management object to perform computation. Readers interested in the syntactic details of this interaction are referred to the TPIE Manual, a draft version of which is currently available [21].

4 TPIE Performance Benchmarks

The benchmarks we implemented work with four of the basic paradigms TPIE supports: scanning, sorting, sparse matrices, and dense matrices. The benchmarks illustrate important characteristics not only of the TPIE prototype and the platform on which the tests were run, but also of I/O-efficient computation in general. In the exposition that follows we will discuss both.

Two of the benchmarks are based on the NAS parallel benchmarks set [5], which consists of kernels taken from applications in computational fluid dynamics. Besides being representative of scientific computations, these benchmarks also provide reference output values that can be checked to verify that they are implemented correctly. In addition to the NAS benchmarks, there are two new benchmarks designed to further exercise TPIE's matrix arithmetic routines.

Each of the benchmarks is accompanied by a graph illustrating the performance of one or more TPIE applications written to execute it. The graphs show both overall wall time and CPU time on the y-axis, as plotted against various problem sizes on the x axis. Given adequate and appropriately utilized I/O bandwidth, the wall time

[1] The following workstation/OS combinations are supported: Sun Sparcstation/SunOS 4.x, Sun Sparcstation/Solaris 5.x, DEC Alpha/OSF/1 1.x and 2.x, HP 9000/HP-UX.

and CPU time curves would be identical; therefore, getting them as close together as possible is an important performance goal.[2]

All of the benchmarks were run on a Sun Sparc 20 with a single local 2GB SCSI disk. The operating system was Solaris 5.3. Aside from the test application being run, the system was idle. In all cases, TPIE was configured to restrict itself to using 4 megabytes of main memory. I/O was performed in blocks of 64KB, with read-ahead and write-behind handled by a combination of TPIE and the operating system. The reason we used such a large block size was so that our computations would be structurally similar, in terms of recursion and read/write scheduling, to the same applications running on a machine with $D = 8$ disks and a more natural block size of 8KB. On such a system, I/O performance would increase by a factor of close to 8, whereas internal computation would be essentially unaffected.

4.1 Scanning

The most basic access method available in TPIE is scanning, which is implemented by the polymorphic TPIE entry point **AMI_scan()**. Scanning is the process of sequentially reading and/or writing a small number of streams of data. Essentially any operation that can be performed using $O(N/DB)$ I/Os can be implemented as a scan. This includes such operations as data generation, prefix sums, element-wise arithmetic, inner products, Graham's scan for 2-D convex hulls (once the points are appropriately sorted), selection, type conversion, stream comparison, and many others. The functional details of any particular scan are specified by a scan management object.

4.1.1 Scanning Benchmark

Because scanning is such a generic operation, we could have chosen any of a very wide variety of problems as a benchmark. We chose the NAS benchmark NAS EP [5] for two reasons: it was designed to model computations that actually occur in large-scale scientific computation; and it can be used to illustrate an important class of scan optimizations called scan combinations.

The NAS EP benchmark generates a sequence of independent pairs of Gaussian deviates. It first generates a sequence of $2N$ independent uniformly distributed deviates using the liner congruential method [12]. Then, it uses the polar method [12] to generate approximately $(\pi/4)N$ pairs of Gaussian deviates from the original sequence of uniform deviates.

Performance of our TPIE implementation of NAS EP is shown in Figure 1. There are three sets of curves, labeled "TPIE, 2 Scans," "TPIE, Optimized," and "Single Variable."

[2] One obvious way to bring these curves together is to increase the CPU time by performing additional or less efficient computation. Clearly, this is not the mechanism of choice. Instead we seek to reduce the overall time by reducing the amount of I/O and/or improving increasing the overlap between computation and asynchronous I/O.

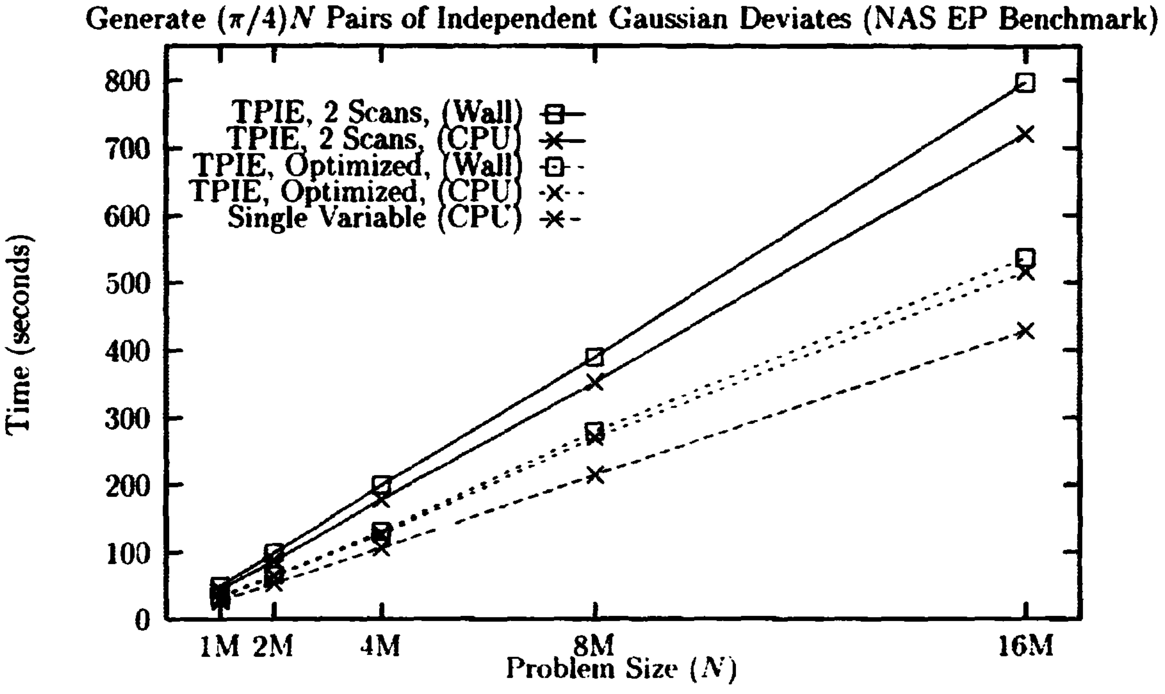


Figure 1: NAS EP Benchmark

The distinction between the 2 scan TPIE curves and the optimized TPIE curves is that in the former, two separate scans are performed, one to write the uniformly distributed random variates and the other to read the uniformly distributed random variates and write the Gaussian pairs, whereas in the latter, the two steps are combined into a single scan. As expected, the optimized code outperforms the unoptimized code.

This difference is significant not so much because it tells programmers they should combine scans, as because of the fact that scan combination is a relatively straightforward optimization that can be automated by a preprocessor. Such a preprocessor would parse the C++ text of a program and, where possible, construct hybrid scan management objects. The scans would then be replaced by a single scan using the hybrid object. Additionally, scans can often be piggy-backed on many other types of operations, such as merges, distributions, sorts, and permutations.

Returning to Figure 1, the single variable curve plots the CPU performance of a C++ program that does not perform any I/O at all, using TPIE or any other system. Instead, each pair of random variates is simply written over the previously generated pair in main memory. The purpose of this curve is to illustrate a fundamental lower bound on the CPU complexity of generating the variates. By comparing this to the CPU curves of the TPIE implementations we can see that the CPU overhead associated with scheduling and performing I/O, communicating between the components of TPIE, and interacting with the user supplied scan management object is quite small. In the optimized case it amounts to approximately 20%.

4.2 Sorting

Sorting is a fundamental subroutine in many computations. There are a tremendous number of sorting algorithms which support many different models of computation and assumptions about input format and/or key distribution. In this section we discuss a number of issues related to sorting in external-memory, both theoretical and practical.

4.2.1 I/O-Efficient Sorting Algorithms

With rare exception[3], I/O-efficient comparison sorts fall into one of two categories, merge sorts and distribution sorts. Merge sorts work from the bottom up, sorting small subfiles and then combining them into successively larger sorted files until all the data is in one large sorted file. Distribution sorts work from the top down by computing medians in the data and then distributing the data into buckets based on the median values. The buckets are then recursively sorted and appended to one another to produce the final output. The I/O structure of radix sort resembles that of distribution sort, except that the entire set of keys is involved in $O((\lg N/M)/(\lg M/DB))$ large $O(M/DB)$-way distribution steps.

One common technique for dealing with multiple disks in parallel is to stripe data across them so that the heads of the D disks are moved in lock step with one another, thereby simulating a single large disk with block size DB. On a striped disk system, the I/O complexity of merge sorting N objects is

$$2\frac{N}{DB}\left\lceil \log_{(M/2DB)} N/M \right\rceil = 2\frac{N}{DB}\left\lceil \frac{\lg(N/M)}{\lg(M/2DB)} \right\rceil. \tag{1}$$

Each item is read once and written once in each pass, and all reads and writes are fully blocked. The logarithmic factor is the number of levels of recursion required to reduce merge subproblems of size M into the final solutic. of size N. Each stream is double buffered, hence we can merge $M/'2DB)$ streams at a time. If we are able to compute medians perfectly with no additional cost, as in the case where the keys are uniformly distributed, we can perform distribution sort in this same bound.

Asymptotically, the I/O bound (1) is not optimal for sorting. By using the D disks independently, we can do distribution sort in

$$2k\frac{N}{DB}\left\lceil \frac{\lg(N/M)}{\lg(M/2B)} \right\rceil \tag{2}$$

I/Os, where $k \geq 1$ is a constant whose value depends on the complexity of finding the medians, the quality of the medians as partitioning elements, and how evenly the buckets are distributed over the D disks. Although the the denominator in (2) is larger than the denominator in (1) by an additive term of $\lg D$, the leading constant factor in (2) is larger than that of (1) by a multiplicative factor of k. A number of

[3] For a recent example, see [3].

independent disk distribution sort algorithms exist [1, 15, 16, 22], with values of k ranging from approximately 3 to 20.

Before implementing an external sort on parallel disks, it is useful to examine the circumstances under which the I/O complexity (2) for using the disks independently is less than the I/O complexity (1) with striping. If we neglect the ceiling term for the moment, algebraic manipulation tells us that it is better to use disks independently when

$$\left(\frac{M}{2B}\right)^{\left(\frac{k-1}{k}\right)} < D.$$

Thus, D must be at least some root of $M/2B$. The critical issue now becomes the value of k. If $k = 1$ (i.e., if we do not need extra I/Os to compute $M/2B$ medians that partition the data evenly and if each resulting bucket is output evenly among the D disks), it is better to use disks independently. However, if $k = 4$, we need $D > (M/2B)^{3/4}$ in order for using disks independently to be worthwhile, which is not the case in current systems. For this reason, TPIE implements both merging and distribution in a striped manner. Work is continuing on developing practical methods that use disks independently.

Another important aspect of the behavior of I/O-efficient algorithms for sorting concerns the behavior of the logarithmic factor $\lceil \lg(N/M)/\lg(M/2DB) \rceil$ in the denominator of (1). The logarithmic term represents the number of merge passes in the merge sort, which is always integral, thus necessitating the ceiling notation. The ceiling term increases from one integer to the next when N/M is an exact power of $M/(2DB)$. Thus over very wide ranges of values of N, of the form $M^i/(2DB)^i < N/M \leq M^{i+1}/(2DB)^{i+1}$, for some integer $i \geq 1$, the I/O complexity of sorting remains linear in N. Furthermore, the possibility of $i \geq 3$ requires an extremely large value of N if the system in question has anything but the tiniest of main memories. As a result, although the I/O complexity of sorting is not, strictly speaking, linear in N, in practice it often appears to be.

4.2.2 Sorting Benchmark

The NAS IS benchmark is designed to model key ranking [5]. We are given an array of integer keys $K_0, K_1, \ldots K_{N-1}$ chosen from a key universe $[0, U)$, where $U \ll N$. Our goal is to produce, for each i, the rank $R(K_i)$, which is the position K_i would appear in if the keys were sorted. The benchmark does not technically require that the keys be sorted at any time, only that their ranks be computed. As an additional caveat, each key is the average of four random variates chosen independently from a uniform probability distribution over $[0, U)$. The distribution is thus approximately normal. Ten iterations of ranking are to be performed, and at the beginning of each iteration an extra key is added in each distant tail of the distribution.

In order to rank the keys, we sort them, scan the sorted list to assign ranks, and then re-sort based on the original indices of the keys. In the first sort, we do not have a uniform distribution of keys, but we do have a distribution whose probabilistic structure is known. Given any probabilistic distribution of keys with cumulative distribution function (c.d.f.) F_K, we can replace each key value k_i by $k_i' = F_K(k_i)$ in order to get keys that appear as if chosen at random from a uniform distribution on $[0, 1]$. Because the keys of the NAS IS benchmark are sums of four independent uniformly distributed random variates, their c.d.f. is a relatively easy to compute piecewise fourth degree polynomial.

For the sake of comparison, we implemented this first sort in four ways, using both merge sort and three variations of distribution sorting. One distribution sort, called CDF1, assumed that the keys were uniformly distributed. The next CDF4, used the fourth degree c.d.f. mentioned above to make the keys more uniform. Finally, as a compromise, CDF2 used a quadratic approximation to the 4th degree c.d.f. based on the c.d.f. of the sum of two independent uniform random variables.

In the second sort, the indices are the integers in the range $[0, N)$, so we used a distribution sort in all cases. The rationale behind this was that distribution and merging should use the same amount of I/O in this case, but distribution should require less CPU time because it has no need for the main-memory priority queue that merge sorting requires.

The performance of the the various approaches is shown in Figure 2. As we expected, merge sort used more CPU time than any of the distribution sorts and the more complicated the c.d.f. we computed the more CPU time we used. When total time is considered, merge sort came out ahead of the distribution sorts. This appears to be the result of imperfect balance when the keys are distributed, which causes an extra level of recursion for a portion of the data. Interestingly, the quality of our c.d.f. approximation had little effect on the time spent doing I/O. We conjecture that this would not be the case with more skewed distributions, such as exponential distributions. We plan experiments to confirm this. The jump in the total time for the merge sort that occurs between 8M and 10M is due to a step being taken in the logarithmic term in that range.

4.3 Sparse Matrix Methods

Sparse matrix methods are widely used in scientific computations. A fundamental operation on sparse matrices is that of multiplying a sparse $N \times N$ matrix A by an N-vector x to produce an N-vector $z = Ax$.

4.3.1 Sparse Matrix Algorithms

Before we can work with sparse matrices in secondary memory, we need a way of representing them. In the algorithms we consider, a sparse matrix A is represented by a set of nonzero elements E. Each $e \in E$ is a triple whose components are row(e), the row index of e in A, col(e), the column index of e in A, and value(e), the value of A[row(e), col(e)].

In main memory, sparse matrix-vector multiplication can be implemented using Algorithm 1. If the number of nonzero elements of A is N_z, then Algorithm 1 runs in $O(N_z)$ time on a sequential machine.

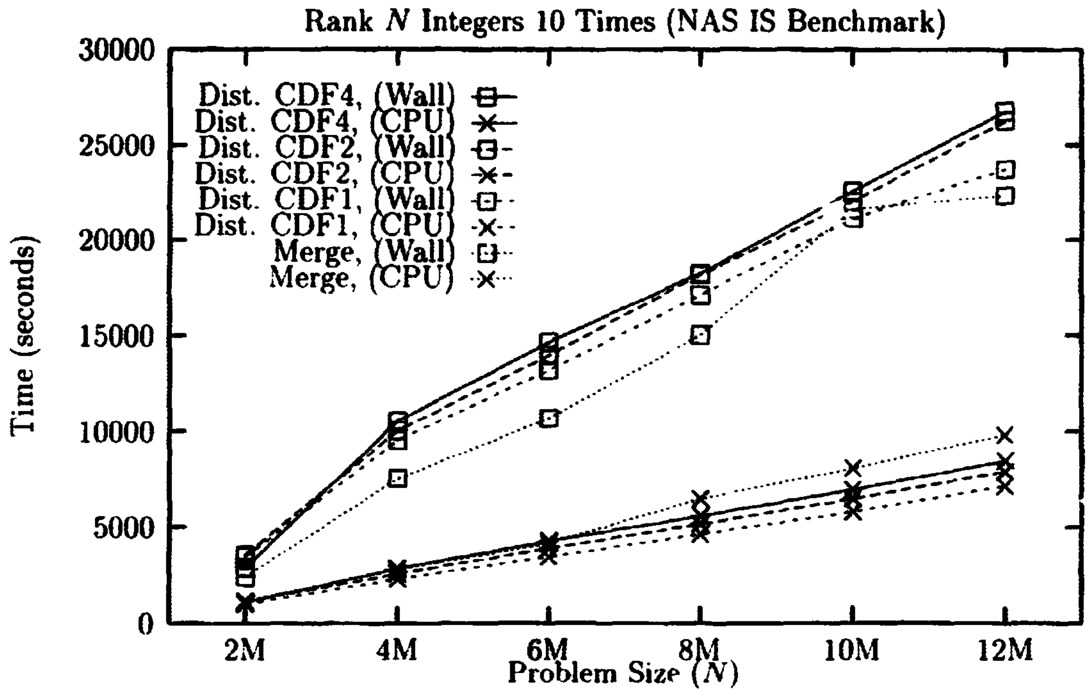


Figure 2: NAS IS benchmark performance

In secondary memory, we can also use Algorithm 1, but I/O performance depends critically on both the order in which the elements of A are processed and which of components of z and x are in main memory at any given time. In the worst case, every time we reference an object it could be swapped out. This would result in $3N_z$ I/Os.

In order to guarantee I/O-efficient computation, we reorder the elements of A in a preprocessing phase. In this preprocessing phase, A is divided into N/M separate $M \times N$ sub-matrices A_i, called bands. Band A_i contains all elements of A from rows iM to $(i+1)M - 1$ inclusive. Although the dimensions of all the A_i are the same, the number of nonzero elements they contain may vary widely. To complete the preprocessing, the elements of each of the A_i are sorted by column.

Once A is preprocessed into bands, we can compute the output sub-vector

$$z[iM \ldots (i+1)M - 1]$$

from A_i and x using a single scan, as shown in Algorithm 2. If we ignore the preprocessing phase for a moment and assume that the elements of x appear in order in external memory, the I/O complexity of Algorithm 2 is $N_z/DB + \lceil N/M \rceil N/DB + N/DB$. The entire preprocessing phase can be implemented as a single sort on the nonzero elements of A, with [illegible] and index being the primary key and column being a secondary key. This takes $2N_z/DB \left\lceil \frac{\lg(N_z/M)}{\lg(M/2DB)} \right\rceil$ I/Os, as explained in Section 4.2.1. Note, however, that the preprocessing only has to be done a single time for a given matrix A. After that, the main phase of the algorithm can be executed repeatedly for many different vectors x. This is a common occurrence in iterative methods.

```
$z \leftarrow 0$;
foreach nonzero element $e$ of $A$ do
    $z[\mathrm{row}(e)] = z[\mathrm{row}(e)] + \mathrm{value}(e) \times x[\mathrm{col}(e)]$;
endforeach
```

Algorithm 1: An algorithm for computing $z = Ax$ where A is a sparse $N \times N$ matrix and x and z are N-vectors.

4.3.2 The SMOOTH Benchmark

TPIE supports sparse matrices at a high level as a subclass of AMI streams. The nonzero elements of a sparse matrix are stored in the stream as (row, column, value) triples as described in the preceding section. AMI entry points for constructing sparse matrices as well as multiplying them by vectors are provided.

In order to test the performance of TPIE sparse matrices, we implemented a benchmark we call SMOOTH, which models a finite element computation on a 3-D mesh.

The SMOOTH benchmark implements sparse matrix-vector multiplication between a $N \times N$ matrix with $27N$ nonzero elements and a dense N-vector. The result is then multiplied by the matrix again. Ten iterations are performed.

The performance of SMOOTH is shown in Figure 3. Although we do ten iterations of multiplication, and only preprocess once, the total time with preprocessing is significantly higher that that of the multiplication iterations alone. As expected, I/O is not a major contributor to this difference, because sorting only requires a small number of linear passes through the data. The big difference is in CPU time. The additional CPU time used in preprocessing the sparse matrix is roughly twice the CPU time used in all ten iterations of the multiplication.

4.4 Dense Matrix Methods

Dense matrices appear in a variety of computations. Like sparse matrices, they are often multiplied by vectors, and banding techniques similar to those discussed in the previous section can by used. Another fundamental operation is multiplication of two $K \times K$ matrices A and B to produce $C = AB$.

4.4.1 Dense Matrix Algorithms

Asymptotically I/O-optimal multiplication of two $K \times K$ matrices over a quasiring can be done in $\Theta(K^3/\sqrt{M}DB)$ I/Os [22]. There are at least two simple algorithms that achieve this bound. The first algorithm, Algorithm 3, uses a recursive divide-and-conquer approach. The second algorithm, Algorithm 4, also partitions the input matrices, but all partitioning is done up front in a single permutation of each matrix.

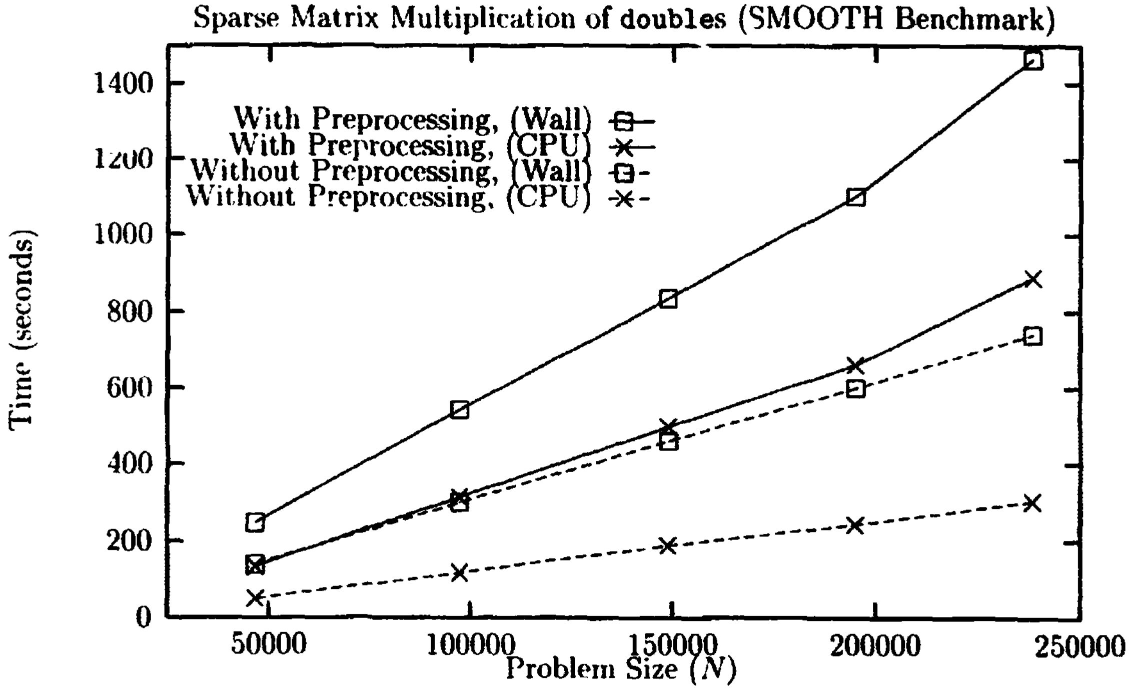
```
// Preprocessing phase:

foreach nonzero element e of $A$ do
    Put $e$ into $A_{\lfloor \mathrm{row}(e)/M \rfloor}$;
endforeach
for $i \leftarrow 0, N/M$ do
    Sort the elements of $A_i$ by column;
endfor

// Main algorithm:

Allocate a main memory buffer $z_M$ of $M$ words;
for $i \leftarrow 0$ to $\lceil N/M \rceil$ do
    $z_M \leftarrow 0$;
    foreach nonzero element e of $A_i$ do
        $z_M[\mathrm{row}(e) - iM] = z_M[\mathrm{row}(e) - iM] + \mathrm{value}(e) \times x[\mathrm{col}(e)]$;
    endforeach
    Write $z_M$ to $z[iM \ldots (i+1)M - 1]$;
endfor
```

Algorithm 2: An I/O-efficient algorithm for computing $z = Ax$ where A is a sparse $N \times N$ matrix and x and z are N-vectors.



