

NASA Conference Publication 3340 Vol. I

033637

10P.

Fifth NASA Goddard Conference on Mass Storage Systems and Technologies

Volume I

*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. I

Fifth NASA Goddard Conference on Mass Storage Systems and Technologies

Volume I

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 Oceanic and Atmospheric 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 programming 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 are 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. Optical 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, Department of Defense
Bill Callicott, consultant
Sam Coleman, Lawrence Livermore National Laboratory
Robert Creecy, Census Bureau
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

Derived Virtual Devices: A Secure Distributed File System Mechanism, <i>Rodney Van Meter, Steve Hotz, and Gregory Finn, University of Southern California</i>	1 -1
Cooperative, High-Performance Storage in the Accelerated Strategic Computing Initiative, <i>Mark Gary, Barry Howard, Steve Louis, Kim Minuzzo, and Mark Seager, Lawrence Livermore National Laboratory</i>	21 -2
An MPI-IO Interface to HPSS, <i>Terry Jones, Richard Mark, Jeanne Martin, John May, Elsie Pierce, and Linda Stanberry, Lawrence Livermore National Laboratory</i>	37 -3
A Proposed Application Programming Interface for a Physical Volume Repository, <i>Merritt Jones, MITRE Corporation; Joel Williams, Systems Engineering and Security; and Richard Wrenn, Digital Equipment Corporation</i>	51 4
A Global Distributed Storage Architecture, <i>Dr. Nemo M. Lionikis and Michael F. Shields, Department of Defense</i>	67 -5
Petabyte Class Storage at Jefferson Lab (CEBAF), <i>Rita Chambers and Mark Davis, Jefferson Lab Computer Center</i>	77 -6
DKRZ Workload Analysis, <i>Hartmut Fichtel, Deutsches Klimarechenzentrum GmbH</i>	91 -7
A New Generic Indexing Technology, <i>Michael Freeston, University of California</i>	99 -8
Advanced Optical Disk Storage Technology, <i>Fred N. Haritatos, Rome Laboratory</i>	121 -9
Is the Bang Worth the Buck? A RAID Performance Study, <i>Susan E. Hauser, Lewis E. Berman, and George R. Thoma, National Library of Medicine</i>	131 -10
The Medium is NOT the Message OR Indefinitely Long-Term File Storage at Leeds University, <i>David Holdsworth, Leeds University</i>	141 -11
Analysis of the Access Patterns at GSFC Distributed Active Archive Center, <i>Theodore Johnson, University of Florida; Jean-Jacques Bedet, Hughes STX Corporatio</i>	153 -12
A Media Maniac's Guide to Removable Mass Storage Media, <i>Linda S. Kempster, IIT Research Institute</i>	179 -13

The Cornerstone of Data Warehousing for Government Applications, <i>Doug Kenbeek and Jack Rothschild, EMC Corporation</i>	191	-14
Incorporating Oracle On-Line Space Management with Long-Term Archival Technology, <i>Steven M. Moran and Victor J. Zak, Oracle Corporation</i>	209	-15
Design and Implementation of Scalable Tape Archiver, <i>Toshihiro Nemoto, Masaru Kitsuregawa, and Mikio Takagi, University of Tokyo</i>	229	-16
Long-Term Archiving and Data Access: Modelling and Standardization, <i>Claude Huc, Thierry Levoir, and Michel Nonon-Latapie, French Space Agency</i>	239	-17
Automated Clustering-Based Workload Characterization, <i>Odysseas I. Pentakalos, NASA Goddard Space Flight Center; Daniel A. Menasce, George Mason University; Yelena Yesha, University of Maryland</i>	253	-18
Digital Optical Tape: Technology and Standardization Issues, <i>Fernando L. Podio, National Institute of Standards and Technology</i>	265	-19
Storage and Network Bandwidth Requirements Through the Year 2000 for the NASA Center for Computational Sciences, <i>Ellen Salmon, NASA Goddard Space Flight Center</i>	273	-20
NASDA's Earth Observation Satellite Data Archive Policy for the Earth Observation Data and Information System (EOIS), <i>Shin-ichi Sobue, Osamu Ochiai, and Fumiyoshi Yoshida, ASDA EOC</i>	287	-21
Progress in Defining a Standard for File-Level Metadata, <i>Joel Williams, Systems Engineering and Security; Ben Kobler, NASA Goddard Space Flight Center</i>	291	-22

Table of Contents

Volume II

Development of Secondary Archive System at Goddard Space Flight Center Version O Distributed Active Archive Center, <i>Mark Sherman, John Kodis, Jean-Jacques Bedet, and Chris Walker, Hughes STX; Joanne Woytek and Chris Lynnes, NASA Goddard Space Flight Center</i>	301
The Global File System, <i>Steven R. Soltis, Thomas M. Ruwart, and Matthew T. O'Keefe, University of Minnesota</i>	319
Distributed Large Data-Object Environments: End-to-End Performance Analysis of High Speed Distributed Storage Systems in Wide Area ATM Networks, <i>William Johnston, Brian Tierney, Jason Lee, Gary Hoo, and Mary Thompson, Lawrence Berkeley National Laboratory</i>	343
Understanding Customer Dissatisfaction with Underutilized Distributed File Servers, <i>Erik Riedel and Garth Gibson, Carnegie Mellon University</i>	371
Mass Storage and Retrieval at Rome Laboratory, <i>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</i>	389
Durable High-Density Data Storage, <i>Bruce C. Lamartine and Roger A. Stutz, Los Alamos National Laboratory</i>	409
A Note on Interfacing Object Warehouses and Mass Storage Systems for Data Mining Applications, <i>Robert L. Grossman, Magnify, Inc., Oak Park, IL; University of Illinois at Chicago, Dave Northcutt, Magnify, Inc.</i>	421
Towards the Interoperability of Web, Database, and Mass Storage Technologies for Petabyte Archives, <i>Reagan Moore, Richard Marciano, Michael Wan, Tom Sherwin, Richard Frost San Diego Supercomputer Center</i>	431
The Challenges Facing Science Data Archiving on Current Mass Storage Systems, <i>Bernard Peavey and Jeanne Behnke, Goddard Space Flight Center</i>	449
Processing Satellite Images on Tertiary Storage: A Study of the Impact of Tile Size on Performance, <i>JieBing Yu and David J. Dewitt, University of Wisconsin-Madison</i>	460

Evolving Requirements for Magnetic Tape Data Storage Systems, <i>John J. Gniewek, IBM Corporation</i>	477
Optimizing Input/Output Using Adaptive File System Policies, <i>Tara M. Madhyasta, Christopher L. Elford, and Daniel A. Reed, University of Illinois</i>	493
Towards Scalable Benchmarks for Mass Storage Systems, <i>Ethan L. Miller, University of Maryland Baltimore County</i>	515
Queuing Models of Tertiary Storage, <i>Theodore Johnson, University of Florida</i>	529
I/O-Efficient Scientific Computation Using TPIE, <i>Darren Erik Vengroff, University of Delaware; Jeffrey Scott Vitter, Duke University</i>	553
Progress Toward Demonstrating a High Performance Optical Tape Recording Technology, <i>W. S. Oakley, LOTS Technology, Inc</i>	571
RAID Disk Arrays for High Bandwidth Applications, <i>Bill Moren, Ciprico, Inc</i>	583
RAID Unbound: Storage Fault Tolerance in a Distributed Environment, <i>Brian Ritchie, Alphatronix, Incorporated</i>	589
SAM-FS-- LSC's New Solaris-Based Storage Management Product, <i>Kent Angell, LSC, Inc</i>	593
Use of HSM with Relational Databases, <i>Randall Breeden, John Burgess, and Dan Higdon, FileTek, Incorporated</i>	601
RAID-S Technical 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, <i>Brett Quinn, EMC Corporation</i>	605
Large Format Multifunction 2-Terabyte Optical Disk Storage System, <i>David R. Kaiser, Charles F. Brucker, Edward C. Gage, T.K. Hatwar, George O. Simmons, Eastman Kodak</i>	627

Derived Virtual Devices: A Secure Distributed File System Mechanism¹

Rodney Van Meter, Steve Hotz, Gregory Finn
Information Sciences Institute
University of Southern California
Marina del Rey, CA 90292
{rdv,hotz,finn}@isi.edu
310-822-1511

Abstract

This paper presents the design of *derived virtual devices* (DVDs). DVDs are the mechanism used by the Netstation Project to provide secure shared access to network-attached peripherals distributed in an untrusted network environment. DVDs improve Input/Output efficiency by allowing user processes to perform I/O operations directly from devices without intermediate transfer through the controlling operating system kernel. The security enforced at the device through the DVD mechanism includes resource boundary checking, user authentication, and restricted operations, e.g., read-only access. To illustrate the application of DVDs, we present the interactions between a network-attached disk and a file system designed to exploit the DVD abstraction. We further discuss third-party transfer as a mechanism intended to provide for efficient data transfer in a typical NAP environment. We show how DVDs facilitate third-party transfer, and provide the security required in a more open network environment.