Figurc 3: SMOOTH Benchmark

The matrix product is then produced iteratively, and a single final permutation returns it to canonical order. Both algorithms assume the input matrices are stored in row major order.

The I/O complexity of Algorithm 3 is

$$\frac{12\sqrt{3}K^3}{\sqrt{M}DB},$$

while that of Algorithm 4 is

$$\frac{2\sqrt{3}K^3}{\sqrt{M}DB} + \mathrm{prep}(K),$$

where prep(N) is the I/O complexity of the preprocessing and postprocessing steps, which can be done by sorting the K^2 elements of the three matrices, giving us

$$\mathrm{prep}(K) = 6\frac{K^2}{DB}\left\lceil 2\frac{\lg(K/M)}{\lg(M/2DB)}\right\rceil.$$

In special circumstances, when B, D, K, and $\sqrt{M}$ are all integral powers of two, the pre- and post-processing are bit-matrix multiply complement permutations, which can be performed in fewer I/Os than sorting [10].

```
(1)  if $3K^2 \leq M$ then
(2)       read $A$ and $B$ into main memory;
(3)       compute $C = AB$ in main memory;
(4)       write $C$ back to disk;
(5)  else
(6)       partition $A$ at row $K/2$ and column $K/2$;
(7)       label the four quadrant sub-matrices $A_{1,1}$, $A_{1,2}$, $A_{2,1}$, and $A_{2,2}$ as shown in Figure 4;
(8)       partition $B$ into $B_{1,1}$, $B_{1,2}$, $B_{2,1}$, and $B_{2,2}$ in a similar manner;
(9)       permute all sub-matrices of $A$ and $B$ into row major order;
(10)      Perform the (11)-(14) using recursive invocations of this algorithm
(11)           $C_{1,1} \leftarrow A_{1,1}B_{1,1} + A_{1,2}B_{2,1}$;
(12)           $C_{1,2} \leftarrow A_{1,1}B_{1,2} + A_{1,2}B_{2,2}$;
(13)           $C_{2,1} \leftarrow A_{2,1}B_{1,1} + A_{2,2}B_{2,1}$;
(14)           $C_{2,2} \leftarrow A_{2,1}B_{1,2} + A_{2,2}B_{2,2}$;
(15)      Reconstruct $C$ from its sub-matrices $C_{1,1}$, $C_{1,2}$, $C_{2,1}$, and $C_{2,2}$;
(16)      permute $C$ back into row major order;
(17) endif
```

Algorithm 3: A recursive divide-and-conquer approach to matrix multiplication. Two $K \times K$ input matrices A and B are multiplied to produce $C = AB$.

4.4.2 Dense Matrix Benchmark

TPIE has high-level support for dense matrices over arbitrary user-defined quasirings. Operations supported include initialization, element-wise arithmetic, and matrix-matrix multiplication. Matrix-matrix multiplication uses Algorithm 4. Separate AMI entry points are available for the preprocessing permutation and the iterative multiplication itself, allowing a matrix to be preprocessed once and then multiplied by a number of other matrices.

We implemented a benchmark, called DENSE, which constructs two $K \times K$ matrices, preprocesses them, and then multiplies them. Times were recorded for both the total benchmark and for the multiplication only. The results are shown in Figure 6. As expected, the CPU time required to multiply the matrices follows a cubic path. Because of read-ahead, I/O is almost fully overlapped with computation, making the CPU and total time curves virtually indistinguishable. The cost of preprocessing the matrices is approximately one third of the cost of multiplying them. Thus if several multiplications are done with the same matrix amortization greatly reduces this cost.

5 Conclusions

We have presented a series of results demonstrating that I/O-efficient computation can be made practical for a variety of scientific computing problems. This computation is made practical by TPIE, which provides a high level interface to computational

$A_{1,1}$	$A_{2,1}$
$A_{2,1}$	$A_{2,2}$

Figure 4: Partitioning a matrix into quadrants

```
(1)  partition $A$ into $K/\sqrt{M/3}$ rows and $K/\sqrt{M/3}$ columns of
         sub-matrices each having $\sqrt{M/3}$ rows and $\sqrt{M/3}$ columns;
     // step (1) is shown in Figure 5
(2)  partition $B$ in a manner similar to $A$;
(3)  permute all $A_{i,j}$ and $B_{i,j}$ into row major order;
(4)  foreach $i,j$ do
(5)      $C_{i,j} \leftarrow \sum_k A_{i,k}B_{k,j}$;
(6)  endforeach
(7)  reconstruct $C$ from all $C_{i,j}$;
(8)  permute $C$ back into row major order;
```

Algorithm 4: An iterative approach to matrix multiplication.

paradigms whose I/O complexity has been carefully analyzed. Using TPIE results in only a small CPU overhead versus entirely in core implementation, but allows much larger data sets to be used. Additionally, for the benchmarks we implemented, I/O overhead ranged from being negligible to being of the same order of magnitude as internal computation time.

If we replace the single disk on which the tests were run with D disks, where D is on the order of 8, we conjecture that the I/O time required in our computations could be reduced by a factor very close to D. In applications like DENSE, where I/O overhead is already negligible, little would change, but in applications like NAS IS and NAS EP, we would see a dramatic reduction in I/O overhead. As discussed in Section 4, CPU time should not change appreciably. Recalling that a portion of the I/O that would be reduced by a factor of D is already overlapping with computation, we expect that in many case the I/O overhead (the portion that does not overlap with CPU usage) could be eliminated. We plan to assemble a parallel disk system to evaluate this conjecture experimentally.

In addition to the problems discussed here, there are many other scientific computations that we believe can benefit from careful analysis and I/O-efficient implementation. These include LUP decomposition, FFT computation, and multi-grid

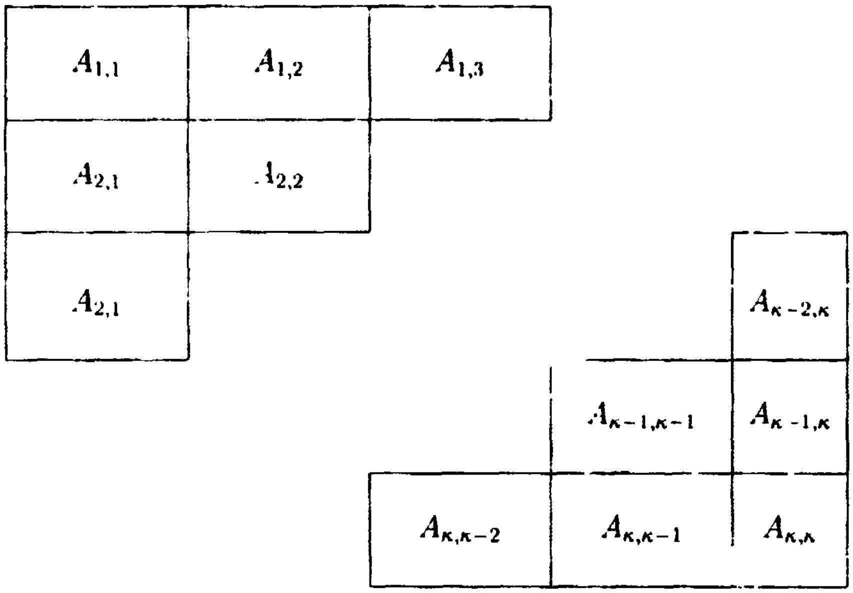


Figure 5: Partitioning a matrix into sub-matrices in step (1) of Algorithm 4. Each sub-matrix $A_{i,j}$ has $\sqrt{M/3}$ rows and $\sqrt{M/3}$ columns. The number of sub-matrices across and down A is $\kappa = K/\sqrt{M/3}$.

methods, all of which we plan to explore as the TPIE project continues. We also plan to investigate the construction of a scan combining preprocessor as described in Section 4.1.1.

Complementing this high level work, there are a number of potentially interesting I/O related research topics concerning how environments like TPIE should interact with operating systems. These include models of application controlled virtual memory and the behavior of TPIE applications in multiprogrammed environments where the main memory available to a given process may vary over time.

In closing, we are encouraged by the results we have presented, which demonstrate that I/O efficient computation using an abstract, high level model is practical. It is important to realize, however, that this research is only in its infancy, and that many more questions, both theoretical and practical, remain to be answered.

6 Acknowledgments

We thank Yi-Jen Chiang, Liddy Shriver, and Len Wisniewski for helpful discussions on implementing I/O-efficient algorithms. We also thank Jim Baker, Philip Klein, and Jon Savage for suggesting that we study sparse matrix methods.

We also thank Yale Patt and the staff of the ACAL lab at the University of Michigan for the facilities and support that allowed much of this research to take place.

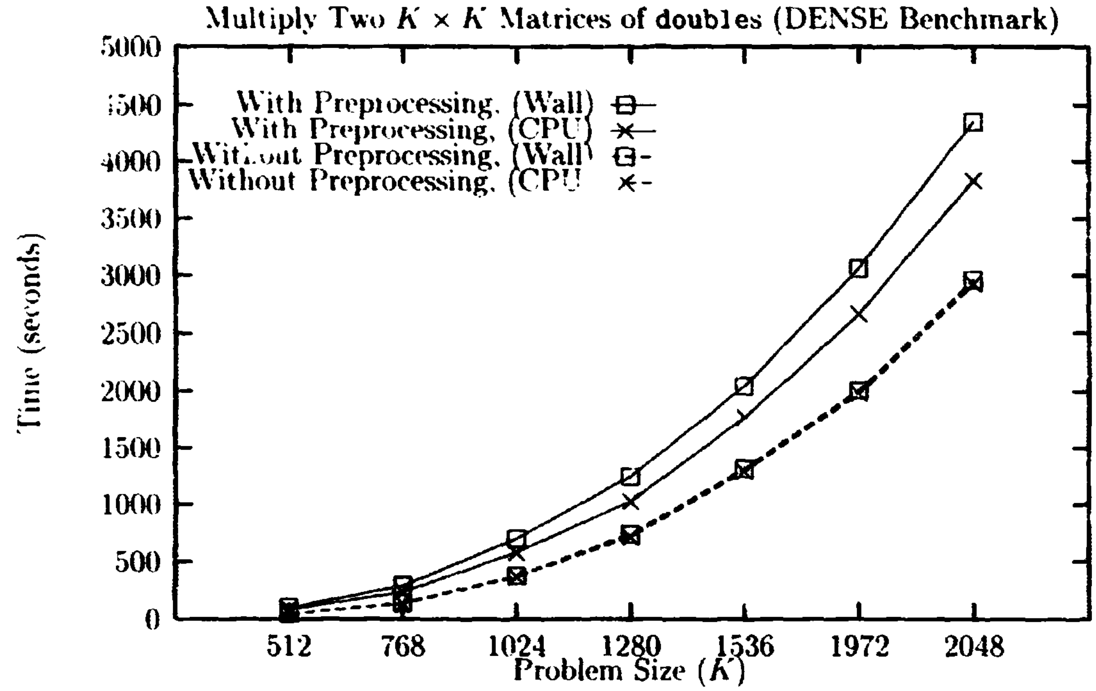


Figure 6: DENSE Benchmark

References

[1] A. Aggarwal and G. Plaxton. Optimal parallel sorting in multi-level storage. In *Proc. 4th Annual ACM-SIAM Symp. on Discrete Algorithms*, Arlington, VA, 1994.

[2] A Aggarwal and J. S. Vitter. The input/output complexity of sorting and related problems. *Communications of the ACM*, 31(9):1116 1127, 1988.

[3] L. Arge. The buffer tree: A new technique for optimal I/O-algorithms. Technical Report RS-94-16, BRICS, Univ. of Aarhus, Denmark, 1994.

[4] L Arge, D. E. Vengroff, and J. S. Vitter. External-memory algorithms for processing line segments in georgraphic information systems. submitted, 1995.

[5] D. Bailey, E. Barszcz, J. Barton, D. Browning, R. Carter, L. Dagum, R. Fatoohi, S. Fineberg, P. Fredrickson, T. Lansinki, R. Schrieber, H. Simon, V. Venkatakrishnan, and S. Weeratunga. The NAS parallel benchmarks. Technical Report RNR-94-007, RNR, March 1994.

[6] Y.-J. Chiang. Experiments on the practical I/O efficiency of geometric algorithms: Distribution sweep vs. plane sweep. In *Proc. 1995 Workshop on Algs. and Data Structures. (WADS)*. 1995

[7] Y.-J. Chiang, M. T. Goodrich, E. F. Grove, R. Tamassia, D. E. Vengroff, and J. S. Vitter. External-memory graph algorithms. In *Proc. ACM-SIAM Symp. on Discrete Alg.*, pages 139–149, 1995.

[8] T. H. Cormen. *Virtual Memory for Data Parallel Computing*. PhD thesis, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, 1992.

[9] T. H. Cormen. Fast permuting in disk arrays. *Journal of Parallel and Distributed Computing*, 17(1–2):41–57, Jan./Feb. 1993.

[10] T. H. Cormen, T. Sundquist, and L. F. Wisniewski. Asymptotically tight bounds for performing BMMC permutations on parallel disk systems. Technical Report PCS-TR94-223, Dartmouth College Dept. of Computer Science, July 1994.

[11] M. T. Goodrich, J.-J. Tsay, D. E. Vengroff, and J. S. Vitter. External-memory computational geometry. In *IEEE Foundations of Comp. Sci.*, pages 714–723, 1993.

[12] D. E. Knuth. *The Art of Computer Programming: Seminumerical Algorithms*, volume 2. Addison Wesley, 2d. ed. edition, 1981.

[13] C. E. Leiserson, S. Rao, and S. Toledo. Efficient out-of-core algorithms for linear relaxation using blocking covers. In *IEEE Foundations of Comp. Sci.*, pages 704–713, 1993.

[14] M. H. Nodine and J. S. Vitter. Large-scale sorting in parallel memories. In *Proc. 3rd ACM Symp. on Parallel Algorithms and Architectures*, pages 29–39, 1990.

[15] M. H. Nodine and J. S. Vitter. Deterministic distribution sort in shared and distributed memory multiprocessors. In *Proc. 5th ACM Symp. on Parallel Algorithms and Architectures*, June 1993.

[16] M. H. Nodine and J. S. Vitter. Paradigms for optimal sorting with multiple disks. In *Proc. of the 26th Hawaii Int. Conf. on Systems Sciences*, Jan. 1993.

[17] Y. N. Patt. The I/O subsystem — a candidate for improvement. *IEEE Computer*, 27(3):15–16, Mar. 1994.

[18] C. Ruemmler and J. Wilkes. An introduction to disk drive modeling. *IEEE Computer*, 27(3):17–28, Mar. 1994.

[19] E. A. M. Shriver and L. F. Wisniewski. Choreographed disk access using the whiptail file system: Applications. manuscript.

[20] D. E. Vengroff. A transparent parallel I/O environment. In *Proc. 1994 DAGS Symposium on Parallel Computation*, July 1994.

[21] D. E. Vengroff. *TPIE User Manual and Reference*. Duke University, 1995. Available via WWW at `http://www.cs.duke.edu:~dev/tpie.html`.

[22] J. S. Vitter and E. A. M. Shriver. Algorithms for parallel memory I: Two-level memories. *Algorithmica*, 12(2), 1994.

[23] B. Zhu. Further computational geometry in secondary memory. In *Proc. Int. Symp. on Algorithms and Computation*, 1994.

NEXT
DOCUMENT

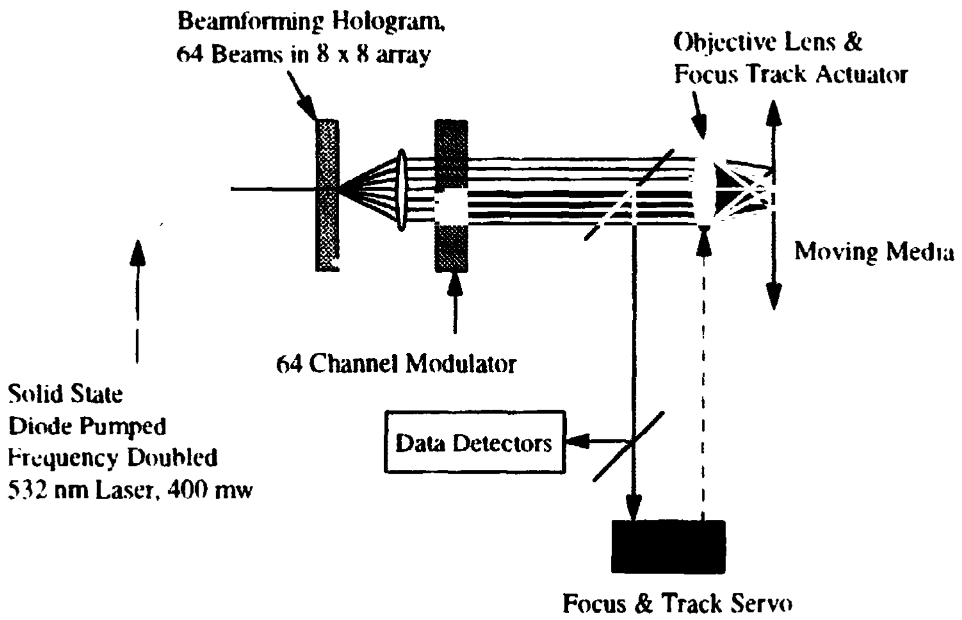
Progress Toward Demonstrating
A High Performance Optical Tape
Recording Technology

W. S. Oakley
LOTS Technology, Inc.
1274 Geneva Drive
Sunnyvale CA 94089
408-747 -1111
Fax: 408-747 - 0245

In September 1995 LOTS Technology received an award under the Advanced Technology Program to pursue high performance digital optical tape recording technology using a green laser source. The program is a two year technology development effort with the goal of demonstrating useful read/write data transfer rates to at least 100 megabytes per second and a user data capacity of up to one terabyte per cartridge implemented in a system using a '3480' style mono-reel tape cartridge. Although both write once and erasable phase change optical media have been previously demonstrated, and both are compatible with this technology, current availability limits this effort to the use of write once media. This paper discusses the technology developments achieved during the first year of the program during the period September 1995 through August 1996.

LOTS
TECHNOLOGY

Figure 1a: Basic System Approach

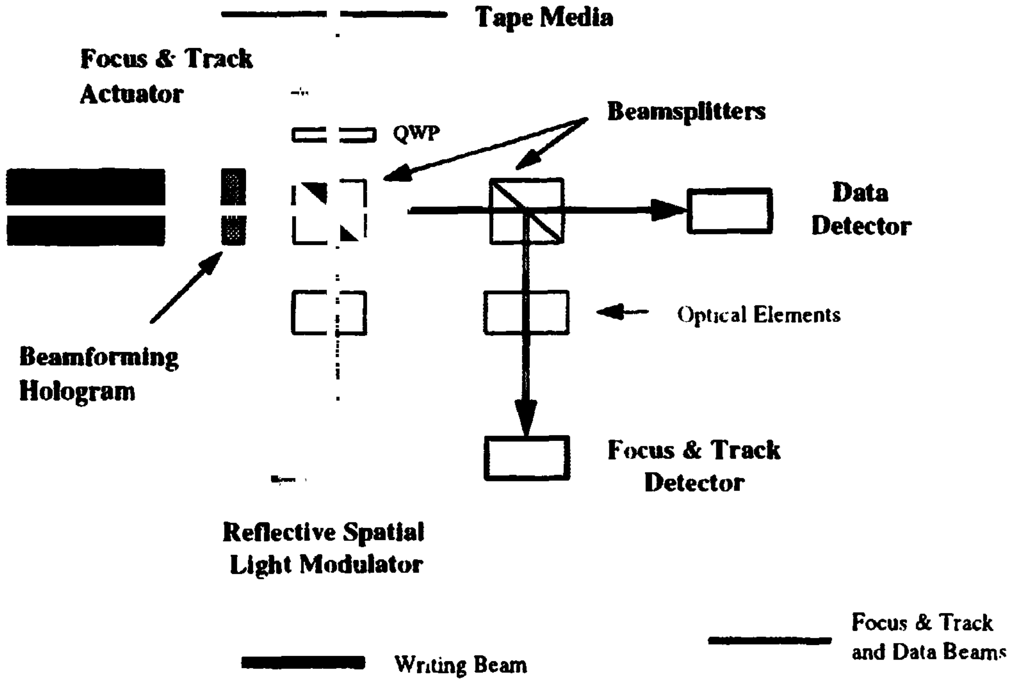


96Nas1a ppt

The primary intent of the program is to develop the technology for multi-beam digital optical recording and playback at high data transfer rates, 100 MB/sec. and above, and consistent with a minimum of a terabyte capacity per data cartridge. The basic design is implemented by a linear tape transport moving tape at several meters per second while the tape media is written to longitudinally by means of an array of focused and modulated laser beams. All writing beams are derived from a single diffraction limited green laser operating at 532 nanometers. The design is implemented using a hologram as a passive Beam-forming element to split the output from a single laser source into an array of 64 similar optical beams, each of which is independently modulated prior to focusing on the media with a nominally half micron spot size. Beam modulation is implemented at rates to 20 MHz. by means of an array modulator of 64 elements, one element for each beam. The basic recorder design concept is shown in **Figure 1a** and the optical implementation in **Figure 1b**.