1. Introduction

A network attached peripheral (NAP) is a device that communicates with the external world via a network interface, rather than a bus. System buses limit the sharing of devices and do not scale well in bandwidth, distance or number of devices. Communication via a local area network (LAN) provides flexibility in system design and avoids the problems of shared-bus communication, while allowing us to exploit the ever-increasing aggregate bandwidth provided by high-speed networks. These advantages are changing the way computer system architectures are defined [1], and we see NAPs becoming a significant component of new storage systems [2,3] and new multimedia architectures [4,5].

However, components of a system built around a LAN cannot depend on the tight coupling and simplifying assumptions provided by a bus-based architecture. The

¹ This research was sponsored by the Advanced Research Projects Agency under Contract No. DABT63-93-C-0062. Views and conclusions contained in this report are the authors' and should not be interpreted as representing the official opinion or policies, either expressed or implied, of ARPA, the U.S. Government, or any person or agency connected with them.

boundary between the "inside" and "outside" of a system grows fuzzier, and the inherent level of trust that can be assumed among components of the system must decrease. Many current NAP system designs simply treat the LAN like a different type of bus, but do not address this added "open-ness" dimension and the consequent security issues. This may prove problematic unless the type and sources of network traffic are limited to prevent misuse of the NAPs.

The Netstation Project is explicitly addressing the problems inherent in using NAPs in an open network environment. A Netstation is a heterogeneous distributed system comprised of NAPs brought together as a single system operating across a network. One of our primary goals is to support multiple Netstations made up of components connected via a single, shared LAN. The requirement to allow individual devices to be shared by multiple Netstation systems results in additional complexity. Moreover, we have chosen to support IP connectivity of these devices, to allow Netstations to be configured across LAN boundaries. Each of these goals introduces issues of safety and security.

This paper presents derived virtual devices (DVDs) as the mechanism used by the Netstation Project to provide secure shared access to network-attached peripherals distributed in an untrusted network environment. The security enforced at the device through the DVD mechanism includes resource boundary checking, user authentication, and operational restriction, e.g. read-only access. Yet, DVDs enable efficiency by allowing user processes to perform I/O transfers directly from devices without intermediate transfer of data through the controlling operating system kernel. DVDs also support nested or recursive granting of access to the device, allowing file and window systems to run recursively.

The remainder of this paper is organized as follows. Section 2 presents an overview of the Netstation Project and its architecture to provide context for the discussion of DVDs. Section 3 describes the DVD abstraction, command interfaces for management and access, and security mechanisms in detail. In section 4, we illustrate the use of DVDs by presenting the interactions between a network-attached disk and a file system designed to exploit the DVD abstraction. Section 5 shows how DVDs can facilitate third-party transfer for efficient data transfer between NAPs. Sections 7 through 9 discuss related work, the current state of our implementation, and our conclusions.

2. Netstation Environment

The Netstation Project [6] evolved from research on the Atomic LAN [7], a 640Mbps point-to-point switched LAN developed at ISI from parallel computing chips designed by Chuck Seitz and his group at Caltech². The idea behind Netstation is to substitute a gigabit network in place of a workstation bus, similar to the efforts of the DAN [4] and ViewStation [5] groups. Devices such as disks, cameras, displays, and low-bandwidth input concentrators are connected to processing nodes via the LAN. Figure 1 shows a typical hardware configuration.

² The Atomic LAN has become a commercial product of Myricom known as Myrinet.

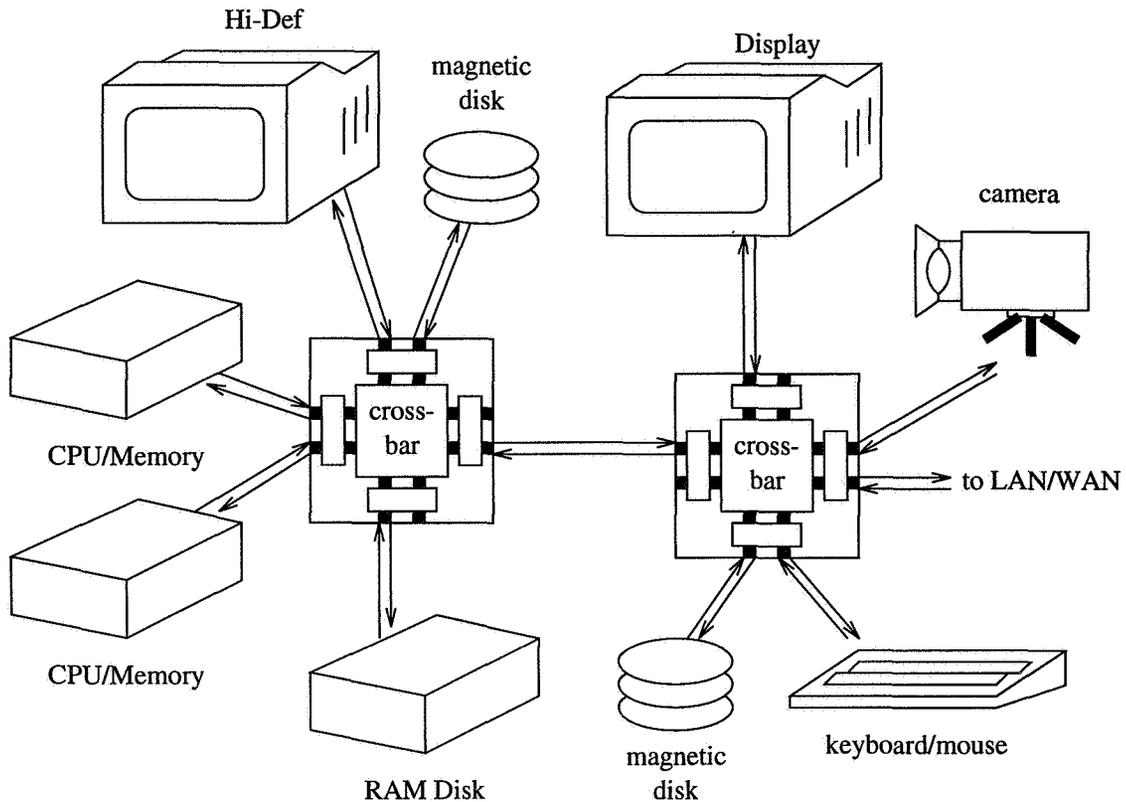


Figure 1: Netstation Architecture

The primary advantage of a Netstation is allowing high-bandwidth devices to communicate directly, alleviating the shared-bus bottleneck. An example application would be the transfer of video data directly from an incoming network port to a network-attached display device, without passing through the main processor or the cpu/memory bus. A second potential advantage is the flexibility afforded by dynamically configuring workstation components from a shared pool of resources.

The Netstation architecture includes (a) two related device abstractions (Network Virtual Devices and Derived Virtual Devices) which provide for the required system functionality, and (b) the management information and protocols needed to access and control the networked devices. The Network Virtual Device abstraction is used for higher level functions such as resource location and naming. Derived Virtual Devices are lower-level abstractions composed of two components: (1) some portion of an NVD resource (an object), and (2) an execution context which provides a functional interface (methods). A concise description of Netstation architectural components and their interactions is given below.

2.1 Network Virtual Devices

A **Network Virtual Device (NVD)** is a named physical device resource that is attached to the network. An NVD is the object granularity for device naming, resource location, and management within a Netstation system. One or more NVDs may be housed in a chassis along with a network media interface and sufficient processing power to provide and

manage the interface that is presented on the network. Each NVD is managed individually (i.e. it has a name and a description in a device management database), even though multiple devices may reside at the same network interface³.

We have chosen to use Internet Domain Names [8] to identify NVDs on the network. The DNS can then provide the mapping from device name to the address of the network interface where the device resides. Given the Internet domain name (and address) of the device, a client that desires to use the device sends requests to a well-known NVD management port.

The functions of system configuration and resource location are built on top of the NVD abstraction. In the simplest case, resource location can simply be obtaining a pre-configured device name. To configure a file system to use a particular NVD disk, the domain name of the NVD would be sufficient to identify the resource. For example, "sd0a.disk1.isi.edu" would replace "/dev/sd0a" in a file system mount table. In the more complex case of dynamically finding available resources, space, the resource location library routines return NVD domain names. This scheme is modular and flexible as different resource location mechanisms can be used depending on the needs of the particular system that requires device resources.

An important point to note is that data-related commands (e.g. READ) cannot be issued to an NVD. Instead, NVDs accept control commands that create and manage a set of abstract devices intended to support data I/O. These abstract devices are referred to as Derived Virtual Devices.

2.2 Derived Virtual Devices

A **Derived Virtual Device (DVD)** is an abstraction of a physical device that is comprised of (a) all or some part of an NVD's resources, and (b) a set of functions that provide access to, and control of, the device resources. DVDs are created (and destroyed) dynamically, and each is accessed through a port number that is unique for the lifetime of the DVD. This port number also serves as the identifier for the DVD resource; the Netstation system does not maintain a persistent DVD identifier similar to NVD domain names⁴.

DVDs enforce the bounds checking and operational restrictions required for safe shared access, providing lower-level functionality than the naming and management functions based at the NVD level.

DVDs can be derived from the resources of an NVD, or from the resources of a parent DVD. In the former case, the default information maintained about the NVD is sufficient to

³ Requests sent to an interface will contain the name of the desired device in order to multiplex between different devices available via the interface.

⁴ Services built on top of DVDs may retain persistent DVD information including an identifier for some portion of its resources.

specify the derived device mapping. In the latter case, the user/owner of the parent DVD may be offering a more complex, "value added" abstraction based on its DVD (e.g. a file system based on a DVD representing a set of blocks from a disk NVD). Creating a DVD in this case requires that the owner explicitly specify the portions of its DVD to be shared, and the required mapping into the derived DVD.

Section 3 discusses DVD functions in more detail.

2.3 Device Command and Access

We chose RPC as the communication abstraction since it models the request-response nature of bus-based device interactions. Use of RPC implies that the NVD presents one (or more) procedural interface(s) to client applications. We distinguish between the following two types of RPCs:

- **DVD Management Procedures (DMPs)** are sent to NVDs to control and manage device access through the creation and maintenance of DVDs (e.g. `create_DVD()` and `install_DVD_map()`).
- **DVD Command Sets (DCSs)** provide an execution context for each DVD which allows safe shared access to device resources (e.g. `readblock()` and `writepixel()`).

Authenticated RPC is used to avoid unauthorized device access, where the type/level of authentication can be configured locally and varies according to device type and RPC procedure.

2.4 NVD Management

Netstation systems are tied together with a local database that defines the available NVDs and DVDs. This database is known as **the Network Virtual Device Manager (NVDM)**. The NVDM contains information as follows:

- NVD entries comprised of attribute-value pairs that describe characteristics of the available devices (e.g. `NVDname: sd0a.disk1.isi.edu`, `NVDtype: disk`, `NVDblock cnt: 65536`).
- DVD entries that associate (1) an NVD resource, (2) a subset of the DVD Command Set for the NVD resource, (3) an access control list to specify users allowed to create the described DVD, and (4) an indication of the required level of user authentication.

Multiple DVD entries may exist for each NVD to grant different access privileges to each system user. An NVD definition language defines the permissible NVD attribute-value pairs, and includes an enumeration of the DVD command set universe.

Users of Netstation devices consult this database to locate devices that can meet required specifications. Each NVD must consult this database to obtain configuration and access control information for the DVDs it will support.

3. Derived Virtual Devices

A derived virtual device (DVD) is an abstraction of a physical device that can be viewed as a set of resources and an *execution context* at the device. The execution context enforces the desired constraints associated with the device⁵. Clients of a device see a virtual device, which provides a set of services such as nonvolatile storage of blocks (a disk drive DVD) or display of pixels (a frame buffer DVD). This virtual device is mapped to real physical resources by processing resources at the NVD (i.e. a device controller).

DVDs are created by any entity with access to device resources. We use the term *derived* to indicate that the device is constructed from an already existing grouping of resources. The original resource is referred to as the *parent*, and the new DVD as the *child*. The parent resource can either be an NVD or another DVD. In either case, the access rights granted to a child must be a subset of the parent's access rights; this constrains both (a) the set of resources accessible, and (b) the operations that may be performed on the device. Note that NVDs are strictly a set of resources, and do not have associated data-access procedures. Hence, in the case of a parent NVD, the constraints placed on the Device Command Set must be obtained from the configuration information maintained by the local Netstation management database (i.e. the NVDM).

The owner of a virtual device grants access to others by creating a mapping within the set of resources it owns, sending that mapping to the virtual device to create a new virtual device, granting client access to the DVD, and informing its client of how to communicate with the new virtual device.

The new, secondary client is then allowed to communicate directly with the device (via the new DVD), without the intervention of the granting server. The key to ensuring that the secondary client does not overstep its newly-acquired authority to execute commands at the device is that *the device enforces the constraints* of the new DVD. These restrictions are implemented by creating a customized set of procedures, parameterized with the particular DVD limitations and lacking the restricted operations.

DVDs can be nested; any client that has access to a DVD can create a child of that DVD. Although the focus is the use of DVDs for mapping files, once the client has access to the DVD it may assign any meaning to the blocks it chooses. Because a DVD presents the same interface as its parent, it is possible to run systems in a recursive fashion; file systems can be built on DVDs created by other instances of the file system, as in stackable filing [10], or window systems can be run on virtual displays that are actually windows on larger virtual displays, much like Plan 9's 8-1/2 [11]. For example, nesting is also used when the file server grants access to a user process, which can then grant access to other devices to facilitate third-party transfer, as described in section 5.

Section 2.3 introduced the two types of DVD RPCs: DVD Management Procedures and DVD Command Sets. The following sections discuss these interfaces in more detail.

⁵ DVDs are related to the concept of virtual store as defined in the Open Storage Systems Interconnection (OSSSI) model [9]. Differences are noted in section 7.

3.1 DVD Management Procedures

We have defined a protocol which describes the full functionality of a DVD. It is a set of commands used for controlling DVDs which we collectively call the DVD Management Procedures (DMP). These are the commands sent to NVDs which manipulate the DVDs themselves, rather than perform actual I/O operations.

The `create_DVD()` command is the most critical of these procedures. To fully specify DVD creation requires all of the following information:

- An ID of the user that is to be granted access to the new DVD.
- A set of resources. This includes an identifiable resource (either an NVD or existing DVD), and a specification of the resource subset to be accessible by the new DVD.
- A (possibly trivial) mapping from the new DVD address space (e.g. block numbers) into the physical resources.
- The subset of DVD Command Procedures that the user is permitted to use (e.g. cannot use write function).
- Ranges for parameters values for each DVD RPC to enforce constraints.
- Authentication level/type required for each DVD RPC.

All of this information is required so that the DVD creator can establish access constraints on the DVD.

Other commands allow the creator to modify the operating environment of the DVD. For example, it must be possible to dynamically increase the size of a child DVD which represents a file mapping (`install_DVD_map()`). This is superior to the simpler alternative approach where the DVD must be destroyed and recreated, forcing the client to reconnect.

The creator must also be able to determine (normally at child DVD destruction time) certain information about the usage of the child DVD. For example, it may be necessary to receive a list of the blocks that were written to the child DVD, a feature necessary for effective implementation of write before read (described in section 4.4).

Examples of semantic constraints that DVD creators must be able to specify include address remapping and limits and access control features such as read only, write before read and append only (for tape), and restrictions on management operations such as modification of NVD owner lists.

The semantic flexibility provided means that it is possible to define new commands, which might be useful for compression, encryption, storage allocation, parity computation for distributed RAID [12,13,14], and "composite" virtual devices (striping for disks or tapes, treating multiple displays as a single large display, etc.).

Our prototype DVD creation mechanism is based on Scheme [15]. The DVD creator downloads Scheme code at create time, specifying a Scheme function to be executed before and after the execution of each command at the NVD, to adjust argument values (block addresses, etc.) and determine if permissions would be violated by executing the command. Note that use of such a language in a non-prototype environment would raise security concerns that must be addressed. In principle, any of the currently proposed "safe" languages (Java, Safe-Tcl, Penguin, etc.) could be used; for ease of implementation we chose Scheme.

3.2 DVD Command Sets

DVDs provide a set of "data-related" or "I/O-related" commands that can be executed. We refer to these as the DVD Command Sets (DCSs). The interface provided is of course device-specific, and the same device may in fact present several levels of interface. A disk drive, for example, may present a lower-level block-oriented interface, such as the Small Computer Systems Interface (SCSI) or Intelligent Peripherals Interface (IPI), or a higher-level file-oriented interface such as NFS. A display may present a simplified pixel-oriented interface, or a high-level interface that includes windowing functionality, font management, etc., as X Windows does. In general, Netstation devices provide lower-level, device-oriented interfaces. The choice of interface for RPCs executed at the DVD for data I/O is, to a certain extent, orthogonal to the DMP.

As an example, the RPCs appropriate for a disk drive patterned on a SCSI interface include:

- data operations: READ, WRITE, ERASE, COPY, VERIFY
- block management: FORMAT UNIT, REASSIGN BLOCKS, READ DEFECT DATA, READ CAPACITY, error level control, etc.
- buffer management: write caching policy, replacement algorithm, full/empty ratios for initiating data transfer, etc.
- physical control: TEST UNIT READY, START/STOP UNIT (spin up and spin down, eject), and PREVENT/ALLOW MEDIUM REMOVAL (for removable drives), etc.