LOTS TECHNOLOGY

Figure 1b: Optical Implementation



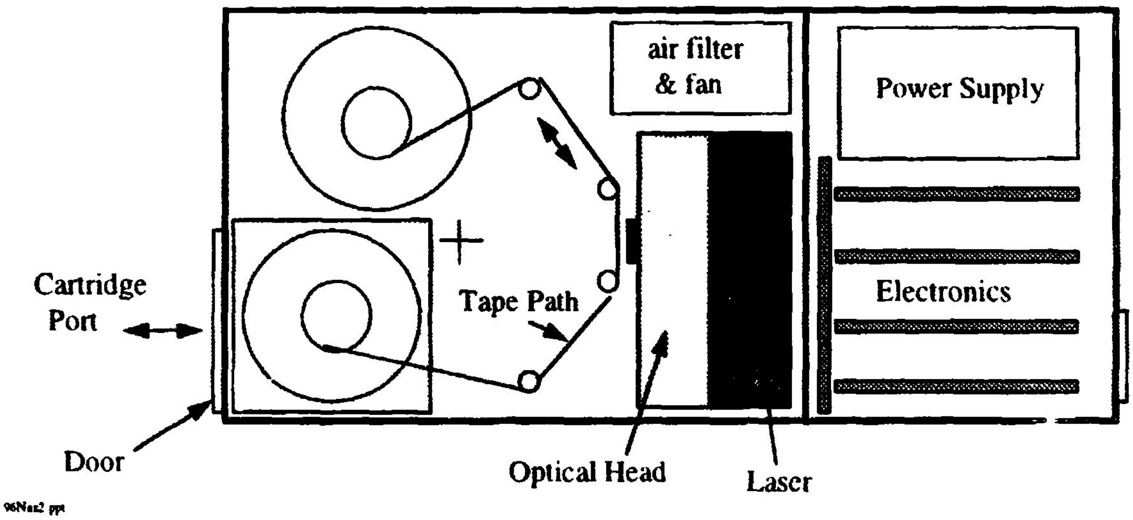
96Nas1b ppt

A conceptual physical layout of a future product is shown in **Figure 2**, emphasizing the inherent benefit of no head/media contact for optical recording and the preference for a clean tape transport environment to minimize media contamination by dust and dirt. The mechanical media transport system is configured to eliminate contact between the media recording surface and any transport component. The only recording layer contact is with the rear surface of the tape when it is wound either onto the take-up reel or into the cartridge.

LOTS TECHNOLOGY

Figure 2: Conceptual Physical Layout

- * No Contact with Media Recording Surface
- * Media in Filtered Air Environment



The overall system performance is directly determined by the specific implementations and the individual performance characteristics of the key components of the intended design. The design approach required to demonstrate a 100 MB/sec. transfer rate consists of the following major components;

(1) The tape transport.
A linear tape transport is desired to move half inch wide optical tape in a controlled manner at speeds up to about ten meters per second. The baseline design requires a tape speed of eight m/sec. for a 100 MB/Sec. user data rate.

(2) The array beamformer.
An array beamformer is required to produce an array of 64 similar optical read/write beams from a single diffraction limited input beam at a wavelength of 532 nm.

(3) The array modulator.
An array modulator is required to modulate each beam in the write array at a rate consistent with the desired bit writing rate, i.e. modulator rise and fall times of under 10 nanoseconds for the 100 MB/sec. system.

(4) Write/read channel data encoding and system.
Each write/read beam in the array and it's associated detector forms a data channel for writing/reading to/from the media. Increased system performance is achieved by encoding the data to improve the linear bit density in a track, thereby increasing system capacity and minimizing both tape speed and laser power requirements.

(5) The 532 nm laser.
A green laser operating at 532 nm is required with sufficient power to write to the media at the desired data rates, allowing for system optical transmission efficiency. The laser must be diffraction limited with a low noise amplitude to preserve data integrity. At the 100 MB/sec. data rate a source laser power of about 400 mW is required for the optical media currently in use.

(6) A multi-element detector.
A detector array matched to the format of the optical footprint on the tape media is required for data retrieval.

(7) Focus and track capability
A means must be provided of both maintaining optical focus on the moving media and following a previously written track group to sub micron accuracy for data retrieval.

The key characteristics of these components are interrelated with the basic system performance being developed as follows. The degree of data encoding employed directly affects the linear bit recording density and in conjunction with a given track spacing and tape width thereby determines the tape length for a given capacity. The bit density also directly affects the tape speed required to achieve a given data rate for a specified number of simultaneously written (or read) bit tracks. For a track spacing of 0.88 microns and a track group consisting of 64 individual bit tracks, each track group occupies section of tape 56.32 microns wide. With a guard band of two bits between track groups a group occupies 66 track widths or 58.08 microns. Therefore, 200 track groups can be written across a half inch (12.7 mm) wide tape and occupy 11.62 mm., leaving unwritten bands of 0.54 mm on each of the upper and lower tape edges. For a system of 1 Terabyte user capacity per cartridge with a data overhead of 30% of the raw capacity, a total capacity of 1,000/0.7 or 1,428 gigabytes is required. For 200 x 64 (= 12,800) data bit tracks this corresponds to a requirement of 111.56 megabytes or 892.5 megabits per bit track. This is 5.0 user (7.14 raw) gigabytes per 64 bit track group per tape length. For a system recording data with a linear density of 1 bit per micron, each bit track would therefore be 1.00 x 111.6 meters = 892.5 meters in length. It follows that a linear density of 2 bits per micron requires a tape length of 446.25 meters, and a tape length of 400 meters requires a bit track density of 2.23 bits/micron, etc..

The maximum length of tape that can be wound onto the 50 mm diameter hub in the industry standard 3480 cartridge is a function of the tape thickness and the maximum allowed outer tape pack diameter. A maximum outer diameter of 100 mm is assumed for the 3480 cartridge tape pack, (the reel flange diameter is 101 mm), which for 13 micron thick optical tape gives a maximum tape length of 453 meters which results in a minimum linear recording density requirement of 1.97 bits per micron for a one terabyte capacity. A more conservative tape length of 400 m requires a linear bit density of 2.23 bits/micron. To provide a one terabyte capacity in the '3480' style cartridge the data linear density per track must therefore be at least 1.97 and preferably greater than 2.23 bits per micron if a maximum of 400 meters of tape is used. The tape pack diameter for various lengths of 13 micron thick tape are given in **Table 1**.

Table 1. Outer Tape Pack Diameter vs. Tape length for 13 Micron Thick Tape.

Tape Pack Diameter in mm.	85	90	95	100
# Tape Wraps in Pack	1,346	1,538	1,730	1,923
Avg. Length/wrap - mm	212	219.9	227.8	235.6
Total Length - m	285.4	338.2	394.1	453.0

Note: The maximum flange diameter for a '3480' cartridge reel is 101 mm.

With a bit density of 1.97 bits/micron the tape velocity corresponding to a 100 MB/s. transfer rate in a design with 64 parallel data channels is 9.06 m/sec.. For the same channel parallelism and data rate, and with a bit density of 2.23 bits per micron the required tape speed is 8.0 m/sec.. Higher bit densities, i.e. of 3 bits/micron (or more), are preferred and would allow tape speeds below 6 m/sec. however such bit track densities are unlikely to be achieved with a 0.532 micron wavelength laser source and PPM (Pulse Position Modulation) encoding. Greater data storage densities can be achieved by the use of PWM (Pulse Width Modulation) encoding but are not necessary to achieve the program performance goals and would entail considerably greater effort and technological risk.

With the user capacity at 70% of the raw capacity a 100 megabyte/sec. user data rate requires a raw rate of (100/0.7 =) 142.86 megabytes/sec., giving a raw bit rate of 1,142.9 megabits/sec. over 64 data channels, or 17.86 megabits/sec. per channel. At a linear density of 1 bit per micron this requires a system tape speed of 17.86 meters/sec.. Higher linear bit densities require less tape to provide a given capacity and consequently require lower tape speeds for any given data rate. Greater read/write channel parallelism, i.e. more bit tracks per track group, also permit a lower tape speed for a given aggregate data rate, but has no effect on cartridge capacity. Higher track densities also reduce the time to end of tape (EOT) for a given total capacity. The tape lengths and speeds required for various linear track densities (# bits per micron) for a one terabyte capacity system operating at a user data rate of 100 megabytes/sec. are given in **Table 2**.

Table 2. Tape Length per Terabyte & Tape Speed vs. Channel Parallelism, at Various Linear Bit Track Densities.

Linear Density	Tape Length / TB	Tape Speed (meters/sec.) vs. Number Bit Tracks per Track Group @ 100 MB/sec.		
(bits/micron)	meters			
		64	96	128
1.0	892.5	17.86	11.91	8.93
1.5	595.0	11.91	7.94	5.95
2.0	446.3	8.93	5.95	4.46
2.5	357.0	7.14	4.76	3.57
3.0	297.5	5.95	3.97	2.98
3.5	255.0	5.10	3.40	2.55
4.0	223.1	4.46	2.98	2.23

With a system capacity of one terabyte configured into 200 parallel track groups each of 5 gigabytes the time to the physical end of the tape is obviously the same for a given data rate regardless of the linear bit track density. i.e. The time to read/write each track group of 5 gigabytes, at a 100 MB/s.(= 0.1 GB/s.) data rate is (5 / 0.1) = 50 seconds. The time to write/read the entire tape is 200 times greater at 10,000 seconds or 2.778 hours.

For a diffraction limited green laser system operating at a wavelength of 532 nm the recording spot size is determined by the F/number (or Numerical Aperture, N.A.), of the objective lens which focuses each spot onto the optical tape media. A N.A. of 0.6 corresponds to an F/number of 0.666 which for a plane wave incident on the lens would create an Airy disc of radius 1.22 x 0.532 x 0.666 = 0.433 microns. For a slightly truncated Gaussian input beam as used in this system the full width at the half maximum (FWHM). power point of the focused writing beam is slightly less than this at about 0.39 microns. This is the nominal width of each written bit track on the media. The bit track separation of 0.88 microns is therefore more than twice the track width, providing greater than 20dB isolation between adjacent tracks on data readback.

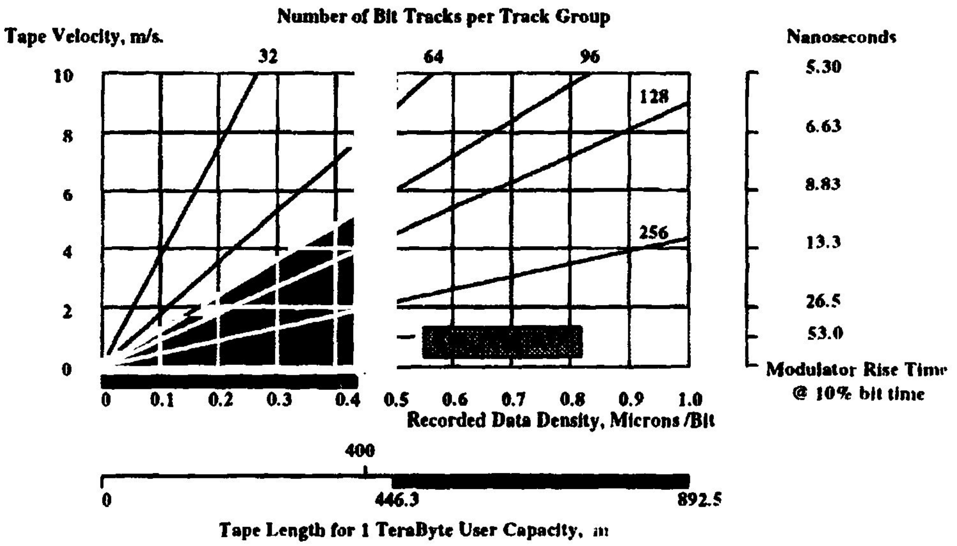
The required modulator response times to write a bit of appropriate mark length on the tape can be determined as a function of tape speed using the somewhat arbitrary criteria of write pulse rise/fall times equal to 10% of a bit mark at that speed. For a system using a green laser operating at 0.532 micron wavelength a minimum sized bit mark can be considered as nominally one wavelength long; i.e. 0.5 microns. Hence, for example; a rise time of one microsecond occurring during transit of 10% of a bit mark of 0.5 microns gives a tape speed of 0.05 microns per microsecond or 0.05 m/sec. Modulator rise times of 100, 10, 1 nanoseconds similarly correspond to tape speeds of 0.5, 5.0, and 50 meters/sec. respectively. As shown above, linear recorded bit densities between 2 and 3 bits per micron require tape speeds of between about 9 to 6 m/s., and therefore correspond to modulator rise times from approximately 5.9 to 8.8 nanoseconds. Tape speeds and array modulator rise times over these ranges have been demonstrated by LOTS during the first year of the ATP program, thereby validating these parameters in regard to the chosen design approach.

The data in Table 1 and Table 2 and the associated modulator response times are shown graphically in **Figure 3**. As the per cartridge tape length is limited to about 450 meters, recording densities below two bits per micron (0.5 microns/bit) do not provide a terabyte capacity. Pulse position encoding does not provide recording densities above about 2.3 bits/micron (0.435 microns/bit). Therefore to provide a terabyte capacity and reduce program risk by the use of standard PPM encoding, the system must operate in the range between 0.435 and 0.5 microns/bit. Selection of the standard PPM (2, 7) code provides a bit density of 2.22 bits/micron (0.45 microns/bit), requiring a tape speed of 8 m/sec for 64 parallel channels, or 4 m/sec. for 128 parallel channels, etc. to obtain the 100 MB/sec. data rate. A design utilizing fewer channels requires a proportionately greater tape speed and consequently a faster modulator response to enable recording. Two dimensional modulator arrays on about 150 micron centers have been fabricated and tested during the first year of the program.

LOTS TECHNOLOGY

Figure 3: Tape Velocity vs. Encoded Data Density for 100MB/s. Data Rate

Bit Track Spacing = 0.88 Microns
Overhead = 30% of Total Data

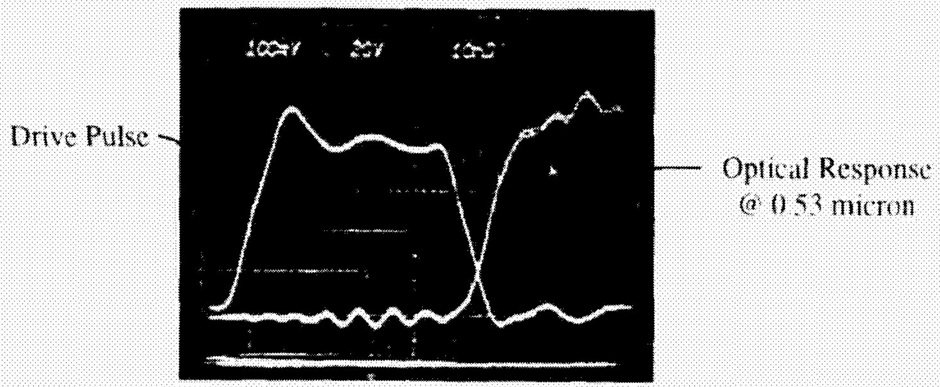


[illegible] NAS3.ppt

An oscilloscope trace of the response of a typical modulator element in the array to a drive pulse is shown in **Figure 4**. The specific modulator configuration tested required an 80 volt drive signal to achieve almost 100% throughput. Tests on various arrays have shown all modulating elements to perform in a nominally identical manner and the inter element crosstalk to be minimal. The writing beam quality is not significantly affected by the modulator, a diffraction limited output being maintained. The use of array modulators is therefore considered validated for the high data rate multi-beam recording application. Work is continuing to optimize the modulator geometry to reduce the drive voltage to a significantly lower level.

LOTS TECHNOLOGY

Figure 4: Modulator Optical Response



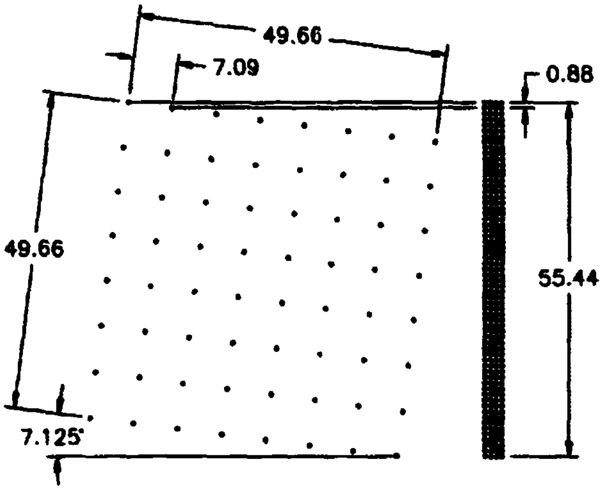
Transmissive Mode Response to 100 v Drive
Time Scale = 10 nanoseconds/div.

WNASrpt

The 64 data bit array to be recorded on the media is derived from a two dimensional array of 8x8 beams where other ancillary beams external to the main array are used for focus and tracking. The 8x8 beam pattern output from the Beam-forming hologram forms the closely spaced track group array of 64 tracks by virtue of being rotated a few degrees to the direction of tape motion as shown in **Figure 5**.

Figure 5: 100 - T Beamforming Pattern

64 Beams in Tilted 8 X 8 Array



GREEN - 532 nm

8 X 8 SPOTS ON .88 CENTERLINES

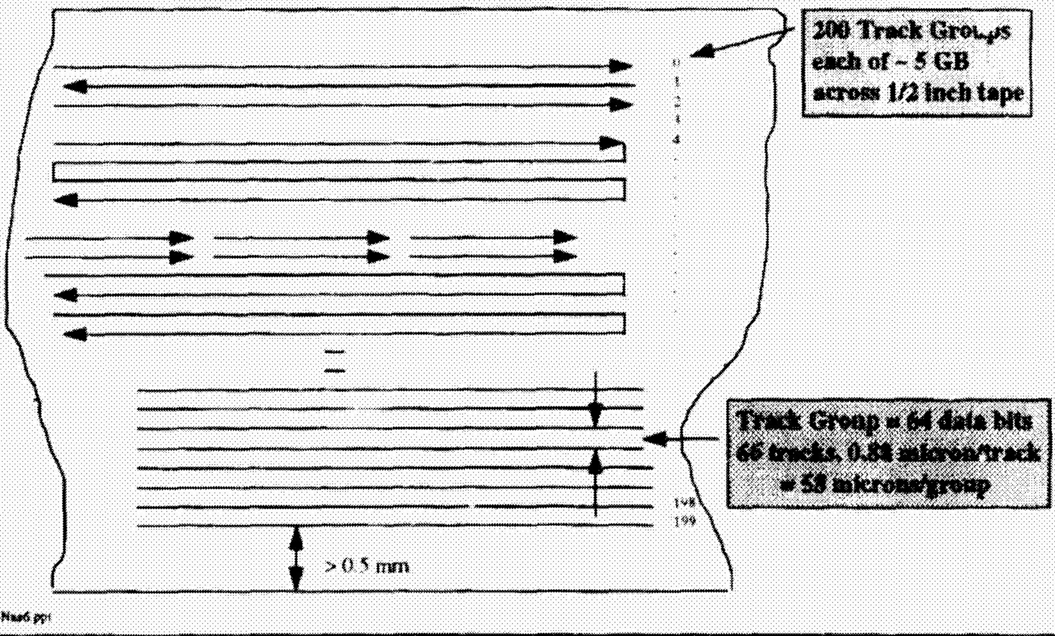
DIMENSIONS IN MICRONS

For any N x 8 sized array with equal spacing in both axes the rotation angle to achieve equal spacing of all tracks in the recorded track group is Tan^{-1} 1/8 or 7.125 degrees. Each track group is separated from those adjacent by two track spaces so the 200 track groups, each effectively 66 bits and 58 microns wide, are evenly spaced across the tape width as shown in **Figure 6**. Read/write access to any one track group is individually achieved by vertically positioning the optical head across the tape width by means of a stepper motor.

Figure 6:100-T FORMAT

LOTS TECHNOLOGY

200 BOT. x 64 data bits, Serpentine, Linkable, Bi-directional

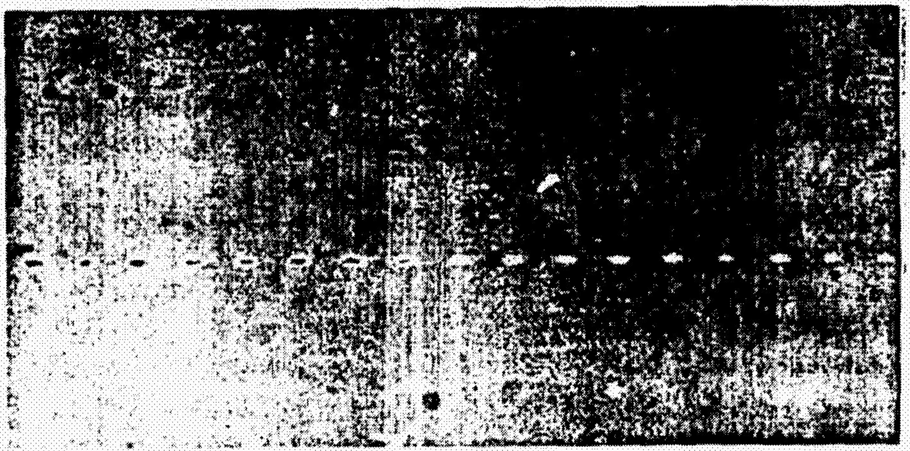


The key requirements for the holographic beamsplitter include generating the desired angular spread for the two dimensional beam array while obtaining a high diffraction efficiency and uniformity of output into each beam. An initial test hologram has been fabricated providing a 3x11 beam array test pattern and shows a total efficiency of 73% in the main array and a beam to beam intensity uniformity of better than 5%. Work continues on improving the holograms with the goal of achieving an efficiency of greater than 90% during the coming year.

A breadboard tape transport has been designed and fabricated and is being used for initial writing tests on phase change optical tape media supplied by Kodak. These tests indicate a power requirement to write a half micron wide line of about 0.4 milliwatts per meter per second of tape speed above four meters per second. Below four meters per second increased write power is required to compensate for thermal diffusion of the write energy.

A tape speed of eight meters per second therefore requires 3.2 milliwatts at the media for each bit track. Sixty four simultaneously written bit tracks are therefore expected to require about 205 milliwatts of power at the media. With an optical system efficiency of nominally 75% excluding the hologram and a hologram efficiency of 73%, a 400 milliwatt laser will deliver 400 x 0.73 x 0.75 = 219 mW of optical power to the media, sufficient for writing 64 tracks at the required tape speed. **Figure 7** shows a single bit track 0.6 microns wide written at two meters per second with a 500 KHz modulation rate imposed via one element of the array modulator.

Figure 7: 500 KHz Written Bit Track



Phase Change Media, 2 meters/sec., 0.6 micron wide, 4 micron/cycle

96NAS7.ppt

Summary and Conclusion

Significant progress has been made during the last year toward validating both the recorder design approach and the individual component technologies necessary to achieve the 100 MB/sec. data rate. The demonstrated performance of the tape transport, the laser, the holographic optical element, the modulator array, the system optics and the media indicate that no serious technological impediment exists to engineering a digital optical tape recorder able to read/write digital data at rates of at least 100 MB/sec. and user data capacities of a terabyte in a single '3480' style cartridge. The next year's effort will be directed at improving the individual components and integrating them into a technology demonstration at the full data rate.

NEXT
DOCUMENT

RAID Disk Arrays for High Bandwidth Applications

Bill Moren
Ciprico, Inc.
2800 Campus Drive
Plymouth MN 55441,
bmoren@ciprico.com
612-551-4000
FAX: 612-551-4002

Introduction

High bandwidth applications require large amounts of data transferred to/from storage devices at extremely high data rates. Further, these applications often are 'real-time', in which access to the storage device must take place on the schedule of the data source, not the storage. A good example is a satellite downlink - the volume of data is quite large and the date rates quite high (dozens of MB/sec, typically). Further, a telemetry downlink must take place while the satellite is overhead; once it passes over the horizon the telemetry is lost forever.

A storage technology which is ideally suited to these types of applications is RAID (Redundant Arrays of Independent Disks). The concepts of RAID were presented in an academic paper from the University of California's Berkeley campus in the mid-1980s. This paper (often referred to as the 'RAID paper') offered five different architectures, colloquially referred to as the 'RAID levels'. Each RAID level, numbered one through five, defined a different methodology for using multiple disks grouped together to improve performance and offer redundancy. Each of the levels had distinct strengths and weaknesses. It is a fallacy to believe the RAID levels with higher numbers (e.g. RAID-4 versus RAID-2) are superior; the ideal RAID level for an application varies with applications - one application may find RAID-1 best suited, RAID-5 for another, and yet another application's best choice may be RAID-3.

RAID Levels

RAID-1 is classic disk mirroring, in which every disk has a mirror image of its data stored on another disk. This level was the frame of reference in the RAID paper. Mirroring has been around for some time, primarily in mainframe computing. Its strengths are redundancy and performance. Any single drive in any given data pair may fail and the disk system will remain accessible, though at a reduced performance level. Because there are two disks for any given piece of data, read performance is quite good as any two arbitrary requests for a single logical disk can be serviced simultaneously on two physical disks. However, the cost for mirroring is quite high - essentially a 100%

premium since every disk is duplicated. The power, cooling, and packaging costs are also quite high. Reliability is also halved because of the duplication of disks.

RAID-2 and -3 stripe user data across a group of data drives (typically four or eight drives per group). Every block of user data is striped, typically a byte at a time, resulting in all the data disks servicing every user data request in parallel. This results in extremely high data transfer rates, since multiple disks are transferring data simultaneously. RAID-2 and -3 differ in their redundancy methodologies. RAID-2 uses multiple disks to implement a Hamming error detection and correction code. The codes stored on a RAID-2's redundant disks were generated from the data on the data disks. RAID-3 uses a single redundant disk to store a error correction code generated by calculating the logical exclusive-or of the data on the data disks. Because RAID controller technology doesn't require the use of a Hamming code to detect a failed drive, RAID-2 hasn't found commercial acceptance as it is more costly than RAID-3.

RAID-4 and RAID-5 also stripe user data across a group of data drives. However, instead of striping every block of data across all drives, each block (or sometimes groups of blocks) is stored entirely on an individual disk. This results in good transaction performance as each disk in the group can service separate requests for individual blocks, simultaneously. RAID-4 and -5 differ in the methodology used for storing the error correction codes. Both use the exclusive-or code as used in RAID-3. RAID-4 dedicates one drive for the error correction codes while RAID-5 rotates the codes throughout all drives in the array. RAID-5 has better write performance because of this rotation as there is less contention for access to the redundant codes.

The Right RAID Level for High Bandwidth Applications

Real-time, high bandwidth applications require the following from disk storage: high sustained data transfer rate under all normal operating conditions. Of all the RAID levels, only RAID-3 fits the profile.

RAID-4 and RAID-5 don't fit because their performance characteristics are designed for delivering a large number of independent requests (high I/Os per second). These RAID levels operate best when each disk is servicing a separate request. However, high bandwidth applications are characterized by large sequentially stored data sets. For such data sets, transfer rate (measured in MB/sec) is the important metric, not I/Os per second. Also, both RAID-4 and RAID-5 have severe performance degradations after a drive failure, which is considered a normal operating condition in RAID disk arrays. For real-time applications this is unacceptable as it is imperative that the RAID subsystem be able to service any request, at any time, regardless if there has been a drive failure.

RAID-3 fits for two primary reasons. First, because all user data is striped across all drives, transfer rate is very high. This is true for either reading or writing. In general, a RAID-3 disk array will have a sustained transfer rate equal to the product of sustained transfer rate of the disks used in the array and the number of data drives in the array. Second, RAID-3 doesn't suffer any performance degradation after a drive fails. Because all of the drives are accessed for each data request, there always is sufficient information being transferred from the array that can be combined with the error correction code (which is also always transferred on every data request) to generate the failed drive's data. Special hardware on a RAID-3 controller is able to perform the failed drive's data reconstruction on-the-fly, with no performance loss.

Other Factors to Consider

In addition to the media redundancy inherent in RAID, other subsystem components should be protected against failure. For instance, most RAID subsystems include AC to DC power supplies. These units have failure rates similar to disk drives. Power supply redundancy should also be considered. One good approach is to incorporate dual, load-sharing power supplies in the RAID subsystem. Each power supply has sufficient power to operate the entire subsystem in case the other should fail.

Another subsystem component worth considering for redundancy are the cooling fans. Fans, being a mechanical device, are also prone to failures. A RAID subsystem can incorporate redundant fans to protect against overheating in case of a fan failure.

All redundant components, drives, power supplies, and cooling fans, can support 'hot swapping'. Hot swapping is the ability to replace a failed component without shutting the subsystem down or taking it offline. Most hot swap components will be housed in canisters or carriers which slide into the RAID subsystem.

Another factor to consider is the host interface. The host interface directly affects the performance a RAID disk array will be able to deliver. The most common interface found is SCSI-2. It is a 16-bit wide parallel interface which clocks data at 10 MHZ for a burst rate of 20 MB/sec. Sustained rates of over 19 MB/sec are possible with SCSI-2 RAID-3 disk array.

The successor to SCSI-2, SCSI-3, includes a performance improvement to 40 MB/sec. This capability, sometimes referred to as UltraSCSI, is backward compatible with SCSI-2. SCSI-3 uses the same 16-bit wide parallel interface as SCSI-2, but data is clocked at 20 MHz, instead of SCSI-2's 10 MHz. UltraSCSI RAID-3 disk arrays are capable of sustained data rates in excess of 38 MB/sec.

Another interface which offers excellent high bandwidth performance is Fibre Channel. This is a serial interface which is clocked at 1 Gbit/sec with a sustained interface capability of 100 MB/sec. Fibre Channel is not physically compatible with SCSI-2 or -3 but is software compatible. Fibre Channel supports a number of software protocols which are encapsulated in 'frames' which are the data packets that are transferred between Fibre Channel nodes. SCSI is one of the software protocols supported. The first Fibre Channel compatible RAID-3 disk arrays are becoming available in 1996 with sustained data rates of nearly 90 MB/sec.

A good example of high bandwidth RAID-3 disk arrays are those available from Ciprico, Inc. (Minneapolis, MN). Ciprico offers a full line of high bandwidth disk arrays which are well suited to real-time, high bandwidth applications. Ciprico's arrays all offer high data transfer rate, no performance degradation after drive failures, and media redundancy. There are a number of interface, redundancy, and capacity options, designed to support a variety of applications. Table 1 summarizes the capabilities of Ciprico's disk array.

Model	Interface	Burst Transfer Rate	Sustained Transfer Rate	Redundancy	Hot Swap
6500	UltraSCSI	40 MB/sec	38 MB/sec	Drives	No
6700	SCSI-2	20 MB/sec	19 MB/sec	Drives Power	YES
6900	UltraSCSI	40 MB/sec	38 MB/sec	Drives Power	YES
7000	Fibre Channel	100 MB/sec	80+ MB/sec	Drives Power Fans	YES

Table 1 - Ciprico's RAID-3 disk arrays offer a variety of performance and redundancy options. Users can select an array which best fits their application.

Summary

High bandwidth applications require high sustained data transfer rates under all operating conditions. RAID storage technology, while offering differing methodologies for a variety of applications, supports the performance and redundancy required in real-time applications. Of the various RAID levels, RAID-3 is the only one which provides high data transfer rate under all operating conditions, including after a drive failure.

NEXT

DOCUMENT

RAID Unbound: Storage Fault Tolerance in a Distributed Environment

Brian Ritchie
Alphatronix, Inc.
4022 Stirrup Creek Drive, P.O. Box 13978
Research Triangle Park, NC 27709
e-mail: sales@alphatronix.com
www.alphatronix.com
919-544-0001 or 800-849-2611
FAX: 919-544-4079

Introduction

Mirroring, data replication, backup, and more recently, RAID are all technologies utilized by corporate America to protect and ensure access to critical company data. IS managers have taken comfort in knowing that critical data was being copied and safely stored for future access in the event of an equipment failure, operator or user error, or even worse, a iocal disaster. If one of these events were to occur, this critical data could still be transparently accessed, or at least recovered, and operations would continue. Or would they?

A whole new set of problems have arisen as a corporation's data becomes more and more geographically distributed. Do conventional protection techniques — mirroring, data replication, RAID, backup etc. — truly provide the level of data protection and data accessibility needed under this changing environment? The answer to this question is — probably not. Each of these technologies provides important benefits; but each has failed to adapt fully to the realities of distributed computing. The key to data high availability and protection today is to take these technologies' strengths and "virtualize" them across a distributed network.

Traditional Backup and High Availability Methods

RAID and mirroring offer high data availability, while data replication and backup provide strong data protection. If we take these concepts at a very granular level (defining user, record, block, file, or directory types), and then liberate them from the physical subsystems with which they have traditionally been associated, we have the opportunity to create a highly-scaleable network-wide storage fault tolerance. The network becomes the virtual storage space in which the traditional concepts of data high availability and protection are implemented without their corresponding physical constraints. Let's look at the evolution of these technologies.

The concept of RAID has existed for several years, giving users the ability to copy and/or stripe data to an array of disks. Because of the redundant design, data remains accessible even if an individual disk should fail. But what if the server fails? Because the disk array is located in a single physical location, its data is vulnerable to a "single point of failure."

Using RAID, no alternate "safe" locations from which to retrieve data exist, making it incomplete as a reliable, uninterrupted storage fault tolerant solution.

Disk mirroring, like RAID, enables IS managers to "duplicate" or "mirror" critical data to a second disk so that the data can later be retrieved in the event of a primary disk failure. This method also poses several drawbacks. For one, mirroring is not efficient; the entire contents of the disk must be duplicated. Neither RAID nor mirroring offers the level of granularity needed to define which users, records, blocks and/or files receive top-level protection. Additionally, mirroring, like RAID, is constrained by physical location and vulnerable to a single point of failure.

Backup and data replication technologies have been used for some time to protect mainframe and workgroup level data. Historically, they ensured that data was always available, but how do you back up a file system when it is hundreds of gigabytes to terabytes in size, or when you're collecting hundreds of megabytes of data per day? The window of time available to perform these tasks is no longer enough. This magnitude of data can't be backed up during regular business hours because of the already high level of network traffic versus the network pipe size available.

Emerging SFT Technology

Although there are still benefits to RAID, mirroring, data replication and backup, today's storage needs demand what these technologies can't provide — high data protection and availability across the entire enterprise. Administrators are looking to a new generation of software to take high data availability and protection concepts one step further. A new concept of network-wide storage fault tolerance (SFT) has emerged, which utilizes the entire network's storage resources, giving administrators the ability to store multiple copies of information at multiple sites in the enterprise, even at remote storage vaults.

SFT technology evolved from a need to ensure that data was consistently and readily available to key users. If key users cannot get consistent and immediate access to their critical data, then individual productivity suffers, meaning loss of money to a company. SFT software sends key user data not to a single, same site location (as in RAID and mirroring), but to various storage devices located throughout a company's computing environment, eliminating any single point of failure (Figure 1). In this way, it offers an enterprise-wide level of high data availability and data protection rather than traditional subsystem-specific security.

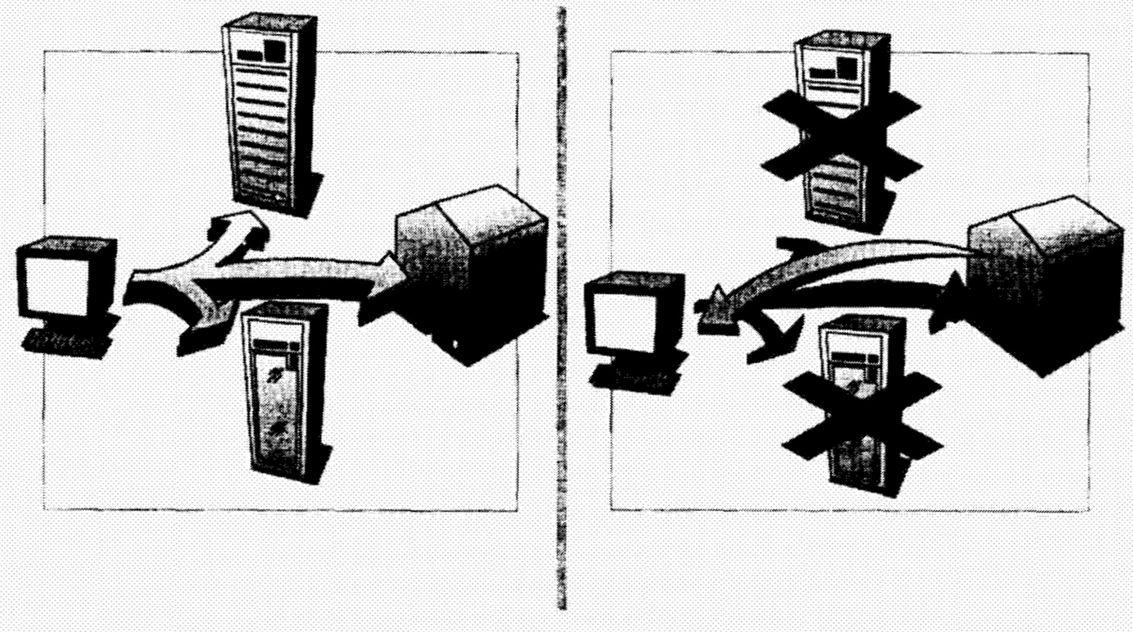


Figure 1. Data high availability and disaster recovery capability for key user data using storage fault tolerant software.

SFT software gives data control back to the administrator and saves time in the process. Possible storage destinations (such as RAID or optical and tape libraries) are defined by administrators based on various criteria including the type of data being stored. The most valuable data (for example scientific information or mathematical calculations) can be sent to two, three or more local storage centers or to a remote site that's considered "most secure." The software recognizes every storage site on the network and marks each site based upon its storage performance. This process ensures that whenever data must be accessed from a secondary source, it is retrieved automatically and immediately from the highest performance device available, regardless of where the device is located. When data is requested, the software automatically and transparently retrieves it without any "downtime" associated with restoring "save sets" to a drive. Because the software is managed centrally, the labor associated with individually managing distributed RAIDs or disk mirroring systems no longer exists. And because the data is always accessible, replication and/or backup processes may be eliminated, or performed much less frequently than before.

The "Virtual" Single Storage Environment

Unlike traditional technologies, Storage Fault Tolerant systems work to make a distributed environment look, act and feel, to the administrator, like a single logical environment (Figure 2).

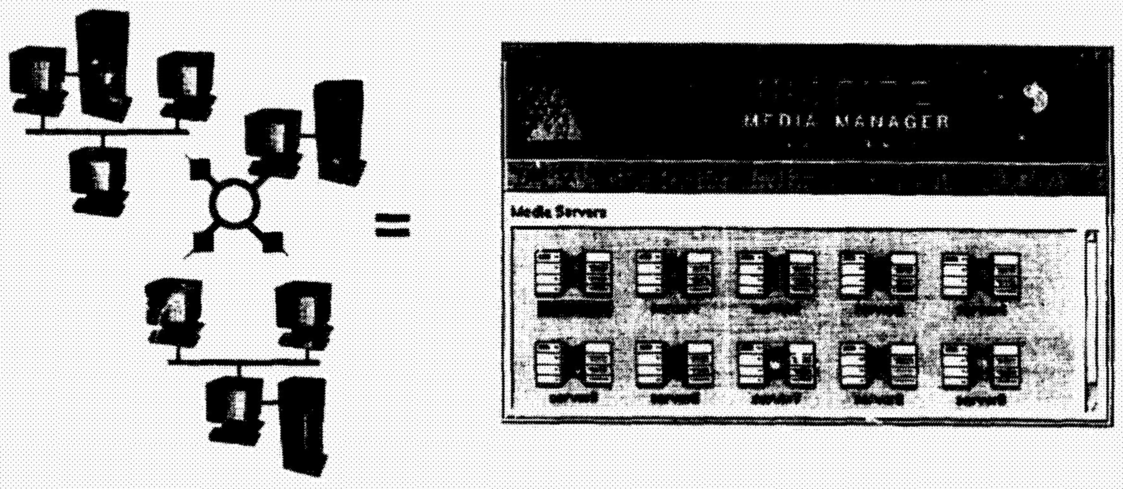


Figure 2. Creating a "virtual single storage environment" from multiple, distributed p ·l systems using storage fault tolerant software

This is important for several reasons. First, administrators are able to see every server and peripheral device connected to the network and can create a higher-level storage fault tolerant environment, across this network, than was possible before. Second, because administrators can select which users and what data get top protection priority, new users and data types can be easily defined and added from a central location. Third, all storage resources (across the enterprise) are visible through a single graphical user interface (GUI) for easy viewing, manipulation and management by the administrator. This is crucial in determining where, when, and to how many sites data should be sent. Fourth, administrators are able to balance their data load across a heterogeneous and distributed network from one common GUI.

SFT technology makes use of RAID or mirroring hardware already in place and offers a whole new set of capabilities and whole new level of control for systems administrators. It makes the need to purchase new hardware for mirroring or a new RAID box each time one fills up obsolete because the software makes every storage destination on the network available. Using SFT software affords administrators, sitting at one location, the ability to see each new server and every additional storage device, as they're connected, so that he/she can ensure the safe storage and availability of all crucial data across the enterprise; it enables a geographically distributed environment to look and act as a single logical environment, providing network-wide storage fault tolerance to key users' data.

Network-wide storage fault tolerant systems also have the potential to eliminate most, if not all, of the burden placed on traditional backup procedures. By migrating data to multiple, redundant locations, SFT systems are already, today, capable of eliminating the need to back up the bulk of a company's data. As SFT systems continue to evolve, they have the potential to entirely replace traditional backup procedures with enterprise-wide data high availability, scaleable to the requirements of today's distributed computing environments.

NEXT
DOCUMENT

SAM-FS -- LSC's New Solaris-Based Storage Management Product

Kent Angell
LSC, Inc.
380 South 200 West
PO Box 657
Farmington UT 84025
kent@lsci.com
801-451-9704

I have been involved with the computer industry for more than 20 years, and I've identified a few constants. These are change, growth, and an exponentially increasing volume of data to be managed.

My industr" niche is in data management, and for that reason I am very focused on the growth of data volume, storage density, data granularity, and I/O bandwidth. I think that the experience of Cray users over the years is a relevant case study.

In the early 1980s, Seymour Cray completed work on the Cray 2, a compact supercomputer that used a revolutionary liquid cooling system. The Cray 2 could create new data at the rate of (about) 5 to 10 megabytes / second. The processing speed was great enough that data generation and use outpaced the machines I/O channels, and the system had to be idled every few hours to allow for export of newly created data, and import of a new data set. The cycle was repeated several times a day. Data transfer was a bottleneck, and the best that could be hoped for was 1 to 2 gigabytes / day.

The Y-MP class of Cray machines provided a greatly increased I/O bandwidth, and at the same time the capacity of both vector and scalar machines was increased so that data could be created at 10, 20, 30 ... megabytes per second. All of the previously compute-bound problems became I/O bound, and this created a logistics problem, which is -- how do we move and store all of this data?

Automated Storage Management

Well, the traditional approach has been to keep our current stuff on magnetic disk, and store less current information on tape. (Stuff, that's a technical term we use to describe the other guy's digital data.) This scheme worked OK, as long as we had "manageable" data sets, and enough operators to keep the data moving. But when data volumes get into the range of 1 to 10 gigabytes per day, we must have automation to handle the sheer volume of data, to provide transparent access, to give us around-the-clock operation, and to manage access control, that is -- security.

IBM marketed one of the first automated storage management systems, based on their proprietary DFHSM software. And, the emergence of UNIX and open systems computing led to a second generation of storage management applications, such as Bump, developed at NRL, and UniTree, developed at Lawrence Livermore.

These were followed by a wave of architecturally similar products, such as Cray's Data Migration Facility (DMF), which falls into the Bump class, and products in the UniTree mold that use proprietary file systems, and operate in parallel with the UNIX file system. Both methodologies offered customers improved functionality, and satisfied many of their customer's requirements at the time.

But the marketplace has changed. UniTree and other like products were traded like baseball cards, and were not valued or supported by the new owners, as strongly as by the old.

There is much that is new in today's market, and much remains the same. Basic customer requirements have not changed; they need stability, scalability, performance, and support for the newest storage subsystems. Customers want to know that their data is secure and accessible, with redundant copies for disaster recovery. But the check-off list of required functionality has grown significantly, today's customers want more functionality than any one software product can provide, and they want it low-cost, with guaranteed updates and tech. support -- forever. .

A major issue for all customers is vendor stability, which means: will the software provider be there to support the product next year - and in five years. Many place their hopes in large companies that are presumed to be more viable, but this is inconsistent with the dynamics of the computer industry, where growth, change and innovation are fueled by products developed at start-ups and small companies. In fact, both large and small companies in the computer industry are volatile and offer volatile product lines. Whoever the provider, customers are advised to secure software that works for their environment and is supported by established systems integrators.

SAM-FS from LSC

Which brings me to my company, LSC, which stands for Large Storage Configurations. Our HSM product, called SAM-FS, is the happy result of an eclectic marriage of the best elements of the old paradigm with a new
generation of code, functionality and performance. SAM-FS is a robust, high-performance storage and archive manager, operating under Sun's Solaris 2.X operating system.

I'll have more to say about performance later, but it's important to make a point about it here. Performance is important in all aspects of HSM operation -- not just in I/O transfer rates.

LSC has designed performance into every part of SAM-FS; into file system restoration for example. In an actual test performed by DLR in Germany (that's Germany's NASA) a SAM-FS file system and a competitor's system were restored from backup media after a simulated interruption. The competitor's product required about 1-1/2 days per 100,000 files restored; SAM-FS took less than a minute per 100,000 files. We didn't do this test, the customer (DLR) designed the test and carried it out.

Don't you think that performance like that will be important to your users or customers if they are trying to get back into production after an interruption, maybe a server disk crash?

SAM-FS is a full featured HSM that operates as a file system on Solaris-based machines. The SAM-FS file system provides the user with all of the standard UNIX system utilities and calls, and it adds some new commands, i.e. archive, release, stage, sls, sfind, and a family of maintenance commands. The system also offers enhancements to the standard UFS, i.e. high performance virtual disk read and write, control of disk through an extent array, and the ability to dynamically allocate block size. This allows for very fast disk access, up to 2X faster than ufs. SAM-FS supports all RAID levels and the use of 3rd party file systems such as Veritas and On-line Disk Suite (ODS).

SAM-FS provides "archive sets", which are groupings of data to be copied to secondary storage. Archive sets can be defined (controlled) as to number of copies, when and where each copy will be stored, how long each copy will be retained, file size included (max. and min. sizes), and by VSN for each copy in the set.

As with other HSM systems, SAM-FS migrates files onto secondary media. In practice, as soon as a file is written to disk, SAM-FS will make copies onto secondary media. These files then become candidates for release from disk cache. The archiving process can be automatic or explicitly driven. This may not sound all that revolutionary, but there are some very neat things going on:

First, one to four copies of a file are dynamically and automatically written to secondary media, either automatically or by specific command. And SAM-FS provides parallel threaded operation so that all files can be written to I/O devices simultaneously.

Second, data is written to secondary media with the metadata included, and can be read independently using standard *tar* on any UNIX system.

OK, now we have the file on disk and secondary storage; what now? The SAM-FS *releaser* utility can be tasked specifically to release a file, or group of files, with immediate operation. Or the *releaser* can be programmed to release files according to predetermined criteria. Files can be specified for immediate release after archiving, or can be tagged release-never, which means they are backed-up (archived) but never leave the disk. Now, the released files are off of cache and reside only on secondary storage in a tape library, jukebox, or on media stored on shelves, possibly in a vault somewhere.