In the normal SCSI model, READ returns data to the original requestor, and third-party copy is a complex variant of the COPY command. In Netstation, DVDs simplify addressing of data blocks, allowing commands such as READ to simply and nearly transparently become third-party transfers. Third-party transfers are discussed in Section 5.

3.3 Security

The havoc that can be wreaked on a disk drive by misuse of commands such as FORMAT greatly exceeds that of TEST UNIT READY. Thus, the level of authentication and privilege required to execute commands differs.

The level of security required to execute specific RPCs is established at DVD creation. Two factors, the level of authentication and the level of integrity, can be specified independently for each of the two parts of an RPC, the RPC control block and the data. The two parts are

controlled separately because they transit the network separately, and may even have different destinations, as in third-party transfer. The large size of most data segments, compared to the RPC control block, also makes it desirable to allow data to be transferred without compute-intensive operations, such as encryption.

Several levels of authentication are provided. Execution of some commands requires no authentication. Others may use known weak methods such as host source address, which has the two major flaws of being spoofable and not unequivocally identifying *who* at a particular node issued the RPC. The examples in this paper assume that the Kerberos authentication system is used, which we expect to be a common mode of operation.

The integrity of the data transferred can also be selected. Some RPC control or data blocks may be protected only by the network's built-in mechanisms, such as checksumming, which can protect the data against accidental corruption in the network but not malicious tampering, while others require that the integrity of data be assured (perhaps via a one-way hash), a common choice for the RPC command block. Still others may require that all data be protected from modification. Management functions (such as the creation of new child DVDs) typically require the highest possible protection.

4. A DVD File System

DVDs can be used as an enabling technology in file systems. We refer to our file manager as STORM (STORAge Manager). In this section, we detail several system operations, including booting a device, booting the file system itself, reading a file, and extending a file for writing.

Note that transport-level network overhead is not included in these diagrams. As these messages are typically sent reliably, additional packets for connection setup and control may be required. However, these message sequences do include some infrequent operations such as acquisition of an authentication key; such sequences will typically not have to be executed for every operation.

4.1 Booting a Device

When a Netstation device boots, it must configure itself, including determining who is allowed to access it. Some of this information must be retrieved from the device's NVDM. The device's built-in configuration must be adequate to allow it to find and communicate securely with its NVDM. The information the device starts with (stored in nonvolatile RAM or otherwise statically configured) includes the identifier of its NVDM and a secret key it shares with Kerberos. Because this secret key will unequivocally authenticate the Kerberos server, which will authenticate the NVDM, it is not necessary to know the locations of the Kerberos server and the NVDM; the locations may be determined dynamically, perhaps by a multicast on the local network. The steps involved are, taking a disk as an example (see figure 2):

1. The disk authenticates itself to Kerberos.
2. The disk receives a Kerberos ticket to access the Ticket Granting Server (TGS).
3. The disk requests a ticket to access its NVDM.
4. The disk receives the ticket.

5. The disk requests its DVD configuration and Access Control List (ACL) from its NVDM.

6. The NVDM sends configuration info to disk.

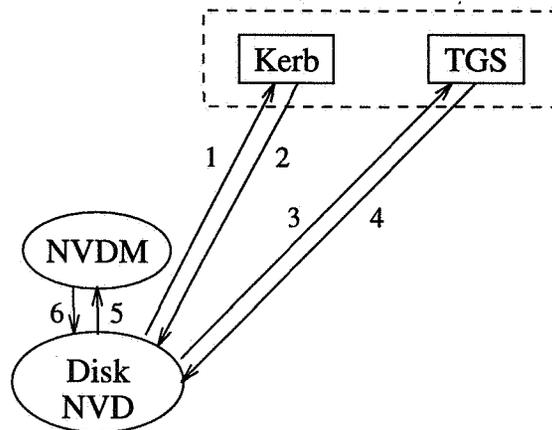


Figure 2: Booting a Disk

In our simple example, the disk receives an ACL indicating that STORM is the only user allowed, and it has unlimited access to the entire disk.

4.2 Booting STORM

Booting STORM requires the following steps (see figure 3):

STORM authenticates, asks for DVD

1. STORM must authenticate itself to Kerberos, the authentication server.
2. STORM receives a Kerberos ticket to access the Ticket Granting Server (TGS).
3. STORM requests a ticket to access the disk.
4. TGS sends STORM the ticket, which contains, among other information, a session key for STORM and the disk to share.
5. STORM requests access to the disk NVD. This is a `create_DVD()` request. In the simple case of the disk containing only a file system managed by STORM, this request will be for unlimited access to the entire disk.
6. The disk checks the permissions, creates the DVD, and returns the DVD identifier to STORM. Setup is now complete, and STORM is free to access the disk NVD, subject to the constraints imposed by the DVD definition.

Data transfer

7. STORM requests a read of the file system superblock.
8. Data is returned.
9. STORM requests a read of the block containing the file system root directory's inode.
10. Data is returned.
11. STORM requests a read of the block(s) containing the root directory.
12. Data is returned.

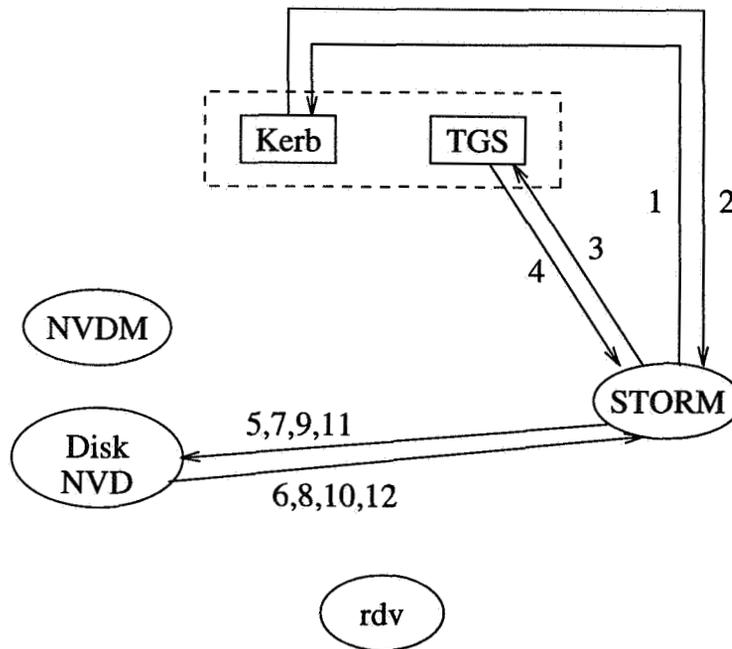


Figure 3: Booting STORM

Note that near the end of the sequence, once the DVD has been established, data requests and responses are processed with a minimum of messages. This is typical of DVD operations; a large number of control messages are used to establish safe conditions for high-speed data transfer. This will be most effective when large amounts of data are to be transferred or many requests executed.

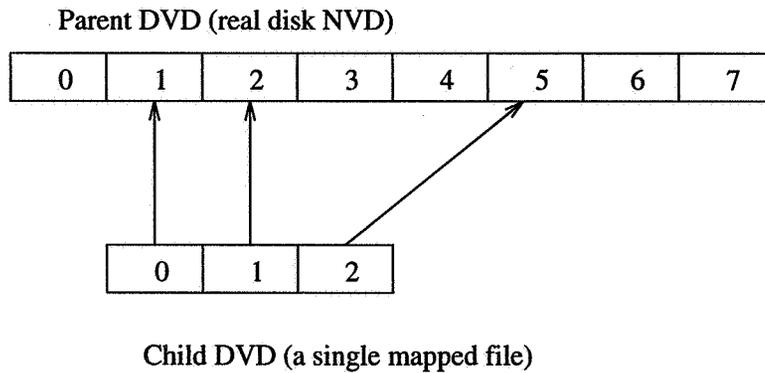


Figure 4: Using a New DVD for a File Mapping

4.3 Reading a File

When an application program opens an existing file, the request is transmitted to STORM. STORM, as the owner of the DVD holding the entire file system, creates a child DVD (with an access list specifying the new user) that includes only the blocks that are part of the file, and returns a DVD identifier (port number) to the new DVD. Figure 4 shows a newly-created DVD that maps a simple three-block file.

Figure 5 shows the steps involved in opening a file through STORM:

1-4. *rdv* authenticates himself to Kerberos and acquires a ticket to access STORM. This is analogous to steps 1-4 of booting STORM.

Establish DVD for *rdv*

5. *rdv* sends a file open request to STORM.

6. STORM determines that the best way to handle this request is to create a DVD for *rdv* at the disk drive, so it sends a `create_DVD()` command to the disk NVD. This command, detailed in section 3.1, contains information about who the DVD is for as well as what access is being permitted.