Files are staged onto cache to get them back. Think of the system as a virtual disk; when the user accesses a file, he wants it as soon as possible. To do this fast you must tune the system to take advantage of the media performance, which means to do parallel I/O at full speed. Stage requests are organized by media type and VSN, with the queue organized for most efficient access to the media. New requests are added to the queue dynamically and are placed in the most logical place based on media type and VSN. Once a request is satisfied, and if no other requests are pending, the tape or MO remains in the drive for a user-defined period of time. Then it is rewound and put away.

File access can be specified *stage-never* so as to bypass the disk cache, allowing large file access directly from secondary media, without disturbing the file mix in cache (3rd party transfers).

Users can access all or any part of a file, specifying the start of data (byte offset) and the number of bytes to retrieve. Only the specified data will be returned.

Files can be archived leaving a stub on disk cache. This allows the file to be opened and read without staging it onto disk.

When files are modified they get a new date and time and are archived as a new file. The pointer to the old version is deleted, and the media now has a hole.

Now, having holes in your media isn't all bad, and I suggest that you keep them. Because as long as that hole is present in the media, the older version of the file that the hole represents is still accessible. Sooner or later, though, you may want to recycle media. The SAM-FS *recycler* will copy the remaining files onto new media, and you can reuse the old media, or you can keep it until you don't need access to the older file versions anymore. SAM-FS provides utilities and procedures to access the older file versions.

Scalability

SAM-FS is a richly scalable storage management system. It can manage N file systems on one server, where N is a very large number limited by 64 bit architecture. The system can manage millions of files per system, though this is limited today by the speed of UNIX and its utilities. Later this year, LSC will implement a new search algorithm that removes logical and performance restrictions on the number of files.

Currently, SAM-FS supports tape and MO libraries from all major vendors, including Grau robots with mixed media tapes and StorageTek libraries with Timberline and Redwood drives.

Performance and Testing

LSC has tested the more popular tape devices under load to validate vendor claims and to determine actual performance with SAM-FS:

	Native	Compressed
DLT 2000	1.2 MByte/sec.	2.0 MByte/sec.
DLT 4000	1.5 MByte/sec.	3.5 MByte/sec.
DLT 7000	not yet available for test	
3490E	3.5 MByte/sec.	5.0 MByte/sec.
Redwood (FW SCSI)	9.5 MByte/sec.	14.0 MByte/sec.

SAM-FS scales in performance from one drive to simultaneous use of multiple drives as follows:

1 DLT 2000	1.2 MByte/sec.	1 Redwood	9.0 MByte/sec
2 DLT 2000	2.4 MByte/sec.	2 Redwood	18.0 MByte/sec.
3 DLT 2000	3.6 MByte/sec.	3 Redwood	27.0 MByte/sec.
.	.	.	.
.	.	.	.
N DLT 2000	n(1.2) MByte/sec.	n Redwood	n(9.0) MByte/sec.

This test assumes the use of multiple SCSI channels. If all drives are on the same SCSI bus, then performance will suffer

SAM-FS was also tested to determine added overhead:

Disk writes	+ 1%
Disk reads	+ 2%
Tape writes	+ 0.1%
Tape reads	+ 0.1%

For more information about SAM-FS testing and performance contact one or both of the following:

DLR Deutsche Forschungsanstalt fur Luft ung Raumfahrt e.V.
German Remote Sensing Data Center (DFD)
82234 Oberpfaffenhofen, Germany
Phone: 49 8153 282 623
E-mail: willi@dfd.dlr.de
rattei@dfd.dlr.de
how@dfd.dlr.de
Contact Names: Wilhelm Wildegger, Willi Rattei and John How

JIC-PAC ISO
P.O. Box 500
Pearl Harbor, HI 96860
Phone: (808) 471-7272
E-mail: djp@pixi.com
Contact: Dale Podoll

Summary

It is a fact of life in the storage management software business that everybody wants something more than we can deliver today. They love what they see but they also want something different or more of it. This is more of a job description than a problem, LSC is a customer-driven software development company. We add the new requested features and enhancements to our release schedule if we think its a good idea, and we get a better product over time.

In the last release several customer-requested features were added:

API: Interface to access SAM-FS from a user application.
Both client and server versions are provided.

Grow fs: Enables additional disk cache devices to be added to a file system as it grows. Additional disk devices can be added without system reinitialization.

Stage All: Provides Associative Staging. Files in a common directory with this control set are all staged when any one of the set is accessed.

In the next release are more customer-requested features:

Checksum Verification: Enables users to verify that data on removable media has not been altered.

Recycling for Archive Sets: Allows recycling based on archive set thresholds, in addition to recycling based on robot thresholds.

API Enhancements: Making the API jump through hoops.

The Last Word

A major reason that customers want HSM is for disaster recovery, and SAM-FS has some unique capabilities for data recovery in case of catastrophic failure:

First, if the disk cache dies, the system can be reinitialized and back on line in a matter of 1 to x minutes depending on the number of files and speed of the tape device reading the last inode dump. Average performance is 1 minute / 100K files in the SAM-FS file system.

Second, if a file is damaged, the archive copy is used. If the archive copy is damaged, SAM-FS will look for a second copy. Older versions (holes) can be accessed using the SAM-FS interface.

Third, if the primary server installation is destroyed, the system can be reinitialized using replacement hardware and a backup copy of the archive maintained specifically for this contingency in a remote vault.

Last, it is important to tune the storage management system with disaster recovery in mind, i.e. files that have not been migrated to secondary storage will be lost.

People who try SAM-FS like it, so I'd like to ask you to give it a try. We have demo software available if you want to wring it out.

NEXT
DOCUMENT

Use of HSM with Relational Databases

Randall Breeden, John Burgess and Dan Higdon
FileTek, Inc.
9400 Key West Avenue
Rockville, MD 20850
jgb@filetek.com
Tel: 301-251-0600
Fax: 301-251-1990

Hierarchical Storage Management (HSM) systems have evolved to become a critical component of large information storage operations. They are built on the concept of using a hierarchy of storage technologies to provide a balance in performance and cost. In general, they migrate data from expensive high-performance storage to inexpensive low-performance storage based on frequency of use. The predominant usage characteristic is that frequency of use is reduced with age and in most cases quite rapidly. The result is that HSM provides an economical means for managing and storing massive volumes of data.

Inherent in HSM systems is system managed storage, which has the system performing most of the work with minimum operations personnel involvement. This automation is generally extended to include:

- Backup and recovery
- Data duplexing to provide high availability
- Catastrophic recovery through use of off-site storage

Types of HSM

HSM can be broken into two main categories based upon the level of the objects that are accessible through the HSM system: file level and record level.

File Level

Most of today's HSM systems operate on a magnetic disk file level basis. In these HSM systems, when data is migrated off magnetic disk, the associated directory entry remains while the actual data is moved down the hierarchy. When the end-user or end-user application needs the migrated data, the file containing the data is opened, and the data is migrated back to magnetic disk. For example, if transaction information for a deposit that occurred a year ago is required, the HSM system copies the entire file back to the magnetic layer, and then the application extracts the specific information it needs.

Record Level

The second type of HSM system operates on a record level access basis. In these HSM systems, data is written with one or more keys or a record number. Then when the end-user or end-user application needs information, the file containing the required data is opened, a key or record number is supplied, and the associated record is transferred. The main difference between file and record level HSM systems is that in record level HSM systems, data can be accessed directly from the storage media without having to be restored to the magnetic layer first. This is particularly useful when storing billions of small objects such as user transactions, phone calls, and statements.

The following table compares the performance of file and record level access HSM systems.

Action	File HSM (seconds)	Record HSM (seconds)
Mount Tape	15	15
Copy Data to Mag (500 Mbytes)	100	NA
Perform high-speed search for block	NA	10
Select 1 Record	1	1
Total	116	26

The preceding table shows a significant performance advantage for record level HSM when only a small object is needed. This is even more significant when optical disks are used instead of tape. This performance improvement can make the difference between being able to provide an online response versus a batch and call back response. Another significant advantage is that the storage drives used to support the accesses are in use much less for each request enabling many more requests to be processed per day.

Record level HSM has been used in mainline storage management for a number of years for microfiche replacement, online report viewing, IBM VSAM archiving and application-based database extension.

HSM In Databases

HSM has seen little use with databases. Only small databases are built on the file system enabling the use of file level HSM. In these cases, the delay required to return the file (table) usually makes it impractical.

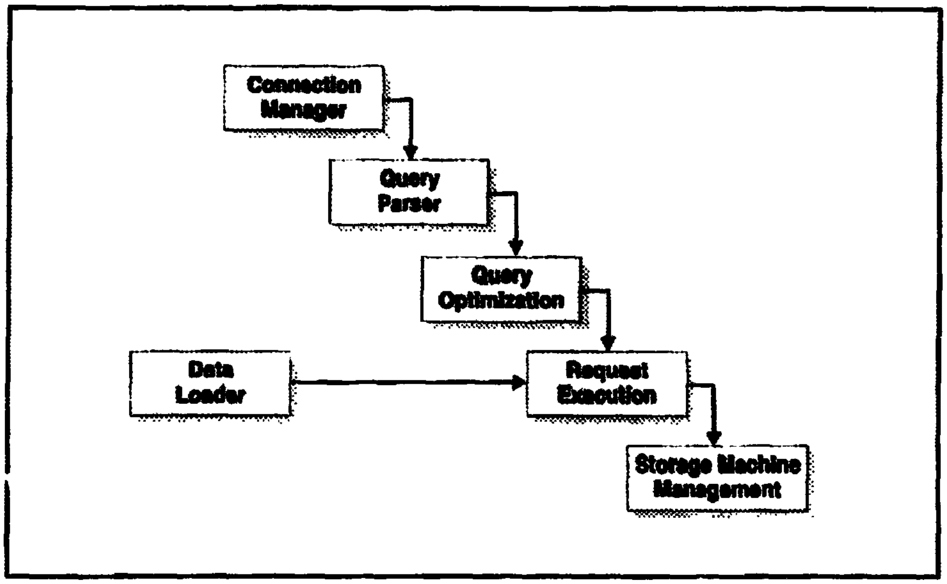
StorHouse™ System

StorHouse is the first relational database system that was developed to be fully integrated with a record level HSM system (DB/HSM™). It is built on the proven base of the FileTek® Storage Machine® (SM) system, which has been in operational use in nearly one hundred sites for managing close to 200 TBytes of online storage.

StorHouse has a high volume data loader for Direct Channel loads from mainframes and FTP loads from the network. It can process 10s of GBytes of table data per day and concurrently build all required indexes. It can build massive tables spanning many years and holding 10s of TBytes. Both hash and value indexes are supported to enable fast exact match retrievals and range-based retrievals. Indexes are multilevel and can reside separately in the storage hierarchy. This enables indexes to reside on high performance storage (RAID or optical) while data resides on less expensive storage (optical or tape).

StorHouse contains its own SQL query processor, optimizer, execution manager and database gateways. The SQL query processor, optimizer and the Storage Machine ensure that SQL queries are processed such that minimal use of robotics (optical and tape) is required. This support includes the use of large magnetic disk performance buffers that enable the storage of 100s of GBytes of the most active portions of indexes or tables to further enhance performance. These performance sensitive capabilities are extremely important because database queries executed against very large databases (VLDBs) can be very demanding.

The following diagram illustrates the various StorHouse components.



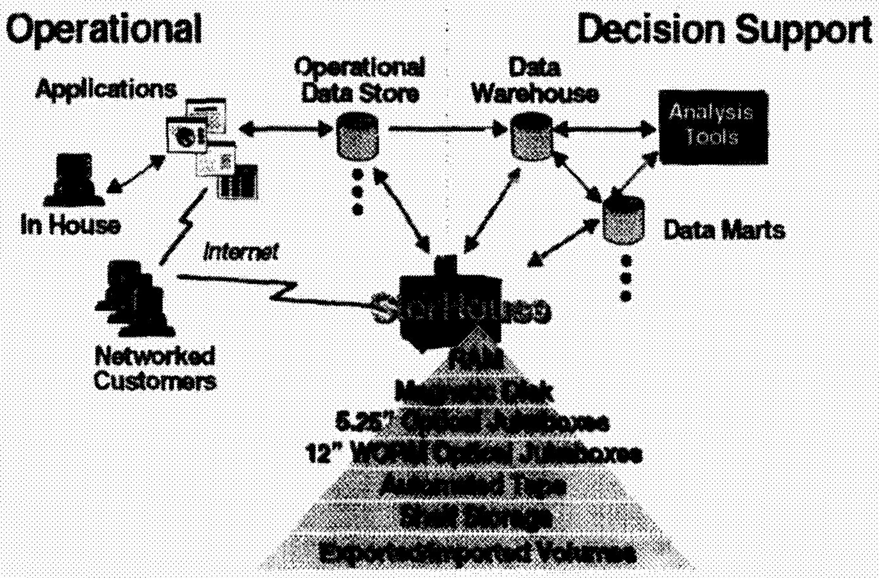
Database gateways provide access to StorHouse from many different database systems. Today, StorHouse supports DB/2® using DRDA™, EDA/SQL™ and ODBC™ In the future, StorHouse will support several other yet-to-be-announced middleware standards. StorHouse and the gateways provide for full sharing of data from different database environments. For example, data stored from MVS® DB/2 can be accessed by ORACLE® environments. This open query capability enables ad hoc queries to be processed online in support of all operational systems.

StorHouse will have a high volume data extractor that can access 100s of GBytes per day for bulk loading into RDBMSs or analytical tools. This will provide data for decision support applications whether they be OLAP or Data Mining.

Summary

StorHouse provides a low-cost storage alternative for RDBMS data using the Storage Machine's automatic managed storage hierarchy. StorHouse eliminates the need for separately archiving SQL databases to tape and supports SQL access to very large and ultra large databases. With standard protocol access from a variety of computing platforms, StorHouse expands the media options by migrating RDBMS data tables from expensive mainframe DASD and client/server magnetic disk to lower-cost reusable or permanent storage. By providing concurrent access to relational data from multiple host cnvironments, applications can truly share data without having to maintain multiple copies. This improves service, reduces the cost of magnetic storage, frees up existing magnetic storage for other applications and eliminates the use of tape as an additional archive method for c'atabase data. Furthermore, network and chcnnel activity is reduced because StorHouse returns only the requested result set.

The following diagram shows StorHouse's role in an information technology environment.



NEXT
DOCUMENT

RAID-S T.·hnical Overview: Raid 4 and 5-Compliant Hardware and Software Functionality Improves Data Availability Through Use of XOR-Capable Disks in an Integrated Cached Disk Array

Brett Quinn
EMC Corporation
Hopkinton, MA 01748-9103
Internet: quinn_brett@isus.emc.com
Web Page: www.emc.com
Telephone: 508-435-1000
Fax: 508-435-8903

1. Introduction

1.1 Objective and Scope

The purpose of this paper is to provide a technical description of RAID-S. It is intended to give the reader an understanding of how RAID-S is architected and implemented in the EMC Symmetrix 3000/5000 series Integrated Cached Disk Array. Topics include a RAID-S taxonomy, configuration considerations, operational characteristics, performance, and implementation guidelines.

It should be noted that the RAID Advisory Board granted EMC's petition to use the conformance logo for RAID Levels one, four, and five for the Symmetrix series of ICDAs. Use of the conformance logo for RAID levels four and five were also granted for the Extended On-line Storage ICDAs in June 1996. Symmetrix is considered RAID Level one-conformant when configured with mirrored devices, RAID Level four-conformant when RAID-S is configured without Hyper-Volume Extension, and RAID Level five-conformant when RAID-S is configured with Hyper-Volume Extension.

The Symmetrix series of Intelligent Cached Disk Arrays represent a family of information storage and retrieval systems available in a broad range of capacities to address current and future business and scientific requirements. Systems provide instant and dependable access to mainframe and open platforms. For further details refer to EMC's web page.

1.2 What is RAID-S?

1.2.1 Improving data availability

RAID-S (Redundant Array of Independent Disks-Symmetrix) is a combination of hardware and software functionality that improves data availability in Symmetrix 3000 and 5000 series ICDAs by using a portion of the array to store redundancy information. This redundancy information, called *parity*, can be used to regenerate data should the data on a disk drive become unavailable.

1.2.2 Flexible availability options

RAID-S is the newest RAID solution to be delivered for the Symmetrix ICDA. RAID-1, also called *Mirroring*, was first delivered in 1991. Compared to a mirrored Symmetrix, RAID-S offers EMC users more usable capacity than a mirrored system containing the same number of disk drives. Also, with the introduction of RAID-S, users can now select the level of protection they desire for data stored in the Symmetrix. Within the same Symmetrix system, data can be protected via RAID-S, Mirroring, SRDF, and/or Dynamic Sparing.

1.2.3 Technological innovation

RAID-S employs the same technique for generating parity information as many other commercially available RAID solutions, i.e., the Boolean operation *EXCLUSIVE OR* (XOR)[1]. However, EMC is the first vendor to reduce the overhead associated with parity computation by moving the operation from controller microcode to the hardware on the disk drive itself. This is done through the use of XOR-capable disk drives. This also positions RAID-S to benefit from future improvements in internal disk subsystem communications protocol performance when SCSI is supplanted by fiber channel technology.

1.2.4 Prerequisites

RAID-S is transparent to the host operating system. The prerequisites required for RAID-S are a 3000/5000 series Symmetrix with XOR capable disk drives and the appropriate Symmetrix microcode level.

2. RAID-S Taxonomy

Like most Symmetrix features, RAID-S introduces new terminology and concepts that need to be clearly understood to properly describe the functions and components of

RAID-S. Figures 1 and 2, will be referenced in the following discussion of RAID-S terms.

2.1 Group

A RAID-S group is the set of four or eight (EOS systems only, see section 2.7) physical disks within a Symmetrix system that are related to each other for parity protection. Current implementation requires that all members of a RAID-S group must be attached to the same disk director. Figures 1 & 2 both depict RAID-S groups of four physical devices each. Note that each of the four disks are on a different Disk Director SCSI bus.

2.2 Logical Volume

A logical volume is a unit of storage implemented on a single Symmetrix disk drive. When Hyper-Volume Extension (HVE) is not used, the size of a logical volume is usually the same as a physical volume. With HVE, up to eight logical volumes can exist on a physical volume.

2.3 Rank

A rank is the set of logical volumes related to each other for parity protection. Each RAID-S group supports a minimum of one rank, and with HVE enabled, a maximum of eight ranks. Figure 2 shows a RAID-S group consisting of four 9 GB drives with four ranks defined across the group. A rank is the "horizontal layer" of logical volumes and utilizes all four SCSI paths attached to a disk director.

A rank is equivalent to a "redundancy group stripe" as defined by the RAID Advisory Board.

2.4 Data Volume

A data volume is similar to a traditional logical volume in Symmetrix terminology. It is the "virtual volume" image presented to the host operating system and defined as a separate unit address to the host. All data volumes within a rank must be the same size. There can be a maximum of 512 data volumes in a Symmetrix.

It is important to note that RAID-S **does not** "stripe" data across members of a rank as is done in traditional RAID implementations. Each data volume emulates either a complete 3380 or 3390 device or a complete FBA logical volume mapped to an Open Systems host. This is a key differentuator because it allows the group to sustain the loss of more than one member and still service requests from all the surviving members. In RAID 4/5 implementations which stripe data, the loss of more than one member would result in data loss for the entire group.

This "direct" mapping of disk images to disk drives also allows standard performance and tuning techniques to be used to manage the volumes in the rank.

2.5 Parity Volume

A parity volume is a logical volume which holds the parity information for the rank. It must be the same size as the data volumes it supports. Parity volumes do not have unit addresses and are transparent to the host software. As is true with M2[ii] volumes in a mirrored Symmetrix, parity volumes are not included in the 512 device limit within a single Symmetrix system. In fact, the parity volume is referred to as an "M2" volume and is associated with three "M1" data volumes in a 3:1 rank. This is illustrated in figure 1.

When using HVE, parity volumes are distributed amongst the members of a RAID-S group, as shown in figure 2. This **distributed parity** provides for improved performance over a single physical volume which could become a performance bottleneck in a heavy write workload.

2.6 Modes of Operation

2.6.1 Normal Mode

When a RAID-S rank is operating with all members functioning it is said to be operating in normal mode.

2.6.2 Reduced Mode

When a RAID-S rank is operating with one failed *data* volume it is said to be running in **reduced** mode. Parity protection is suspended for the rank. Referring to figure 1, the failure of device 00 would force the rank to operate in reduced mode. In figure 2, the failure of device 00 would cause the first three ranks to operate in reduced mode.

2.6.3 Non-RAID Mode

When a RAID-S rank is operating with one failed *parity* volume it is said to be running in non-RAID mode. As in reduced mode, parity protection is suspended for the rank. Again referring to figure 2, the failure of device 00 would cause the fourth rank to operate in non-RAID mode.

2.6.4 Regeneration

When a data volume fails, the data on that volume is reconstructed by XORing the parity volume with the remaining data volumes in the same rank. This process is called **regeneration** and is used in place of the normal READ command when one data volume has failed. The regenerated data is placed on the parity volume of the rank. Any subsequent request for the data will be serviced by the parity volume, which is acting as a data volume for the regenerated data.

Referring to figure 1, if device 01 were to fail, the data on volume B would be regenerated by computing the exclusive OR of the data on volumes A, C, and the parity volume.

2.6.5 Rebuild

When a parity volume fails, RAID protection is suspended for the rank. When the failed device is replaced as part of a service action, the parity volume is reconstructed. This process is called **rebuild**.

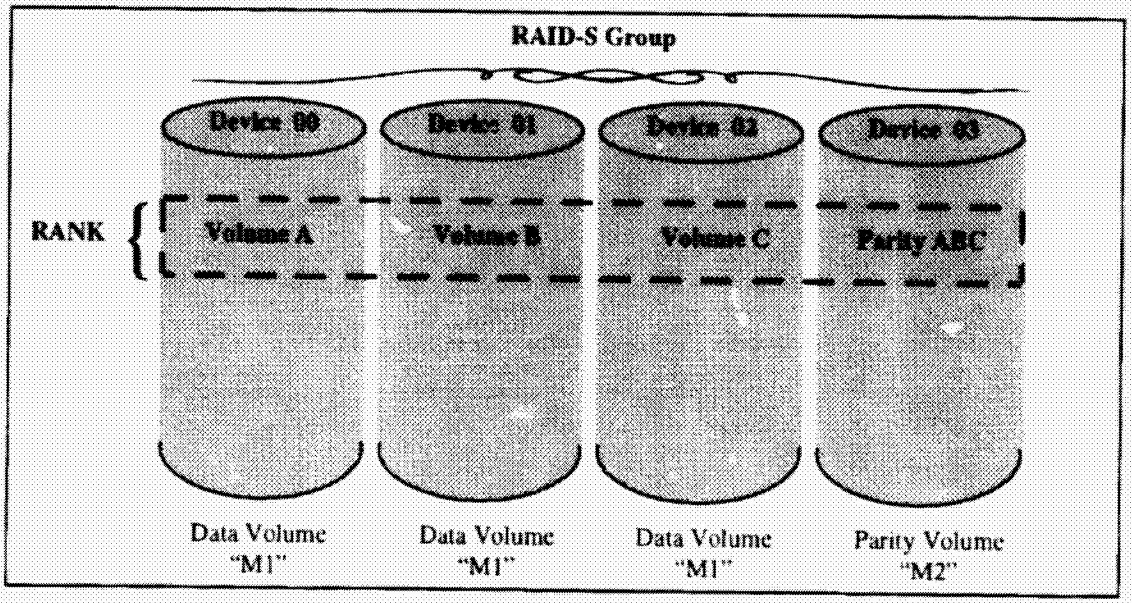


Figure 1: RAID-S Group w/o Hyper Volume

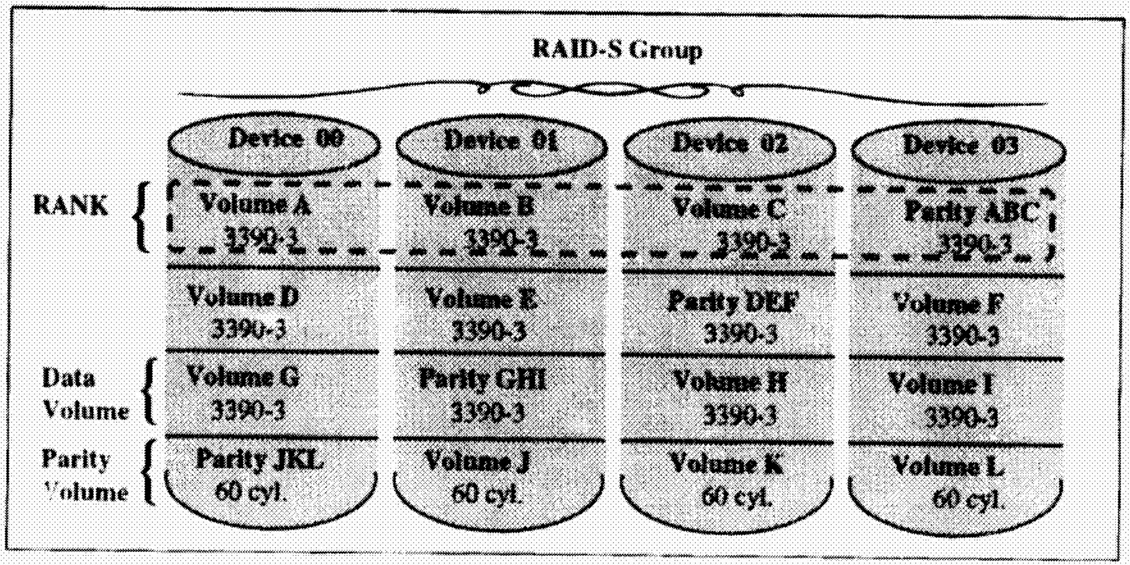


Figure 2: RAID-S Group w/Hyper Volume

2.7 EMC Extended On-line Storage (EOS)

In March 1996, the flexibility of RAID-S design and MOSAIC 2000 architecture was demonstrated with the announcement of the EOS base product (model EOS-90XX). EOS is a high capacity storage solution intended for archived data that is typically accessed in a read only mode, and where high performance is not a requirement. An EOS disk storage array offers either Dynamic Sparing or RAID-S protection for the disks in the system. The group size for EOS systems was expanded from 4 disks (3 data + 1 parity) to 8 disks (7 data + 1 parity). This has the effect of increasing the amount of storage available for user data from 75% of the array's capacity to 87.5%.

In July 1996, the EOS product line was expanded with the introduction of the EOS 9R models (EOS-9RXX). EOS 9R models offer improved performance over the base EOS models and support some of the advanced microcode features of the Symmetrix.

In both the EOS base and EOS 9R models, the number of data volumes in a rank was increased from 3 to 7. A 7+1 RAID-S group is depicted in figure 3 below. (Note that each SCSI bus now contains two members of a RAID-S group)

RAID-S in EOS systems, as in Symmetrix, can be implemented as either RAID level 4 or RAID level 5 as defined by the RAID Advisory Board. When implemented without Hyper-Volume Extension it conforms to the definition of RAID level 4. When implemented with Hyper-Volume Extension, as in figure 3, it qualifies as a RAID level 5 array.

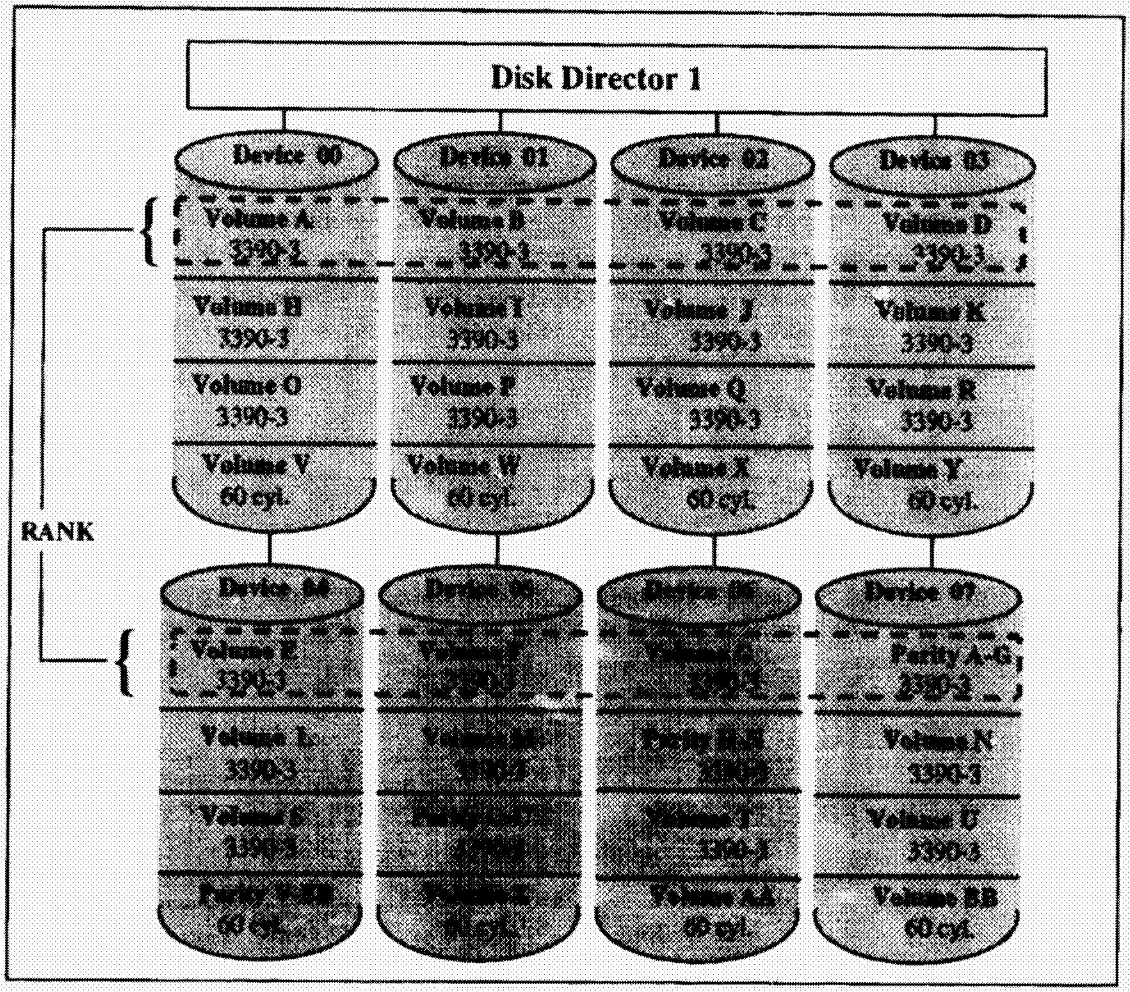


Figure 3 RAID-S 7+1 Group size in Extended On-line Storage (EOS) systems

3. Config·ring RAID-S

3.1 Host addressable volumes

In a RAID-S Symmetrix, only data volumes are host addressable. Consequently, the number of host addresses is less than the number of logical volumes defined in the Symmetrix. Using a 5100-9016 (with all volumes RAID-S protected) as an example, the number of host addressable volumes is computed as follows: (each device contains 3 logical volumes, rank size is 4 volumes, 3 data :1 parity)

16 devices X 3 logical volumes/device = 48 logical volumes
25% of volumes are parity (3:1) = 12 parity volumes
48-12 = 36 host addressable volumes

3.2 Ranks and SCSI buses

Normally RAID-S configurations will have a rank size of four, with three data volumes and one parity volume per rank. In these configurations each member of the rank will be on a different SCSI bus behind the same disk director. This improves the performance of the rank by reducing SCSI bus contention during XOR calculations. The only exception to this is EOS systems which support 7+1 ranks. These implementations support two members of a group per SCSI bus.

3.3 HVE considerations

During installation and configuration of the Symmetrix 3000/5000, parity volumes are distributed across all devices in the RAID group. Obviously, the maximum number of logical volumes that can be defined on each physical device, without having two parity volumes on one device, is four. However, the maximum number of hyper-volumes allowed, including parity volumes, remains eight.

3.4 Intermixing with Local Mirroring

RAID-S groups can coexist with mirrored pairs in the same Symmetrix. It is important to remember that RAID-S groups must be defined behind the *same* disk director, while mirrored pairs must be defined behind *different* disk directors. In addition, RAID-S volumes cannot be locally mirrored, and locally mirrored volumes cannot be part of a RAID-S group.

It is possible to dynamically reconfigure a mirrored configuration to RAID-S and vice versa.

3.5 SRDF and SDM Support

RAID-S is supported with the Symmetrix Remote Data Facility and the Symmetrix Data migrator. This support is described below.

3.5.1 Symmetrix Remote Data Facility

SRDF provides the capability to remotely mirror logical volumes to another Symmetrix system. This logical volume approach is maintained in a RAID-S environment. SRDF **does not** require that a RAID-S rank or group be remotely mirrored in its entirety. Rather, SRDF simply allows a logical data volume in a rank to be remotely mirrored to another system where it can be protected via local mirroring, RAID-S, and/or dynamic sparing. Note that parity volumes are not remotely mirrored. SRDF views this remote copy (target volume) of the data as a third copy which can be accessed via the SRDF link in the event that the local copy (source volume) becomes unavailable.

This offers the benefit of using the remote copy of a volume to access data in the event the local copy is unavailable, thus avoiding the overhead of RAID-S regeneration when accessing a failed volume.

3.5.2 Symmetrix Data Migrator

SDM is a Symmetrix microcode based product which allows the direct migration of data from an existing, installed control unit (called the "donor") to a Symmetrix (called the "target"). During a migration, the target Symmetrix is connected to a mainframe host and the donor control unit is connected to the Symmetrix. Data is then migrated from the donor to the target Symmetrix in either an on-line or off-line fashion. Parity computation can be performed during migration (the default), or after all data has been migrated to the data volumes in group.

The introduction of RAID-S protected target volumes into an SDM migration does not impact the configurability of the target Symmetrix. Donor control unit volumes are mapped to target Symmetrix volumes just as they were in a mirrored scenario.

4. RAID-S Operational Characteristics

4.1 Normal Mode operation

4.1.1 Write Operations

Fast write: As with all Symmetrix operating modes, 100% of writes are fast writes and are satisfied in the cache.

Destaging write: Write operations to a RAID-S rank are completed using a Read-Modify-Write sequence of I/O operations as depicted in figure 4 and described below.

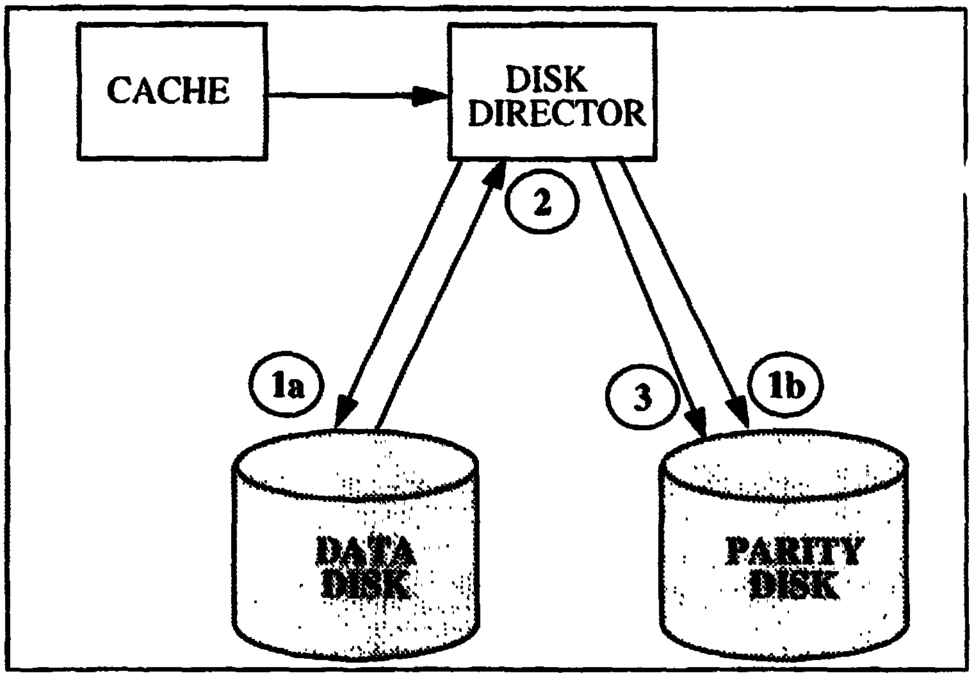


Figure 4: Read-Modify-Write Sequence

1a	The Disk Director begins the Read-Modify-Write sequence by sending the new data to the data drive using a new SCSI command called an XOR READ. This command reads the old data into the disk's buffer, XOR's it wi:h the new data (creating difference data), and writes the new data in the disk.
1b	Simultaneously, the DD sends the parity drive another new command called an XOR WRITE (command phase only). This command instructs the parity drive to read the old parity into its buffer in preparation for XORing with the difference data from 1a.
2	The difference data is sent to the DD for transfer to the parity drive.
3	The DD sends the difference data to the parity drive during the data phase of the previously issued XOR WRITE command. The difference data is XOR'd with the old parity waiting in the buffer, and the resulting new parity is immediately written to the disk.

This Read-Modify-Write sequence constitutes the "write penalty" in RAID-S. It is significantly different from the write penalty in other RAID 4/5 implementations. The typical RAID 4/5 approach requires four discrete, sequential I/O operations be executed by the controller:

1. Read old data
2. Read old parity
3. Write new data

4. Write new parity

In addition, two processing steps must be executed by the controller microcode:

5. XOR old data with new data (creating difference data)
6. XOR difference data with old parity (creating new parity)

In contrast, RAID-S requires only two discrete sequential I/O operations be executed by the controller.

1. Write new data
2. Write difference data

The design of RAID-S distributes the work of computing parity between the disk director and the disk drives, using the XOR chip and the disk level buffer. The disk containing the data volume performs the read of the old data, the XOR to compute difference data, and sends the difference data to the disk director. The disk containing the parity volume reads the old parity (at the same time that the data drive is reading the old data), XOR's it with the difference data received from the controller, and writes the new parity to the disk.

The **parallelism** introduced into the parity computation process through the use of XOR drives allows the "controller" (disk director) to do only half the number of back-end I/Os as competitive RAID solutions. This reduces the impact of the write penalty significantly and improves the overall performance of RAID-S compared to competitive implementations.

4.1.2 Read Operations

Read hits: Read hits are processed via the cache as in normal Symmetrix processing.

Read Misses: Read misses are directed to the disk drive and processed as normal Symmetrix read misses. There is no XORing of the data, and only one disk drive is involved in servicing the request. This is a significant advantage over other RAID 4/5 implementations that "stripe" data across multiple disk drives. In these implementations more than one disk drive may be required to service the request.

4.2 Reduced Mode Operations

Note: In reduced mode operations parity protection is suspended for the rank. **No new parity data is written.**

4.2.1 Failed Data Volume

Read Miss Operation: Read requests not satisfied in the cache are called read misses, and are serviced by the disk drive. When read requests are made to a failed member of a rank the data must be regenerated to service the request. The regeneration process is depicted in figure 5.

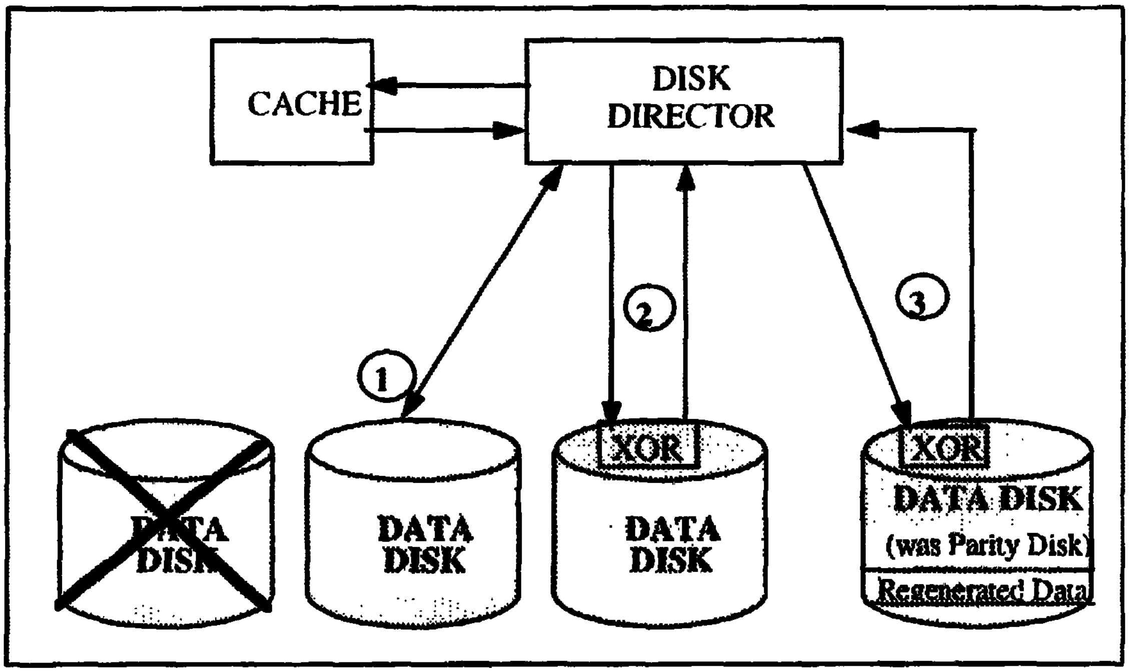


Figure 5: Regeneration Function

1	Regeneration begins with the DD issuing a standard SCSI READ command to the first surviving member in the rank and receiving the data back from the drive.
2	The data is sent to the second member using an XOR READ command with a bit set to instruct the drive to **not** write the data to the disk, but allowing it to perform the XOR computation with the data on that disk drive. The XOR'd data is sent back to the DD.
3	The DD issues another XOR READ sending the XOR'd data to the last drive in the rank (the parity drive), again with the bit set to prevent the data that was sent from being written to the disk. The data is XOR'd with the data on the disk and the result (the regenerated data) is sent back to the DD. In addition to being sent to the DD to service the request, the regenerated data is written to the parity drive as data. This improves the performance of subsequent requests for the data. The parity volume is now considered a data volume for the affected tracks.

Write operations: De-stages to the failed member of a rank first require that the data be regenerated in preparation for the write operation. The track(s) which contains the data to be written is regenerated by Boring corresponding tracks on the surviving data volumes and the parity volume. The track is then updated with the new data and written to the parity volume as data. As with read operations, this is done to improve the performance of subsequent requests for the data.

4.2.1.1 Media errors

In the event of a media error, the affected tracks will be regenerated and placed on the parity volume as data. This condition will cause the Symmetrix to place a remote service call to the Customer Support Center. The Product Support Engineer (PSE) at the support center will determine if a disk drive has been identified for replacement and dispatch a Customer Engineer. Once on site, the CE will invoke the Symmetrix Hot Replacement procedure on the service processor. The logical volumes on the disk being replaced will be placed in a not ready state and the associated ranks will begin either reduced mode or non-RAID mode of operation (depending on if the logical volume which was made not ready contained data or parity information). Once the new drive is in place, the rebuild process (described below) begins.

4.2.1.1.1 Manual Sparing

When the Symmetrix places a remote service call to report a disk drive problem, the PSE has the ability to invoke a sparing operation to a spare disk located anywhere in the system. This sparing operation will copy the data volumes from the failing disk to the spare, regenerating data where necessary to ensure a complete copy of the data volume is placed on the spare disk While the array is in this spared state no new parity is generated, and the array operates in non-RAID mode.

When the service action is complete the data volumes will be copied to the new disk and parity will be rebuilt where necessary to return the array to normal operation.

4.2.1.2 Dynamic Sparing

The Dynamic Sparing function exploits an architectural enhancement made to Symmetrix which allows up to four copies of data to be maintained in the subsystem. As a result of this change, the minimum number of spares required to provide dynamic sparing protection for RAID-S was reduced from one per disk director to three for the entire subsystem. These spares may also be used to protect local mirrors or the local copy of a remotely mirrored pair. The sparing process itself was also improved and now works as follows:

When the Symmetrix detects the pending failure of a disk drive it establishes a mirrored relationship between the data volumes in the RAID-S group and three spare drives (which can be located anywhere in the system). The data volumes on the unaffected disks, along with readable data volumes from the failing disk, are copied to the spare disks. Data from unreadable volumes is regenerated and placed on the parity volumes of the unaffected disks as well as on the spare disks (see figure 6 below). No parity data is copied to the spare disks and no parity generation occurs since **all the data is now protected via mirroring**.

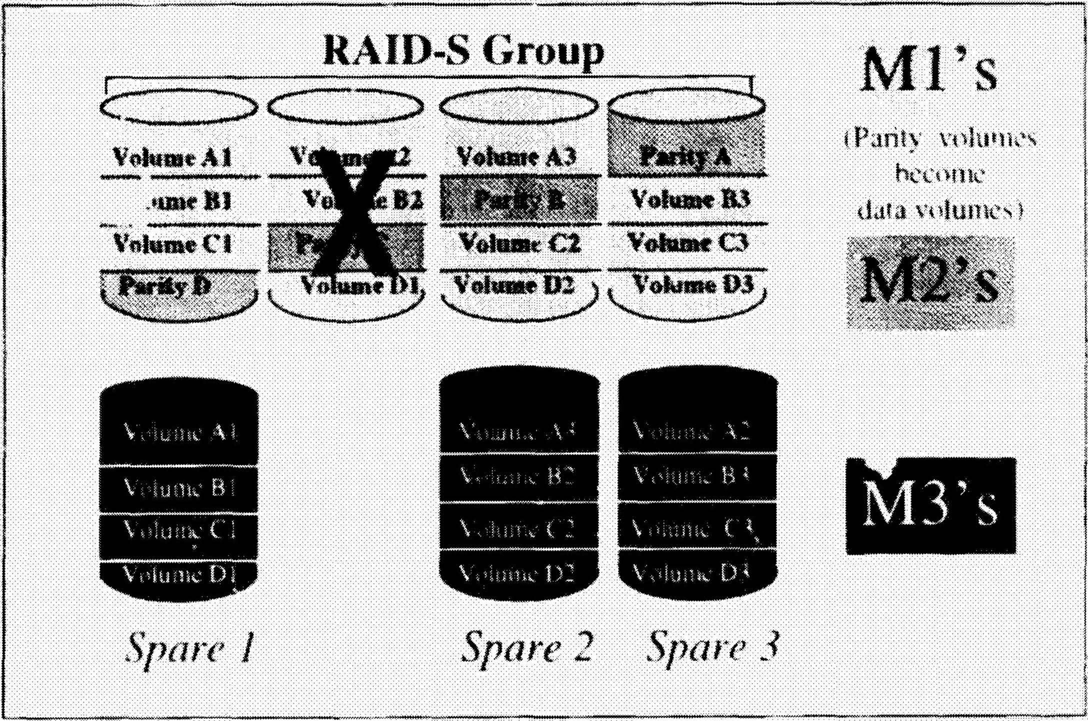


Figure 6: Dynamic Sparing

4.2.1.2.1 Partial Dynamic Sparing

In an effort to provide as much data protection as possible, the dynamic sparing process will also invoke when only one or two dynamic spares are defined or available at the time of the failure. In this case the Symmetrix will regenerate the data volumes from the failed disk and write the data to the first spare disk. Data from one unaffected disk will be copied to the second spare and mirrored relationships will be established. In this way some of the data continues to be protected from a second disk failure via mirroring.