7. The disk ACKs the DVD create with the appropriate information.

8. STORM bundles the DVD identifier into a package and sends it to *rdv*. STORM may have to include extra information for the file system library code being executed by *rdv*, such as what operations require the cooperation of STORM, how to handle partial blocks, what to do about EOF, etc. DVD setup is now complete.

rdv gets a ticket

9. *rdv*, who has not previously accessed the disk, requests a ticket for this purpose. If subsequent file opens access the same disk, this step will not have to be executed.

10. TGS returns the ticket.

Data transfer

11. rdv sends his first data request to the disk NVD.

12. The disk NVD responds with the data.

13. rdv sends his second data request to the disk NVD.

14. The disk NVD responds with the data.

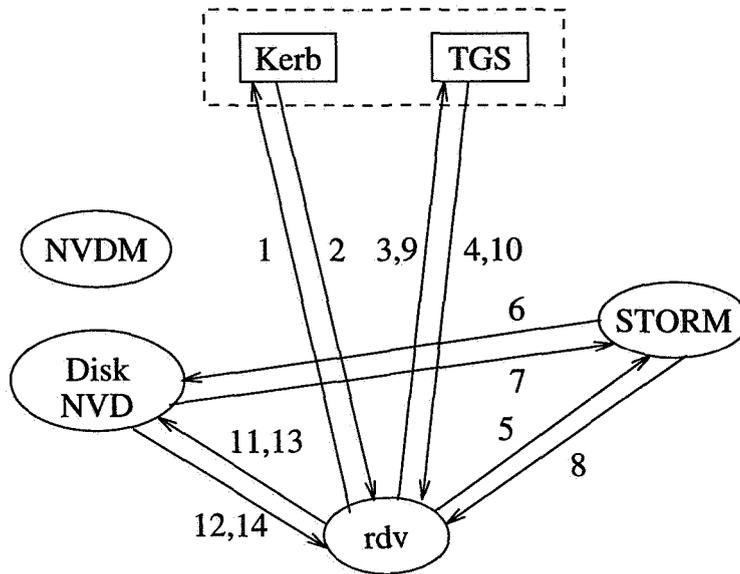


Figure 5: Opening a STORM File

The use of DVD file systems is most efficient for applications in which the data transfer phase is the primary performance bottleneck. The process of opening a file should happen only rarely compared to the number of read/write operations to be performed on the file. If that is not the case (e.g., an application that opens many small files), a normal file system RPC is likely to be more efficient. A storage manager can maintain file-size information to recognize small file requests. Then, rather than establishing a DVD, it can retrieve data and forwards it to the client similar to a conventional NFS interaction.

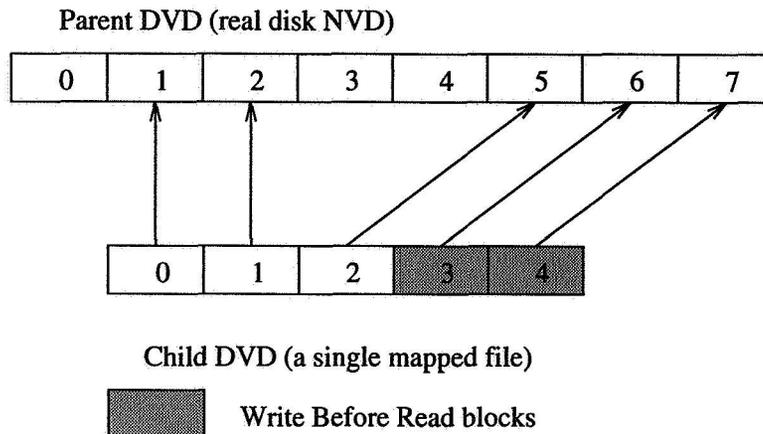


Figure 6: Write Before Read DVD for a File Mapping

4.4 Write Before Read

An optimization we have developed in conjunction with DVDs is *write before read* (WBR). It allows servers to grant access to resources containing sensitive data, without disclosing that data, and without requiring explicit, expensive erase operations.

In a traditional kernel-based system, new blocks are allocated to a user's file when writes are made to the file, or, depending on the FS implementation, when the file's size is extended but not all of the blocks are written. These unwritten blocks cannot be read until they have been written, because they may have once been allocated to a different (now deleted) file containing someone else's private data. This constraint is enforced by the kernel and file system. The safest solution is of course to erase the blocks explicitly before granting access, however this has a large negative performance impact. Thus, the concept of write before read comes into play.

The DVD abstraction allows STORM to create a DVD representing a file, and tailor the DVD Command Set so the `WRITE` procedure is parameterized to enforce the WBR restriction. STORM simply provides (as an optional argument to a create DVD RPC) a list of blocks to be written before they are read.

Figure 6 shows the same file from figure 4, extended two blocks, presumably as a result of a client request to read or write past the end of the physical file allocation. The server that lengthened the file (the owner of the child DVD) marked the two new blocks as WBR, since it knows that those blocks may contain data from having previously been used as part of another file.

When a child DVD is destroyed, the write-first list must be reconciled with its parent. This is executed at the parent DVD. The owner of the parent DVD can request notification of the destruction of the child DVD, and along with it an accounting of blocks that remain unwritten, data which it can use when creating its next child DVD.

5. DVD Third-Party Transfer

Third-party transfer is a mechanism that specifies movement of data, where the party requesting the transfer is neither the source nor the destination of the data. This is a common mechanism in NAP systems that provides support for efficient data transfer between devices, without a copy through the controlling entity.

In this section, we show how DVDs support third-party transfer by presenting an example transfer from a disk drive to a display DVD.

In a Netstation, third-party transfer differs from a primary transfer only in that the locus of control is different; the mechanics of the transfers are the same. It does, however, result in an increase in the number of messages transferred across the network.

One DVD can transfer data to another. This can be done by creating two DVDs, one for the source and one for the destination, that each linearize the area to be transferred, creating a *virtual mapping window* in a fashion similar to the Parallel Transport Protocol (PTP) [16]. As with PTP, the mapping to create the virtual mapping window is done at the storage server, rather than at the device. However, using DVDs, this mapping is then communicated to the devices in the form of the creation of child DVDs. The mapping is then enforced by the child DVDs themselves.

Using a DVD as the destination has the advantage that improper behavior by the source of the data cannot corrupt data at the sink. Giving the source device unlimited access to the destination can allow overwriting or erasing of data if the source misbehaves due to programming errors, concurrency conflicts, or malicious misuse of the source. An important point is that the destination device does not have to trust the source, only the storage server from whom it receives the mapping it enforces. This helps limit the damage that can be caused by security breaches at the devices, though the storage server itself remains the ultimate key to overall system security.

5.1 "Push" Transfer

Figure 7 shows the operations necessary to initiate one type of third-party transfer, "pushing" data from a disk drive to a display. This figure assumes (1) rdv has already opened the file on the disk as detailed in section 4.3, and (2) the display has already booted and retrieved configuration information as explained in section 4.1). From this point, the steps in establishing the connection are:

- 1-2. rdv gets a ticket to talk to the display.
3. rdv sends a `create_DVD()` request to the display, requesting write access to the whole screen for himself.
4. The display ACKs the create with the appropriate information.
5. rdv sends a `create_DVD()` command to the display, giving the disk NVD write access to a rectangular region of the screen, and mapping it so that (0,0) for that DVD maps to the upper left hand corner of the rectangle. This simplifies the disk's access to the display.
6. The display ACKs the create with the appropriate information.

7. rdv sends a third party copy command to the disk DVD he has access to, requesting that the disk drive send data to the display. This first request includes the ticket and DVD identification information the disk needs to access the display, but that information does not need to be transferred for subsequent requests.

8-9. The disk has not accessed the display, so it gets a ticket from TGS.

10. The disk sends data to the display.

11. The display ACKs the command to the disk.

12. The disk ACKs the command to rdv.

Note that subsequent requests can execute much more quickly, since the DVD state is preserved; this eliminates steps 3, 4, 5, and 6. Note also that interactions with Kerberos and TGS may be eliminated for subsequent setups if valid tickets are still held. Caching the ticket eliminates steps 1, 2, 8, and 9, leaving a eight-step process instead of twelve. Additional requests from rdv for data transfer result in four messages:

13. rdv requests the disk drive to transfer data.

14. The disk drive sends the data to the display.

15. The display ACKs the command to the disk drive.

16. The disk drive ACKs the command to rdv.

This is the bare minimum of messages possible. STORM has not had to be involved at all in this child DVD create or the individual I/O operations, because rdv is granting access to resources he already has access to.

Note that this is asymmetric; the disk drive has access to the display, but not vice-versa, because no DVD allowing the display to access the disk drive has been set up.