The array will then operate as if in reduced mode. *Any* I/O to the failed volumes, or any *write* I/O to the surviving volumes, will cause a regeneration of data to the parity volume. Note that in this case data is not regenerated to the parity volumes automatically, but rather on demand. Since a full complement of spares was not available (only one or two were) full mirroring protection for the group could not be achieved, and the system will not incur the overhead of completely rebuilding the data volumes onto the parity volumes.

This will reduce the parity rebuild workload when the service action is complete and return the array to normal operation as quickly as possible

When the service action is complete and the new disk is in place, the data volumes are copied onto the replacement disk and a parity rebuild is performed for all parity volumes in the group.

This approach to full and partial dynamic sparing provides several benefits:

- Elapsed time for rebuild is lower since less reconstruction via XOR is required
- Performance during rebuild is improved through the use of mirroring.
- The amount of storage dedicated to the sparing function is less, especially in larger configurations.

4.2.2 Failed Parity Volume

The failure of a parity volume does not place a rank in reduced mode. All I/Os are serviced from the surviving data volumes as normal non-RAID requests, and parity protection is suspended for the rank. The disks then operate as normal non-RAID devices. When the parity volume is replaced, the rebuild function restores the volume as a parity volume and parity protection is resumed for the rank and RAID-S operating mode is restored.

4.2.3 Surviving Members

Read Miss Operations: Read operations to surviving members in a reduced mode rank are equivalent to non-RAID operating mode.

Write Operations: Write operations to a surviving member in a reduced mode rank triggers the regeneration of corresponding tracks for the failed member, and the writing of the regenerated data to the parity drive. Once this is complete, the write I/O is allowed to complete to the surviving member. The reason that the failed member's data is regenerated first is because writing data directly to a surviving member would immediately invalidate the parity data. Rather than allow parity data to be invalid, the Symmetrix replaces it with valid data for the failed member, thus ensuring data integrity in the rank and improving performance both for future requests to the failed member and the resynchronization of the failed member after a service action.

Note: Gradually, the parity volume will take over for the failed data volume and service all I/O intended for the failed volume. All read and write requests to the failed volume, as well as all write requests to the surviving volumes, result in regenerated data being written to the parity volume.

5. Rebuild

When a drive in a RAID-S group is replaced, the rebuild process begins. Rebuild consists of distinct phases. The first phase is the restoration of the data volumes on the affected disk drive. This can occur in one of two ways; either regenerated data from the parity volume is copied to the data volume, or the data is regenerated from the surviving members. The second phase is the rebuilding of the parity volume, and is depicted in figure 7. During parity rebuild, only locations that stored regenerated data during the reduced mode operation are rebuilt. This helps to improve the overall rebuild time for a group.

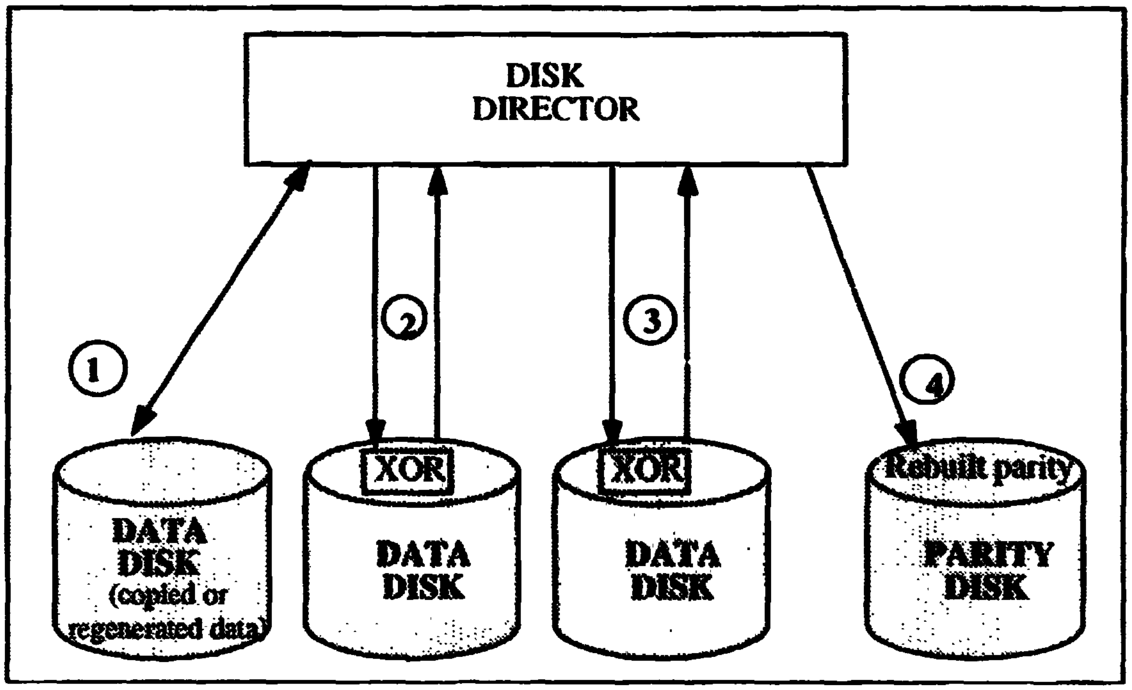


Figure 7: Rebuild Function

1	Rebuild begins with the DD issuing a standard SCSI READ command to the first data volume in the rank and receiving the data back from the drive.
2	The data is sent to the second data volume using an XOR READ command with a bit set to instruct the drive to **not** write the data to the disk, but allowing it to perform the XOR computation with the data on that disk drive. The XOR'd data is sent back to the DD.
3	The DD issues another XOR READ sending the XOR'd data to the third data volume drive in the rank again with the bit set to prevent the data that was sent from being written to the disk. The data is XOR'd with the data on the disk and the result (parity for the rank) is sent back to the DD.
4	The DD issues a standard SCSI WRITE command to write the rebuilt parity to the

	disk drive.

The rebuild process is invoked by a CE or PSE as part of the disk drive replacement procedure. During the rebuild process requests can continue to be serviced by the rank, however, parity protection is not restored to the rank until the rebuild operation is complete. The rebuild process is a background task that is secondary to servicing host I/O requests.

6. RAID-S Performance

6.1 RAID-S Performance Advantages

RAID-S' unique implementation and general Symmetrix architecture together offer significant performance advantages over traditional parity based RAID implementations. These advantages are summarized below:

- **Large Cache**
  Symmetrix large central cache continues to provide customers with very high read hit rates regardless of the RAID protection scheme implemented in the "back-end" of the subsystem. Cache resources are not used to store or compute redundancy data (i.e., parity or mirrored data).

- **100% Fast Write**
  The "cache all" philosophy of the Symmetrix ensures that all writes are fast writes thus ensuring the highest possible "front-end" performance for write requests. Since RAID-S made no changes to the front end of the system, the benefits of this architecture continue to accrue for RAID-S system.

- **Distributed XOR**
  The use of XOR capable disks in RAID-S improves the performance of the Read-Modify-Write sequence for parity generation compared to traditional RAID schemes. By reducing the workload on the controller, back end path contention is also reduced, contributing to faster performance when operating in normal mode.

- **Segregated RAID Groups**
  RAID-S groups are segregated from each other in the back end of the system. Rebuild activity on one group does not impact the performance of the remainder of the groups in the system.

- **Tuning**
  RAID-S keeps logical volumes intact by exploiting the Symmetrix Hyper-Volume Extension feature to map logical volumes to one and only one member disk. As a result, traditional performance tuning techniques that have been employed by storage administrators for decades can be used to tune RAID-S systems.

6.2 Performance Considerations

As with all Symmetrix ICDAs cache size and cache friendliness of the workload have a major impact on the performance delivered by the Symmetrix. Standard cache sizes for RAID-S Symmetrix systems have been adjusted upward to ensure a consistent level of performance when workloads have a heavier write orientation. As is true for all parity based RAID implementations, RAID-S is best suited to workloads whose write content is less than 25%.

The configuration flexibility of Symmetrix is an important feature in ensuring good performance for all workloads. The ability to configure a pool of RAID-1 protected volumes in a Symmetrix that is mostly protected via RAID-S is called "scalable availability", and should be used to support applications with very high write content, such as disk to disk copies and large sequential file loading operations.

Having said that, however, it is important to keep in mind that a cache hit is a cache hit, and in this respect RAID-S Symmetrix performs in the same manner as non-RAID and mirrored Symmetrix. Understanding workload characteristics, and exploiting scalable availability where appropriate, will help ensure successful RAID-S implementations

6.2.1 Normal Mode

For read miss I/Os, RAID-S performance in normal mode is equivalent to non-RAID performance.

During periods of high utilization, I/Os may be impacted and experience a modest increase in response time. It is impossible to specifically quantify the effect since it is a function of read/write ratios, I/O rates, cache size, data blocksize, and duration of the high demand. Like other parity based RAID implementations, RAID-S does exhibit a write penalty, however the basic design of the Symmetrix (i.e. large cache) and the innovative approach taken with RAID-S minimizes the impact compared to traditional RAID implementations. Overall performance of RAID-S will obviously be dependent on the I/O rate and write content of the workload. Higher I/O rates and write content (>25%) may result in longer elapsed time due to the write penalty under these conditions.

6.2.2 Reduced Mode

Reduced mode performance of a RAID-S group is dependent upon several factors including: which volumes in the rank are being accessed, the number of volumes affected by the failure, the layout of the ranks within the Symmetrix, the I/O rate, and the read/write ratio of the workload.

Reads misses and de-stages to the failed members invoke the regeneration process for the first access to each track. De-stages to surviving members also invoke the regeneration process for the failed member (if the corresponding tracks on the failed member have not already been regenerated).

Reads misses to surviving members are treated as normal non-RAID I/Os.

These variables, the types of I/O and which volumes are being accessed, combine to make it difficult to predict the exact performance of a reduced mode RAID-S group.

6.2.3 Rebuild Mode

The performance metrics of interest in rebuild mode are *response time* for host I/Os and the *elapsed time* of the rebuild (i.e., the wall clock time spent returning the array to normal mode). These metrics are affected by the amount of host I/O, the distribution of that I/O between the replacement disk and the other disks in the array, and the read/write ratio of the workload.

Elapsed time is also directly impacted by the size of the disks in the RAID-S group. While this may seem obvious, it is often overlooked, especially when comparing different vendors RAID-5 implementations. RAID-S uses either 4GB or 9GB disks. A rebuild can clearly execute faster on a 4-GB disk since less than half the amount of data is being rebuilt. It is inaccurate to compare the rebuild times of a four disk array utilizing 4-GB drives with a four disk array which uses 9-GB drives.

It is also important to remember that the performance impact of a RAID-S rebuild is isolated to the group undergoing the rebuild. The remainder of the system is essentially unaffected.

6.2.3.1 Symmetrix RAID-S Rebuild performance

"Rebuild" is one of the three modes of operation in a RAID-S protected Symmetrix. (The other two modes are "normal" and "reduced"). A RAID-S group enters rebuild mode when a failed member of the group is replaced with a new disk and that new disk must be populated (rebuilt) with user data and parity re-calculated for the group.

6.2.3.1.1 RAID-S Rebuild Differentiators

RAID-S is implemented on a disk director level and utilizes all four SCSI buses on a disk director (i.e., a RAID-S group cannot span disk directors). **Consequently, the impact of rebuilding a RAID-S group is isolated to the group undergoing the rebuild and does not affect the rest of the subsystem.**

Also, RAID-S is the only parity based RAID system on the market that does not maintain parity when running in reduced mode. Rather than maintain parity when operating with a failed member, RAID-S places regenerated data on the parity volume so that subsequent requests for the same data do not incur the overhead of regeneration. This feature is exploited during rebuild mode since a new disk can be pop ated by copying previously regenerated data, rather than by incurring the overhead of a rebuild. This helps reduce response times for host I/O requests during the rebuild operation.

Further, RAID-S rebuild runs as a lower priority task on the disk director, so host I/O is serviced before rebuild i/O, resulting in lower response times for host I/Os.

[i] For a detailed description of XOR see one of the following:
"A Comparison of RAID-1 and RAID-5" *ESG Marketing Corporate SE Services* [February 13, 1995]
"What is Exclusive OR?" *SalesAdvantage* [February 27, 1995]
"The RAIDBook," *The RAID Advisory Board* [September 1, 1994]
[ii] M2 is the term used to describe the second volume in a mirrored pair.

NEXT
DOCUMENT

Y3226

Large Format Multifunction 2-Terabyte Optical Disk Storage System

David R. Kaiser, Charles F. Brucker, Edward C. Gage,
T.K. Hatwar, George O. Simmons

Eastman Kodak Company
460 Buffalo Road
Rochester, NY 14652-3816
kaiser@kodak.com
Tel: 716-588-5589
Fax: : 716-588-7693

Abstract

The *Kodak Digital Science* OD System 2000E Automated Disk Library (ADL) Base Module and write-once drive are being developed as the next generation commercial product to the currently available System 2000 ADL. Under government sponsorship with the Air Force's Rome Laboratory, Kodak is developing magneto-optic (M-O) subsystems compatible with the *Kodak Digital Science* ODW25 drive architecture, which will result in a multifunction (MF) drive capable of reading and writing 25 gigabyte (GB) WORM media and 15 GB erasable media. In an OD System 2000E ADL configuration with 4 MF drives and 100 total disks with a 50% ratio of WORM and M-O media, 2.0 terabytes (TB) of versatile near line mass storage is available.

Introduction

The architecture of the MF drive is a highly leveraged version of the WORM drive. With the exception of the MF optical head, MF analog head electronics, and bias field magnet the drive hardware is unchanged from the commercial WORM design. The MF analog electronics condition the M-O readback signals such that when they are forwarded to digitizing electronics, they are compatible with WORM signals, thereby prese ving a majority of the hardware architecture.

The MF optical head has a 680 nanometer wavelength laser and 0.55 numerical aperture lens, which provide a 0.7 micron minimum mark size. The signal balancing capabilities in the MF analog electronics reduce effects of power variations and media birefringence. At 12 meters per second using an optimum record power of 5 milliwatts, a narrow band carrier-to-noise-ratio greater than 56 dB has been obtained.

The M-O media is fabricated on the same 356 millimeter diameter aluminum substrate as the commercial WORM media. While this approach required technological advances in MF head electronics because of the polycarbonate coversheet birefringence and the characteristic media noise of the underlayers, the benefits of this approach are numerous. Utilization of existing manufacturing processes and fabrication equipment positively affect quality, process yield, and unit costs for a new media offering. Furthermore, the commercial cartridge hardware provides turn-key mechanical compatibility with existing drive and robotic library designs.

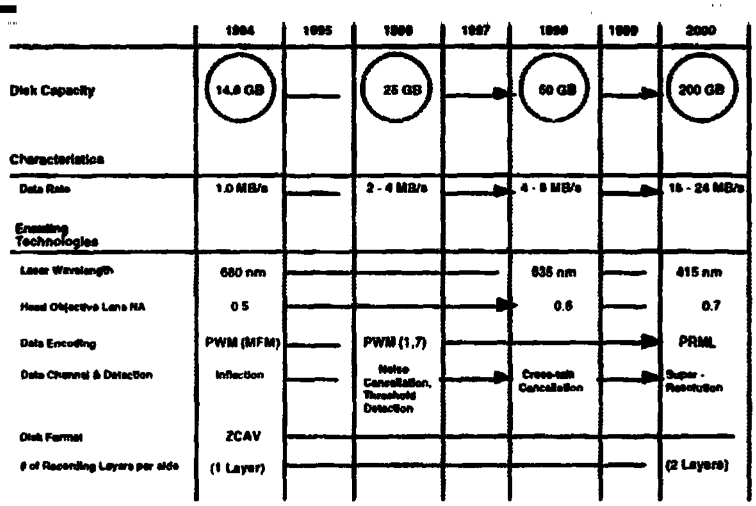
As manufactured, the media is featureless. Tracking pads and sector headers are servo written as part of the manufacturing process. The featureless characteristic allows the erasable media to be re-formatted to accommodate performance improvements in track pitch and capacity as they become available later in the product lifecycle.

The ODW25 drive uses the Intel 960 processor and employs an object orientated design. Therefore, adding the erasable functionality to the WORM baseline is straightforward. The ODW25 drive is field upgradable to MF by means of an optical head change and firmware download through a PCM/CIA card.

Kodak Digital Science ODW25 Optical Drive Architecture

The System 2000E is an evolutionary product based upon the current Kodak Optical Storage Products' large format high capacity automated disk library, the System 2000. The "E" connotation refers to the enhanced capabilities that are provided via the next generation ODW25 optical drive,. The drive is fully backward compatible with Kodak's pre-existing 14" media types, can be readily installed into existing System 2000 libraries and features dual write/read heads, increased data rates, reliability and media capacity. The drive size and weight has been reduced to support a four-drive library configuration providing additional throughput and back-up capability.

The ODW25 drive has been engineered with a platform architecture to facilitate future enhancements and features. Kodak's 14" optical media format was designed to be "dual-head" ready from the onset by formatting opposing spirals on either side of the disk. The platform architecture concept was applied to both the drive and media to support a product family commensurate with the "Technology Roadmap" shown in Figure 1. The strategy behind the platform concept was to develop a "system" design that would provide both a hardware and software base which could be enhanced to support additional features and functions requested by the customer in a timely, cost effective manner.



	1994	1995	1996	1997	1998	1999	2000
Disk Capacity	14.8 GB		25 GB		60 GB		200 GB
Characteristics							
Data Rate	1.0 MB/s		2 - 4 MB/s		4 - 8 MB/s		15 - 24 MB/s
Enabling Technologies							
Laser Wavelength	680 nm				635 nm		415 nm
Head Objective Lens NA	0 5				0.6		0.7
Data Encoding	PWM (MFM)		PWM (1,7)				PRML
Data Channel & Detection	Inflection		Noise Cancellation, Threshold Detection		Cross-talk Cancellation		Super - Resolution
Disk Format	ZCAV						
# of Recording Layers per side	(1 Layer)						(2 Layers)

Figure 1. 14" Optical Storage Technology Roadmap

The ODW25 disk drive features a variety of innovative hardware and software techniques intended to improve reliability and flexibility for future applications. The block diagram shown in Figure 2 illustrates the major subsystems of the drive.

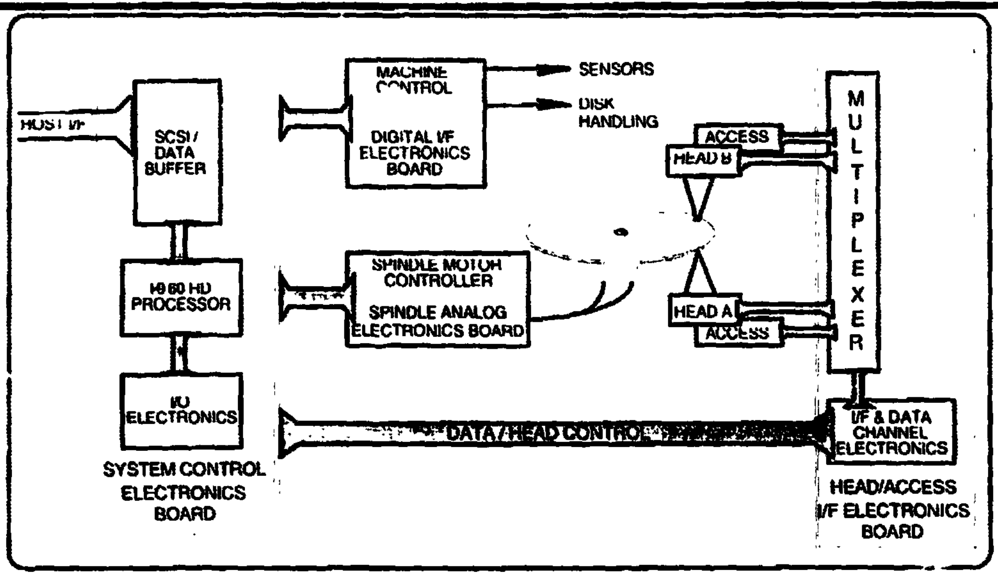


Figure 2. System 2000E ODW25 Arc'hitecture Block Diagram

The System Control Electronics (SCE) circuit board contains the Intel I-960 microprocessor which controls the entire machine, performs error detection and correction on data read back from the media in the drive and handles all communication with the host through a SCSI interface. The Digital Interface Electronics board (DIE) is responsible for all the machine control and I/O not associated with the optical head. These functions include media handling, disk clamping, temperature sensors, monitoring power supply voltages, and more. The Spindle Analog Electronics board (SAE) delivers the required power to the spindle motor and processes the hall sensor data to provide velocity feedback to the servo. The spindle motor is controlled utilizing a pulse width modulated motor driver under software servo control. The Head/Access Interface Board (HIE) processes all signals coming to, or going from, the optical heads. A multiplexer approach switches between the top and bottom heads allowing for near instantaneous access to either side of the disk. Each head is driven by its own carriage motor and motor driver to facilitate access to data on either side of the disk without external robotics. The present HIE contains only one programmable data channel and data channel controller. Future generations of the ODW25 drive will employ simultaneous access to both sides of the media platter.

Sensors integrated into the media handling robotics determine the media type upon insertion into the drive. The Intel I-960 microprocessor then "programs" the gate arrays that comprise the data channel into the proper format for that media capacity. Therefore, future media upgrades and capacity increases can be accommodated with existing hardware. Also, the data channel is fully backward compatible with previous media types.

Control of the various servo subsystems required by the optical drive to maintain media velocity, access position, focus, tracking and laser power is critical to obtain performance expectations. The machine servo control systems must meet product specifications over a wide operating range. The ODW25 drive has utilized a digital servo system, which is controlled by the I-960 processor. These software servos provide both effective machine control and flexibility. The microprocessor samples the servo error signals from the optical head and can "tune" itself to provide optimum performance, something that cannot be accomplished with conventional analog implementations. Modifications to the optical head and head electronics required for future performance upgrades may be accommodated by "reprogramming" the servo functions without changing hardware in the drive. The software servo controls have improved diagnostic capability and reliability via the reduction in the number of electronic components required to operate the optical head.

The Error Detection and Correction (EDAC) algorithm employed by the drive is also coded in software. This concept allows real-time access to byte error rate measurement, improves reliability and lowers cost by eliminating the need for expensive ASIC's and logic arrays. The software EDAC subsystem can be easily modified for future formats as EDAC strategies change as a result of increased packing densities.