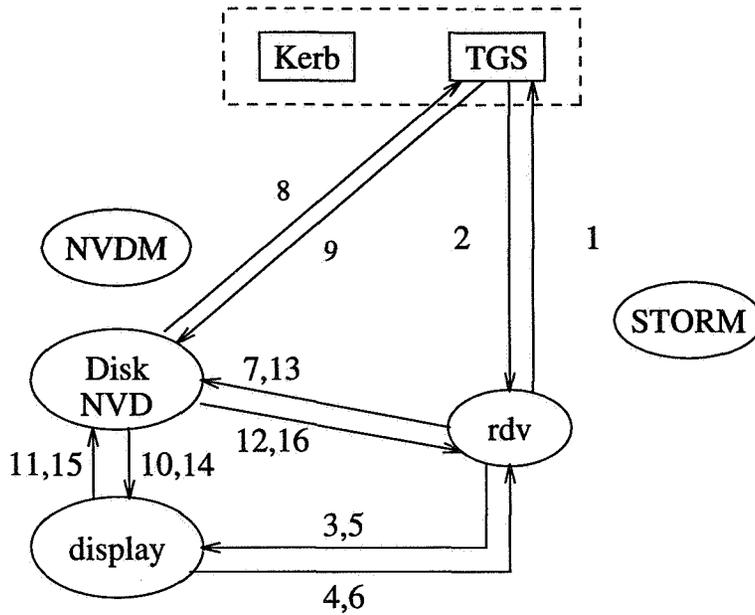


Figure 7: Third Party "Push" Transfer

5.2 "Pull" Transfer

The previous example was shown as a "push" transfer, with the data source initiating the transfer. An equivalent transfer can be set up in the opposite direction, with the display sending READ commands to the disk drive rather than the disk drive sending WRITE commands to the display. This we refer to as a "pull" transfer.

The choice of whether to use a push or pull transfer can be made based on the relative capabilities of the two nodes. If rdv's latency to the two devices is significantly different, the choice can be made to reduce the total time for the four messages necessary for each transfer. A push transfer would be appropriate if rdv has low latency to the disk drive and high latency to the display, and a pull would be the correct choice in the opposite case.

6. Implementation Issues

The client of a DVD (for example, a user process) accesses the DVD as if it were a regular block-oriented device. Library code would implement `read()` and `write()` transparently to application code, thus preserving the investment in software development. A relink may be required, however.

This library code will run entirely in user mode; once the mapping of the file has been done and the DVD created by the device's owner, no further communication with the owner (the storage manager) is required. Reads and writes are done via user-level RPCs directly to the DVD. If the network code runs in the user's context as well (as is done on some high-performance systems), file I/O may be executed entirely without leaving the context of the application. This can be especially useful on systems that provide low latencies on RPCs.

The library read and write code maintains some structures similar to those normally handled by the kernel file system, such as the EOF marker, which must be returned to the true file system at file close (or process termination) time.

The ability to execute file I/O without the intervention of another process or kernel may be especially useful on distributed systems or on multicomputers, where the file system manager may not be local to the client's node. This allows separation of the operations for actually executing I/O from those for managing disk space, directory structures, etc., which may be centralized or distributed without regard to where the I/O must be conducted.

This limited-functionality library will also result in less memory use at the compute nodes. On massively parallel processors (MPPs), for example, the memory savings of not running the full file system code locally on each of a thousand nodes can result in savings of tens to hundreds of megabytes of RAM.

When the process attempts to write past the end of the existing block allocation, the library code recognizes this, and communicates a request to the file manager to extend the file. The file manager then allocates additional data blocks and communicates an updated file mapping to the child DVD. Should the application (either deliberately or through a mistake in the library code) attempt to read or write past the end of the child DVD, an error is returned. See section 4.4 for more details.

7. Related Work

The projects most similar to the basic concept of Netstation are MIT's ViewStation [5] and Cambridge's Desk Area Network (DAN) [4]. Both projects are ATM-specific, and concentrate more on local-area traffic, with careful distinctions between the "inside" and "outside" of the system, whereas Netstation is fully Internet-accessible and has no system boundary.

As already discussed, DVDs have much in common with *virtual stores* from the IEEE Open Systems Storage Interconnect model [9,17]. DVDs differ from virtual stores in several respects. DVDs do not support composite devices, while a virtual store may represent striping across more than one disk, for example. However, DVDs provide more semantic flexibility, in that the owner of a device (or DVD) is allowed to grant any arbitrary subset of its own capabilities (including management functions) to its children when creating DVDs, while virtual store is limited to a data mapping of storage devices. Moreover, although the focus of this paper is on DVD use in file stores, DVDs are more general and may be used for other network attached peripherals such as displays.

Numerous projects have proposed giving the disk node more autonomy, as part of parallel file systems [18,19] or to execute their own space allocation [20,21]. DVDs, with their flexibility and programmability, provide a platform which could be used for similar purposes.

8. Status and Future Work

Much remains to be implemented before Netstation can be considered complete. The network-attached display hardware is complete, and programming of it nearly so. An implementation of the X Window System using the display, with a prototype implementation of the DVD definition mechanism, is complete. A prototype software

version of the keyboard device is under way. Design of the hardware for the camera is under way.

STORM itself, and the user library that accesses it, are in the early stages of implementation. Early goals for the implementation include the ability to use third-party transfer to move data to and from the display, via DVDs. The details of the API for file-related and non-file I/O are still in development. The Kerberos authentication system has not yet been incorporated into the system.

Future research includes defining a composition function so that multiple devices can behave as a single virtual device. As mentioned above, DVDs are typically derived from a single device, however it is desirable to provide a higher-level abstraction to create composite devices.

9. Conclusion

We have shown the design of a device abstraction, derived virtual devices, which provides the efficiency of low-level device access while maintaining many of the protections of higher-level abstractions such as files. We have described a file system design based on DVDs which supports third-party transfers from device to device and allows direct access to the devices by clients at all levels. Derived virtual devices also recurse to allow clients to safely grant access to subsets of their resources to their clients.

References

- [1] Rodney Van Meter. A brief survey of current work on network attached peripherals (extended abstract). *ACM Operating Systems Review*, pages 63-70, January 1996. Full version available on the web at <http://www.isi.edu/~rdv/nap-research/index.html>.
- [2] R. W. Watson and R. A. Coyne. The parallel I/O architecture of the high-performance storage system (HPSS). In *Proc. Fourteenth IEEE Symposium on Mass Storage Systems*, pages 27-44. IEEE, September 1995.
- [3] Randy H. Katz. High-performance network and channel based storage. *Proc. IEEE*, 90(8):1238-1261, August 1992.
- [4] P. Barham, M. Hayter, D. McAuley, and I. Pratt. Devices on the desk area network. *J. Selected Areas in Communications*, 13(4):722-732, May 1995.
- [5] Henry H. Houh, Joel F. Adam, Michael Ismert, Christopher J. Lindblad, and David L. Tennenhouse. The VuNet desk area network: Architecture, implementation and experience. *J. Selected Areas in Communications*, 13:710-721, May 1995.
- [6] Greg Finn. An integration of network communication with workstation architecture. *ACM Computer Communication Review*, October 1991. Available on line at <ftp://venera.isi.edu/atomic-doc/ATOMIC.Netstation.ps> or <http://www.isi.edu/netstation>.
- [7] R. Felderman, A. DeSchon, D. Cohen, and G. Finn. ATOMIC: A high speed local communication architecture. *J. High Speed Networks*, 3(1):1-29, 1994.
- [8] P. V. Mockapetris. Domain names - concepts and facilities. RFC 1034, USC Information Sciences Institute, November 1987.

- [9] IEEE P1244. Reference Model for Open Storage Systems Interconnection - Mass Storage System Reference Model Version 5, September 1994.
- [10] John Heidemann and Gerald Popek. Performance of cache coherence in stackable filing. In Proceedings of the 15th Symposium on Operating Systems Principles, pages 110-127. ACM, December 1995.
- [11] Rob Pike. 8-1/2, the plan 9 window system. In Proc. Summer 1991 USENIX Conf., Nashville, June 1991.
- [12] John H. Hartman and John K. Ousterhout. The zebra striped network file system. ACM Transactions on Computer Systems, 13(3):274-310, August 1995.
- [13] Pei Cao, Swee Boo Lim, Shivakumar Venkataraman, and John Wilkes. The TickerTAIP parallel RAID architecture. In Proc. 20th Annual International Symposium on Computer Architecture, pages 52-63, May 1993.
- [14] Darrell D. E. Long, Bruce R. Montague, and Luis-Felipe Cabrera. Swift/RAID: A distributed RAID system. Computing Systems, 7(3):333-359, 1994.
- [15] William Clinger and Jonathan Rees (editors). Revised Report on the Algorithmic Language Scheme, November 1991.
- [16] Lawrence Berdahl. Parallel transport protocol proposal. Lawrence Livermore National Labs, January 3, 1995. Draft. <ftp://svr4.nersc.gov/pub/Pio-1-3-95.ps>.
- [17] IEEE P1244. Virtual Storage Architecture Guide, March 1995.
- [18] Peter C. Dibble and Michael L. Scott. Beyond striping: The Bridge multiprocessor file system. Computer Architecture News, 19(5), September 1989.
- [19] David Kotz and Nils Nieuwejaar. Flexibility and performance of parallel file systems. ACM Operating Systems Review, 30(2):63-73, April 1996.
- [20] Robert M. English and Alexander A. Stepanov. Loge: A self-organizing disk controller. In Proc. Winter '92 USENIX, pages 237-251, January 1992.
- [21] Garth Gibson. Secure distributed and parallel file systems based on network-attached autonomous disk drives. White paper, September 1995.

52-82

Cooperative High-Performance Storage in the Accelerated Strategic Computing Initiative

83184

Mark Gary, Barry Howard, Steve Louis, Kim Minuzzo, Mark Seager

Lawrence Livermore National Laboratory¹

P.O. Box 808

Livermore CA 94550

mgary@llnl.gov

Tel: 510-422-7527

Fax: 510-423-8715

Abstract

The use and acceptance of new high-performance, parallel computing platforms will be impeded by the absence of an infrastructure capable of supporting orders-of-magnitude improvement in hierarchical storage and high-speed I/O. The distribution of these high-performance platforms and supporting infrastructures across a wide-area network further compounds this problem. We describe an architectural design and phased implementation plan for a distributed, cooperative storage environment (CSE) to achieve the necessary performance, user transparency, site autonomy, communication, and security features needed to support the Accelerated Strategic Computing Initiative (ASCI). ASCI is a Department of Energy (DOE) program attempting to apply terascale platforms and problem-solving environments (PSEs) toward real-world computational modeling and simulation problems. The ASCI mission must be carried out through a unified, multi-laboratory effort, and will require highly secure, efficient access to vast amounts of data. The CSE provides a logically simple, geographically distributed, storage infrastructure of semi-autonomous cooperating sites to meet the strategic ASCI PSE goal of high-performance data storage and access at the user desktop.

Introduction

The Accelerated Strategic Computing Initiative (ASCI) [1] is a critical element of the Department of Energy (DOE) response to recent decisions ending nuclear testing. In the past, a large part of integrating fundamental science into nuclear weapon safety, reliability, and performance was accomplished through underground testing. In the future, the simulation capabilities provided by ASCI will provide much of that integration. The DOE Science-Based Stockpile Stewardship (SBSS) program, which encompasses ASCI, plans to build upon a strong foundation of simulation capabilities, non-nuclear testing, and basic science to assess the performance of nuclear stockpile systems, predict their safety and reliability, and certify their functionality.

A primary ASCI strategy is to facilitate an unprecedented level of cooperation among the three DOE Defense Programs (DP) laboratories (Lawrence Livermore, Los Alamos, and Sandia National Laboratories). These three DP labs (Tri-Lab) will share ASCI code development, computing, storage systems, and communication resources across laboratory boundaries in joint development efforts, while still maintaining a high degree of local

¹ This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under contract number W-7405-Eng-48, and supported by DOE Defense Programs Accelerated Strategic Computing Initiative funding.

autonomy. Together with a philosophy of creating a cooperative management environment, there are several technical strategic thrusts within ASCI. Each of these technical strategies influences the need for a high-performance, cooperative storage environment.

ASCI is developing new software applications for virtual testing and prototyping that integrate 3-D capabilities, fine spatial resolution, and accurate physics. These applications will generate much larger quantities of simulation data (multiple petabytes) than ever before. Access to other large data collected from non-nuclear testing and historical underground tests will also be needed to verify and validate results of new applications. ASCI is pushing the computing industry to develop high-performance computers with processing speeds and memory capacities thousands-of-times greater than currently available computers, and tens-to-several-hundred-times greater than the largest computers likely to result from current trends. Much larger capacities and data transfer rates will be required from ASCI storage resources to keep up.

ASCI will require an effective problem-solving environment (PSE) to support application code teams. The PSE provides a robust computing environment where codes may be rapidly developed. The PSE also provides an infrastructure to allow applications to execute efficiently on ASCI compute platforms and to allow easy accessibility to terascale resources from the desktop. This computational infrastructure will consist of local-area and wide-area high-speed networks, software development and visualization tools, and advanced storage facilities (see Figure 1).

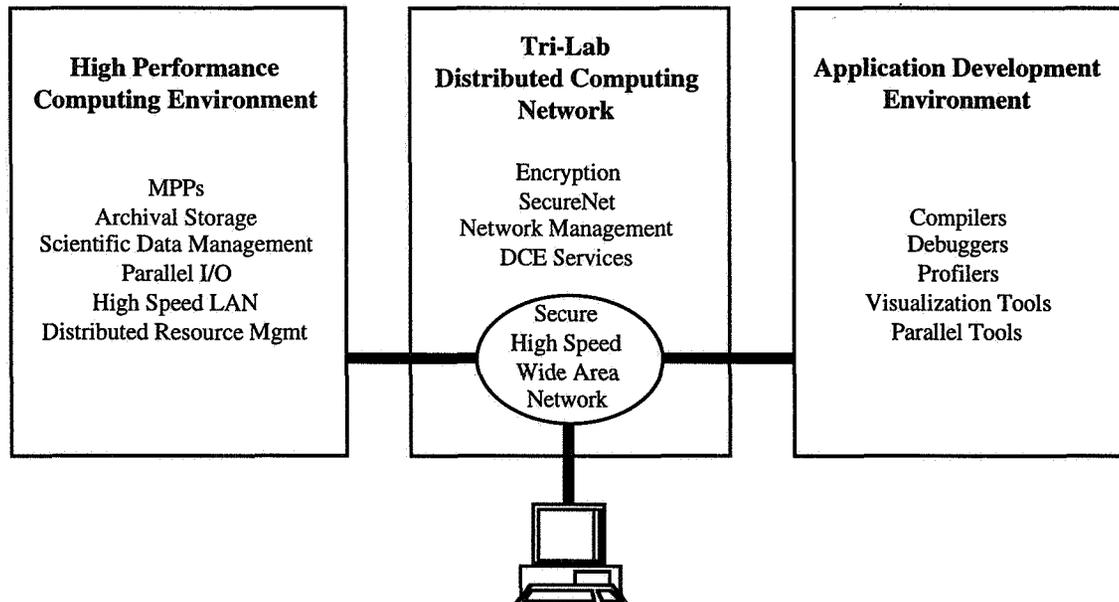


Figure 1. The ASCI Problem Solving Environment

Motivation

To make effective use of terascale platform and application capabilities being developed in ASCI, scientists will need to store, access, and manipulate unprecedented amounts of data. The inability of existing data storage systems and I/O capabilities to perform these tasks are now being recognized. Recent projects, such as the San Diego Supercomputer Center

initiatives in data-intensive computing and massive data analysis, have started to explore better architectures for multi-teraFLOP computing, multi-petabyte storage, integrated databases and archives [2]. Without improvements, users will spend significant time and effort working around storage problems and I/O bottlenecks, and therefore be unable to realize gains from the ASCI terascale environment. The ASCI PSE concentrates heavily on high-performance archival storage, hierarchical storage management, and scalable I/O to address successful use of data. Other important areas in the PSE, closely related to storage but outside the scope of this paper, are scientific data management and visualization (i.e., tools to aid locating and understanding data), parallel file system advancements (i.e., mechanisms for intelligent application use of scalable I/O), and high-speed interconnect fabrics (i.e., networks to provide efficient local and remote transmission of data).

ASCI is an application-driven effort, and as such, an overall PSE goal is to develop needed software and deploy necessary hardware to intelligently present designers and developers with all of the resources they will need. This view of convenient one-stop, *black-box* desktop access implies that individual components comprising the overall resource environment must cooperate. Requirements of cooperative storage differ from those of the scientific data management and visualization areas of the PSE. Cooperative storage is primarily a data logistics problem, concerned with how data is moved, shared, supplied, and stored. In contrast, scientific data management is more accurately a data semantics problem, concerned with finding and understanding meaningful data. Both are needed to provide a true *end-to-end* data management solution within ASCI.