The ODW25 drive platform was conceived to provide optimum performance to the customer and maximum flexibility to allow for future upgrades without complete hardware redesign. The platform concept will protect the customer's investment in hardware and ease the integration of future features. The current drive/media system will operate at a write/read data rate of 1.5 to 2.5 megabytes per second per optical head depending upon the media type (capacity). The media is manufactured with a zoned constant angular velocity (ZCAV) format, which provides the most effective compromise between access time and media capacity. The ODW25 drive will utilize all previous media types manufactured by Kodak, 6.8 GB (read only), 10.2 GB, and 14.8 GB, as well as the upcoming 25 GB platters and beyond with additional firmware upgrades. With future performance enhancement in mind, each drive was equipped with independent access systems for each side of the disk. This will facilitate future enhancements to enable simultaneous access to both sides of the disk which will, in effect, double both the write and readback data rates without substantial hardware modifications to the drive. The ODW25 drive with the 2000E automated library will propel the user and the optical storage technology into the 21st century.

Rome Laboratory Erasable Optical Program

Program Overview

The objective of the Rome Laboratory Contract (# F30602-94-C-0047) is to develop erasable optical recording hardware and media subsystems for integration into Kodak's commercial large format drive and library system. The system portrayed in Figure 3 will be delivered to the Air Force and integrated with other storage devices (magnetic disk and magnetic tape) as part of a hierarchical storage management (HSM) system. Large file size intelligence data processing and mission planning operations will be demonstrated using the HSM solution.

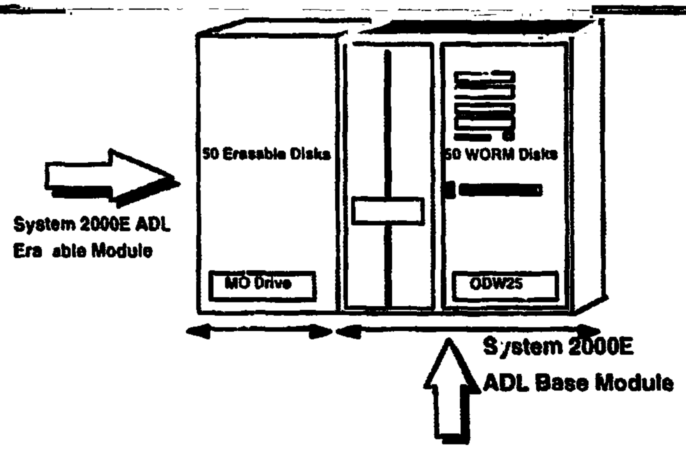


Figure 3: Rome Laboratory System Configuration

The approach employed in the program is to design the erasable subsystems utilizing and/or leveraging the commercial write-once design such that an offering of a commercial erasable drive in the future will require a minimum level of engineering work. Thus, the engineering task focus areas under development including: (1) the optical head; (2) analog conditioning/processing electronics; (3) servo written media format; and (4) high-level SCSI interface command and control software all have significant linkages to the commercial product family. Low-level servo control for laser writing and reading, focus, and tracking are aimed specifically at the MF head and erasable media.

Expanded detail of the technology development work and results obtained to date are contained in the following sections.

Magneto-Optical Media

Here we describe the overall disk structure, characterization of the individual layers, and optimization of the optical stack.

Disk Structure

A new simple trilayer disk structure was used. The disk structure, Al Substrate/Ti reflecting layer/MO layer/AlN antireflection layer, eliminates the second dielectric in the conventional quadrilayer structure, while essentially maintaining its performance. Also, it eases some of the tight manufacturing tolerance limits involved in the quadrilayer structure. Ti metal layer can be deposited more easily and at a significantly higher rate than its dielectric counterparts, e.g., AlN or Si_3N_4. Figure 4 contains a side view detail of the MO disk structure.

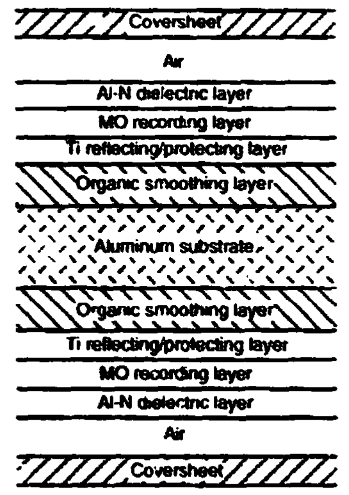


Figure 4. MO Disk Structure Side View

Magneto-optic media was fabricated using a modified Balzers LLS-801 sputter deposition system. The sputter deposition is carried out using three cathodes for depositing a Ti reflector layer, a TbFeCo-based MO layer, and the AlN dielectric layer. During deposition, the substrate is rotated around an axis perpendicular to the sputtering cathode using a turn table affixed to the indexing drum. In this way, all three layers are deposited in sequence with no vacuum break. Subsequently, a protective polycarbonate coversheet is attached and the disk is cartridged identically to the ODW25 product.

MO Layer Characterization

Recording Layer. The recording layer composition, thickness, and deposition conditions were chosen to provide the optimal combination of signal quality, recording power, and environmental stability. The sputtering pressure and film composition were adjusted for a coercivity less than about 10 kOe to enable static room temperature disk erasure using a large area electromagnet; this is a much faster method of initialization compared to dynamic erasure using a focused optical stylus. The circumferential variation in recording layer properties was negligible due to the rotating substrate motion, and the radial variation in thickness was held within ±5% using a specially designed mask. Additions of small amounts of Zr and Pd, have been shown to enhance the intrinsic environmental stability and writing sensitivity of the MO layer.

Dielectric Layer. An AlN dielectric layer was used to optimize the Kerr rotation and reflectivity of the optical stack and, importantly, to provide corrosion protection for the MO layer. It was deposited by DC reactive sputtering of an Al target in an Ar and N_2 atmosphere. The reactive AlN sputtering process involves feedback control of the N_2 flow to maintain constant current at constant pressure. The AlN mechanical and optical properties, as well as thickness uniformity, are critically important for the performance of the disk. Preparation of low stress and crack-free AlN layers is essential for providing long-term corrosion protection of the oxidation susceptible MO layer. AlN films with optiimum properties were obtained by controlling the sputtering power, Ar:N_2 pressure ratio and total sputtering pressure. A radial thickness variation of less than ±5% was obtained. The measured refractive index at 680 nm for AlN is n + ik = 2.06 + i0.01. The low coefficient of absorption k = 0.01 is desirable for efficient optical performance.

Reflector Layer. Ti metal was used as a reflecting layer. Ti metal has low thermal conductivity so in addition it acts as a thermal barrier between the MO layer and the surface smoothed aluminum substrate, thus improving the writing sensitivity of the disk. The Ti layer also provides corrosion protection for the MO media from the organic surface smoothing material. Its thickness uniformity was within ±5%, similar to the MO and dielectric layers. An additional beneficial effect of the Ti underlayer was to enhance the coercivity and squareness of the Kerr hysteresis loop, advantageous for low disk recording noise.

Optimization of Optical Stack. The multilayer stack was designed to obtain adequate figure-of-merit (reflectivity times Kerr rotation) subject to practical constraints on reflectivity and corrosion protection. Several small coupons were made with varying thickness of AlN MO, and Ti layers. Figure 5 show variations of reflectivity R, Kerr rotation θ_k, and figure-of-merit $R\theta_k$, plotted as a function of AlN layer thickness. The optimal combination of figure-of-merit, reflectivity, and passivation was obtained for 50 nm Ti / 45 nm MO / 80 nm AlN. Several full structure disks were fabricated. A lower thickness for the MO and Ti layers was found to give higher writing sensitivity if desired. Also, it was found that the CNR performance was quite insensitive to AlN thickness, demonstrating the robustness of the optical stack design.

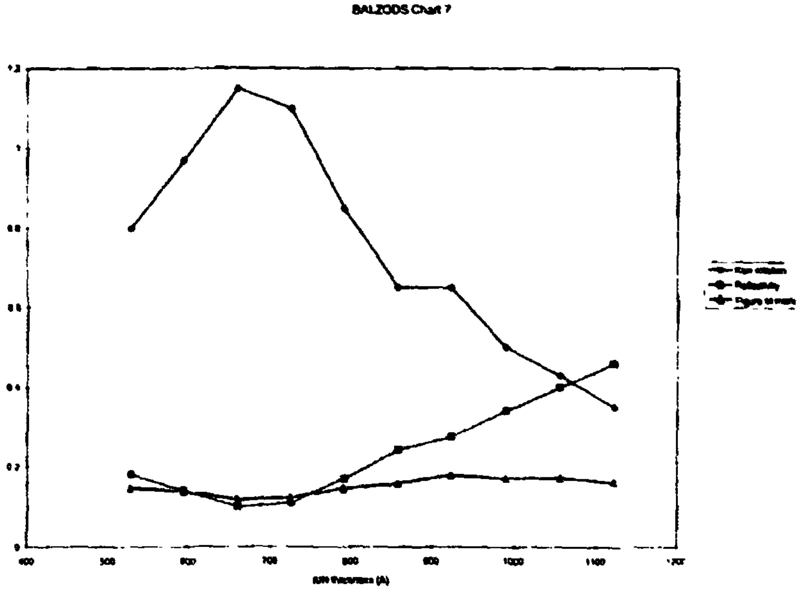


Figure 5. Dependence of reflectivity, Kerr rotation, and figure-of-merit on AlN layer thickness.

Media Keeping

A life test program was designed and carried out to characterize the media shelf life for two recording layer compositions, TbFeCo and TbFeCoZrPd. The fabrication equipment, process, and optical stack structure were identical to those intended for final disk production. Results to date, based on static measurements of coupon samples, indicate exceptional environmental stability for both compositions. In particular, no change in Kerr rotation and reflectivity have been detected after exposure to 70°C/85% RH, 90°C/17% RH, or 32°C/90% RH for six weeks.

Multifunction Optical Head and Analog Electronics

A schematic of the multifunction optical head is shown in Figure 6. The head is leveraged from the current System 2000 WORM optical head and its properties are summarized in Table 1.

The 30 mW SDL laser diode has undergone extensive testing and has been shown to be extremely reliable with very low relative intensity noise. The laser is collimated with 7.5 mm focal length, precision glass molded (Kodak A375) lens. The optical stack uses the same glass types as our commercial product to provide achromatic beam expansion. The coating on the partial PBS was redesigned to maximize the MO data signal and provide an acceptable head efficiency as described in Table 1. A turning prism reflects the beam up to the 0.55 numerical aperture (NA) molded glass objective lens. A 1.1 mm coverplate and the 90 µm coversheet for both the MO and WORM media packages compensates for the lens design substrate thickness of 1.2 mm.

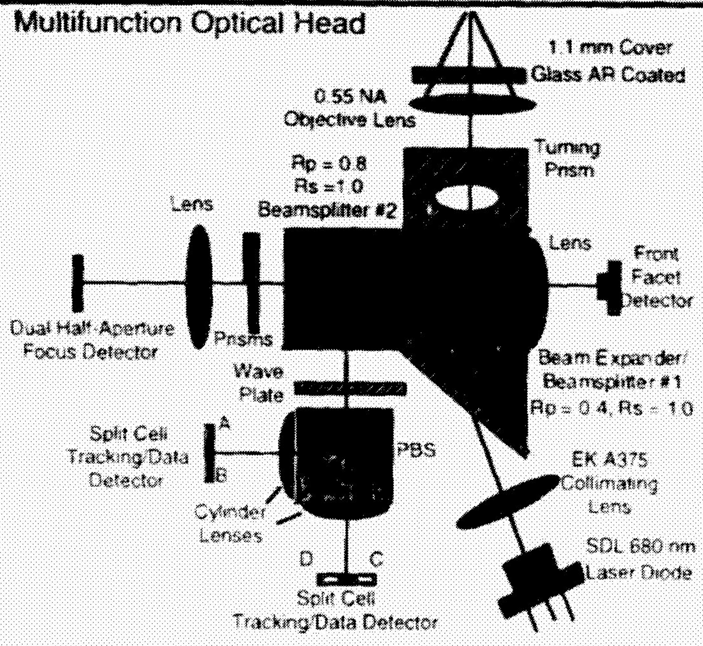


Figure 6. Multifunction 680 nm, 0.55 NA Optical Head

Table 1 Properties of 680-mm Multifunction Optical Head

Media Type	14" MO or WORM
Substrate	90 µm coversheet
Wavelength	680 nm
Spot Size FWHM	0.70 µm
Numerical Aperture	0.55
Head Efficiency	30%
Power at Disk (Maximum):	8 mW
Focus Method	Dual Half Aperture
Tracking Method	Full Aperture Push-Pull

The return path is designed to maximize the data and tracking detection signal-to-noise ratio. The return beam is reflected by the partial polarization beamsplitter #1. The dual half aperture focus detector receives ≈ 20% of the p-polarization component of the return beam. The reflected light from beamsplitter # 2 is directed though a waveplate that corrects for media and head phase shifts and results in approximately equal intensity from the two beams from the polarization beamsplitter (analyzer). The two beams are brought to line foci (elongated in the cross track direction) on a pair of bi-cell detectors that provide signals A, B, C, and D for tracking error and data detection. These signals are processed by the multifunction data/tracking electronics as shown in Figure 7. The push pull tracking signals are given by:

$$\text{WORM TES} = (A + C) - (B + D) \qquad (1)$$

$$\text{MO TES} = (A + D) - (B + C) \qquad (2)$$

where the tracking error signal is sampled from the diffraction effects over servo written tracking pads (long data marks). The data signals are given by:

$$\text{WORM DATA} = (A + B) + (C + D) \qquad (3)$$

$$\text{MO DATA} = (A + B) - (C + D) \qquad (4)$$

The signal balancer electronics utilize variable gain amplifiers tc 'nimize the effects of birefringence and laser power fluctuations on the data signals R! .nd RF B before the final sum (WORM) and difference (MO) are generated according ; equations 3 and 4. This additional step is required with a bi-refringent coversheet in a multifunction system.

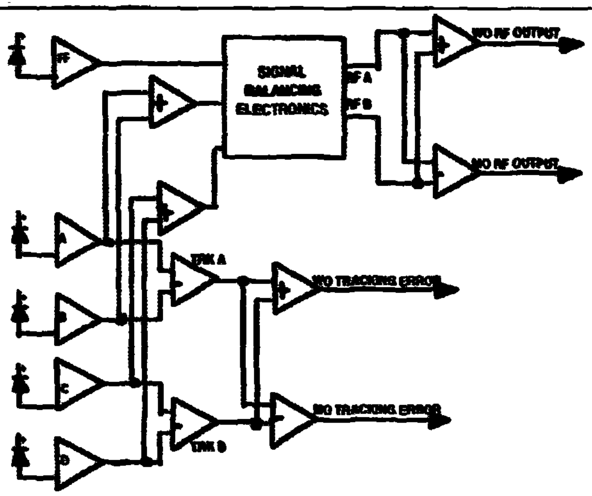


Figure 7. Schematic of the multifunction data/tracking detection electronics

Dynamic Testing

The main thrust of the dynamic testing is to ensure that the WORM performance is similar to the production WORM system and that the MO system exceeds the contract requirements for data integrity, capacity (> 10 Gigabyte/Disk), and data rate (>1 Megabytes/sec). The fundamental performance of the system with the two media types is illustrated in Figure 8. The readout spectrum of the WORM system shows a carrier-to-noise ratio in a 30 KHz bandwidth (CNR) of 57.0 dB. This was recorded at the second harmonic minimum, with an optimum recording power (ORP) of 5.1 mW. By comparison, the MO system shows a CNR of 56.5 dB at an ORP of 5.0 mW. The WORM system is dominated by media noise at low frequencies and laser noise at higher frequencies, while the MO system is dominated by electronic and shot noise. Thus the MO has a whiter noise signature and a lower integrated signal-to-noise ratio. The similarity of the ORP and CNR for WORM and MO is important to a multifunction system design.

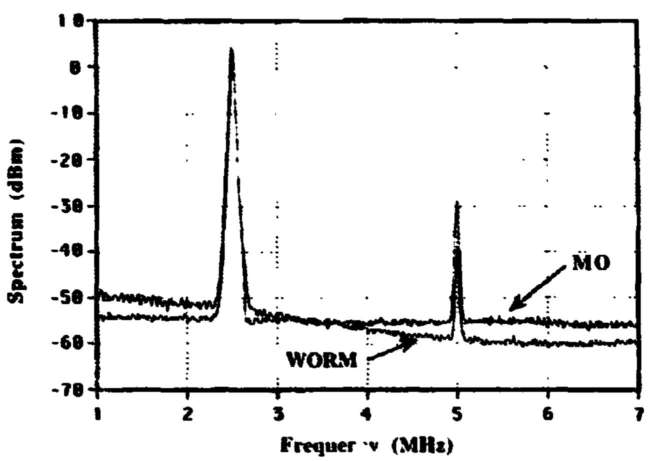


Figure 8. The readout spectrum is shown for WORM and MO readout for a 2.5 MHz tone at a media velocity of 12 m/s. The recording power is 5.1 mW for WORM and 5.0 mW for MO. The read powers are 1.0 and 1.5 mW for WORM and MO readout respectively.

This system has also been integrated with the System 2000E read channel. The figure of merit percent phase margin (% PM) indicates the amount of the bit window remaining after noise and mark length errors are considered for a raw bit error rate of 10^{-6}. Commercial goals are typically 20-50% PM depending on the systems error budget. For the multifunction drive with a worst case pattern at the conditions of 12 m/sec, MFM encoding, and a raw data rate of 10 Megabits/sec, the WORM system has a % PM of 70% consistent with our future 25 Gigabyte/Disk commercial product and the MO system has a % PM of 45%, which will allow the contract specifications to be met.

Ultrahigh Capacity Optical Disk (UCOD) Program

The National Storage Industry Consortium (NSIC) is leading an Advanced Technology R&D project with the Department of Commerce for the development of an optical data storage system that will place U.S. technology at the forefront of commercial data storage markets throughout the remainder of this century and well into the next century. The program teams Eastman Kodak Company, a leading supplier of high-end optical data storage libraries, SDL, Inc., the world-wide leader in high-power laser diode manufacturing, and Carnegie-Mellon University, a leading research facility in optical storage in a highly focused program to produce an optical data storage system with the following attributes:

- 1 Terabyte storage capacity, a 40 x increase over current technology
- 30 Megabyte/sec data transfer rate, a 10 x increase

The technology developed will be rapidly incorporated into products at both Kodak and SDL throughout the program, such that the program will serve to strengthen and solidify the technical position of many U.S. industries, including high-definition television (HDTV), medical and library data storage systems, biotechnology, and visible laser diode systems.

A four-year research and development program (Fiscal Year 1996-1999) with four major technologies is underway. The four technology areas are: (1) advanced laser sources, (2) multilayer media technology, (3) advanced channel coding techniques, and (4) high numerical aperture optics development. The development is being pursued in three major phases. An assessment phase will concentrate on gathering data and building integrated models. The experimental phase will include ta-geted work on the four technology elements discussed above using refined goals from the assessment. The final stage is the design, fabrication, and testing of the a prototype system. The technical challenges in this development are listed in the Table 2.

Table 2: Research Task/Barrier/Approach Matrix

Task	Barrier	Primary Approach
• Multilayer media	Ultra-thin film performance	Self passivation, new materials
• Blue Light source	Crystal damage	Fabrication, Doubling Method
• High NA Objective	Aberration Tolerances	Molded precision glass
• High Density Code	SNR Requirements	PRML

While the Rome Laboratory (RL) program will provide the first Beta version of Magneto-Optic recording subsystems integrated in a commercial drive platform, the UCOD program will leverage the RL program and advance state-of-the-art in write-once and erasable optical recording.

Conclusions

A key component of the "platform approach" of the ODW25 was to provide the capability to implement future enhancements with reduced resources and cycle times. The Rome Laboratory erasable optical project has utilized the platform effectively. The direct compatibility of the multifunction optical head, media substrate and cartridge, and implementation of featureless servo written formatted media will provide the capability to commercialize a multifunction drive in the future. The UCOD program will develop new technology which will continue to efficiently add significant performance improvements to the ODW25 platform.

References

1. T.W. McDaniel and F.O. Sequeda, Appl. Phys. Commun. . 1, 427 (1992).

2. T.K. Hatwar, J. Appl. Phys. 70, 6335 (1991).

3. E. C. Gage, S. Beckens, P. Cronkite, S. Dohmeier, D. Kay, M. Meichle, and R. Metzger, "Low Noise, High Reliability 680 nm Optical Head Enables Robust 14.9 Gigabyte/Disk Product," SPIE Proc. Vol. 2514, Optical Data Storage '95, eds. G. R. Knight, H. Ooki, and Y. S. Tyan, p. 129 (1995).

4. D. B. Kay, S. B. Chase, E. C. Gage, and B. D. Silverstein, "Write Noise from Optical Heads with Non-Achromatic Beam Expansion Prisms," SPIE Proc. Vol. 1499, Optical Data Storage '91, eds. J. J. Burke, T. A. Shull, and N. Imamura, p. 281 (1991).

5. E. C. Gage and B. J. Bartholomeusz, "Directional Asymmetries due to Write-Laser Mode Hopping during Optical Recording," J. Appl. Phys. 69, 569 (1991).

REPORT DOCUMENTATION PAGE	*Form Approved* *OMB No. 0704-0188*

Public reporting burden for this collection of information is estimated to average 1 hour per response including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information Send comments regarding this burden estimate or any other aspect of this collection of information including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway Suite 1204, Arlington VA 22202-4302 and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	September 1996	Conference Publication

4. TITLE AND SUBTITLE

Fifth Goddard Conference on Mass Storage Systems and Technologies - Volume II

6. FUNDING NUMBERS

Code 505

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

96B00117

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS (ES)

National Aeronautics and Space Administration
Washington, DC 20546-0001

10. SPONSORING / MONITORING AGENCY REPORT NUMBER

NASA CP-3340, Vol. II

11. SUPPLEMENTARY NOTES

Kobler: Goddard Space Flight Center, Greenbelt, Maryland;
Hariharan. Systems Engineering and Security, Inc., Greenbelt, Maryland

12a. DISTRIBUTION / AVAILABILITY STATEMENT

Unclassified - Unlimited
Subject Category 82

Availability: NASA CASI (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 Fifth Goddard Conference on Mass Storage Systems and Technologies held September 17 - 19, 1996, at the University of Maryland, University Conference Center in College Park, Maryland As one of an ongoing series, this conference continues to serve as a unique medium for the exchange of information on topics relating to the ingestion and management of substantial amounts of data and the attendant problems involved This year's discussion topics include storage architecture, database management, data distribution, file system performance and modeling, and optical recording technology. There will also be a paper on Application Programming Interfaces (API) for a Physical Volume Repository (PVR) defined in Version 5 of the Institute of Electrical and Electronics Engineers (IEEE) Reference Model (RM) In addition, there are papers on specific archives and storage products

14. SUBJECT TERMS

Magnetic tape, magnetic disk, optical disk, mass storage, software storage, digital recording, data compression, storage architecture, optical recording, database management.

15. NUMBER OF PAGES

348

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

National Aeronautics and
Space Administration

Goddard Space Flight Center
Greenbelt, Maryland 20771

Official Business
Penalty for Private Use, $300