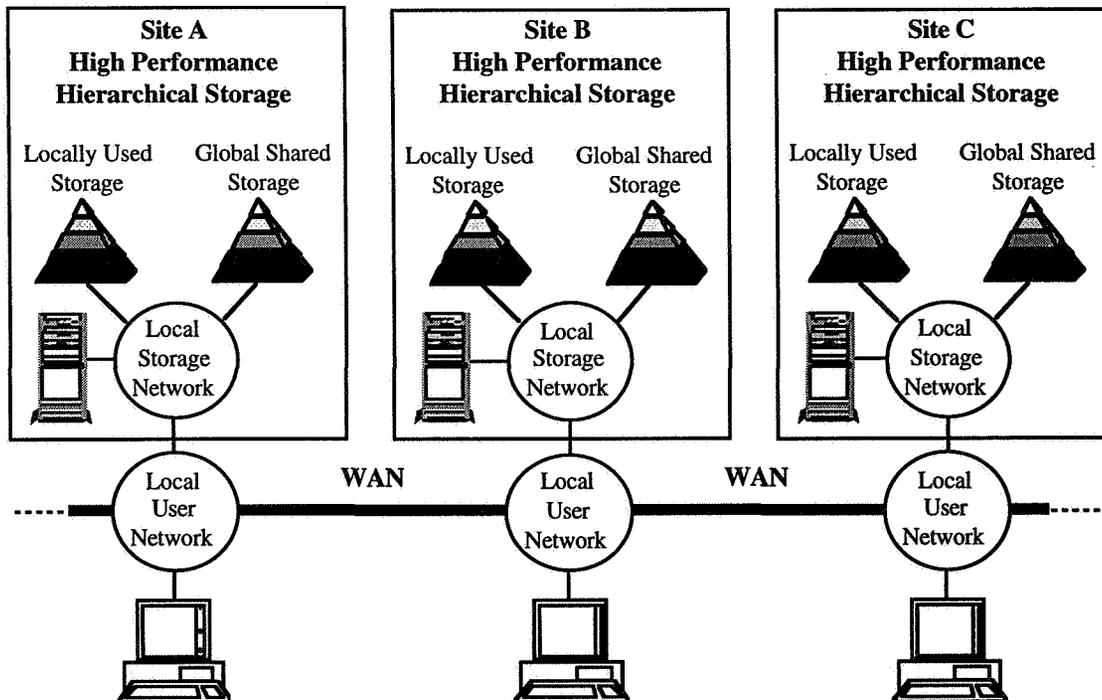


Figure 2. Integration of Autonomous HSM Systems

In this paper, we concentrate on the data logistics problem. We define cooperative storage as an *interworking* group (sometimes called a *federation*) of autonomous, hierarchical storage management systems (HSMs), joined to meet terascale computational needs through common shared vision and coordination, but employing local integration and administration (see Figure 2). HSMs provide management of a hierarchical set of storage

devices and file *stewardship* services to its clients. Individual systems tend to view their resources as local and other systems' resources as remote. Within ASCI, systems will contribute part of their storage resources to the cooperative ASCI whole in compliance with well-defined Tri-Lab global policies. At the same time, systems can also provide local resources that are not sharable, again in agreement with management policy.

Coordination of networked block-data servers to implement a distributed, parallel data storage system has been investigated in other work, notably the Image Server System network-distributed data server [3]. Still other projects have explored integration of mass storage with file systems [4,5]. Unlike those efforts, ASCI attempts to integrate autonomous HSMs into a cohesive whole across a high-speed WAN, each HSM composed of multiple servers distributed across a local network. The NASA EOSDIS Project [6] has similar HSM requirements in its attempt to build a globally distributed, heterogeneous, autonomous data system for the Earth Sciences community. EOSDIS is based on a cooperative architecture where data archiving and distribution services can be configured to achieve a balance between centralized management and distributed authority.

Requirements

There exists a rich literature on the requirements of distributed file systems [7]. In this section, we investigate critical requirements that an ASCI cooperative model places on HSMs. Many distributed file system and cooperative HSM requirements overlap due to common issues of resource sharing and remote access. ASCI will require its terascale resources to be viewed in uniform ways (i.e., as part of a seemingly single, local, scalable resource space that can be globally optimized). Scalability becomes especially important as systems grow more complex. One of the ASCI PSE goals is to bring what appears to be unlimited storage and processing capacities to the desktop.

Transparency

In a black-box view, users should not be concerned with how, where, or in what format data is stored. Various transparencies can be provided to reduce or hide the complexity of system interaction and internal detail, but there may be disadvantages if efficiency is lost in resource utilization or performance. We discuss some important transparencies for ASCI below.

Access and location transparency are related. Access transparency (sometimes called network transparency) hides whether objects or services are local or remote. Location transparency hides where objects or services are physically located. Both help make a system easier to use. A related transparency is migration transparency, where users are unaware that an object or service has moved. These are key requirements of a cooperative storage design. In many systems, a user often must explicitly know where objects (typically files) are stored. An example is FTP access to a remote site where, to obtain data, a user must 1) know where the data is, 2) contact that site explicitly, and 3) login to view or access files. ASCI's black-box philosophy will require that the burden of multiple access methods and needing to specify data location be eliminated or made easier for users and applications.

A related transparency is data representation transparency, where users are not aware that different data representations may be used in local and remote parts of the system. FTP may require the user to specify binary or ASCII for proper data transfer. Although standard access interfaces such as FTP are required and will be supported by ASCI,

distributed file system interfaces that can better support these transparencies will also be required. An important ASCI requirement is that data stored using a specific type of interface be accessible by another. For example, a file stored using a local parallel file system interface should be accessible to the user (at the desktop) through a distributed file system interface. Note that, although the data may have been stored in parallel, actual distributed file system access may be serial. The data may also be physically stored in several different formats throughout the HSM.

Another transparency is naming. We define naming transparency as the ability to obtain a single namespace view, typically as a directory structure, independent of data location or access interface. Storage objects should have globally unique identifiers independent of resource location and access mechanism. Satisfying the naming transparency requirement presents users (logged into any part of the ASCI cooperative environment) with identical namespace views of storage resources.

Two other transparencies are replication and administration. Replication transparency hides the fact that objects or services may be replicated within the system (usually for performance or reliability reasons). Clients may not necessarily know if objects or services are replicated, and services may not know if clients are replicated. Administration transparency provides a management process with the same view of manageable resources, independent of the location of those resources. This allows administrators to assemble resources to fulfill user needs without an obligation to know what resources were local or remote. Such administrative transparency will, of course, be constrained by global management policies.

Site Autonomy

While transparency eases access and use, maintaining local autonomy or control of local resources is also important. Some sites will have local policies that they may be required to enforce, such as security, accounting, scheduling, and system administration. A site may also need to restrict access to particular data or hardware resources on a *need-to-know* basis, or may simply desire a greater degree of control over local assets. There are failure scenarios where a site may need to isolate itself from the cooperative whole while still providing service to local users. Likewise, sites need to be able to operate, possibly in a degraded fashion, when other sites or components of the cooperative are unavailable. To support these situations, interworking sites must strike a balance between local management of resources through site-specific policies and continuing to function as a usable part of the cooperative environment. The introduction of greater autonomy may result in much more visible component boundaries to users.

Communication and Security

Because extending the reach of scientists from the desktop to Tri-Lab resources is key, an ASCI cooperative storage infrastructure must not be isolated from the communication and security infrastructure shared by other ASCI elements. The ASCI communication and security infrastructure provides services and tools to support creation, use, and management of distributed applications in a heterogeneous environment. It also provides services that allow distributed applications to interact securely with a collection of possibly heterogeneous computers, operating systems, and networks, as if they were a single system. It is a requirement that cooperative storage work with, and be manageable within,

this infrastructure. It would also be prudent for the HSM's internal component interactions to use this same communication and security infrastructure.

Performance

Even if all of the requirements listed above are satisfied, a system must meet minimum performance requirements for its users. Providing transparency, autonomy, security, and communication is valueless if the software and hardware resources providing them are unable to provide data storage and access in acceptable time. For ASCI, performance will be a dominant requirement.

Architecture

To satisfy the above requirements, the ASCI PSE effort has established a preliminary architectural vision for a cooperative high-performance storage environment (CSE). This vision brings together distinct elements of storage to form a cohesive, unified, orderly whole. The desired result is a logically single, geographically distributed, storage system of semi-autonomous cooperating sites. The system will be designed and implemented to meet the goal of high-performance data access at the desktop. In this section, we describe aspects of a cooperative architecture that target our requirements. We address each area and discuss the planned architectural approach to satisfy the requirement. ASCI is a multi-year effort, and therefore a phased implementation approach is planned.

Achieving Transparency

To achieve the transparencies described in the requirements section, our approach uses interfaces that naturally provide transparency to users, and tie those interfaces through additional linkage of Tri-Lab name spaces and data spaces. This provides ASCI users of non-transparent interfaces many of the beneficial properties afforded users of transparent ones. The CSE will use DFS [8] as a key file system interface. This interface provides users with a serial interface to storage that satisfies many transparency requirements. Interfaces to local NFS systems will also exist and be supported by CSE HSMs. We chose to concentrate on DFS because it is better able to support the wide-area requirements of ASCI and the PSE middleware infrastructure.

DFS uses a set of cooperating clients and servers to provide geographically separated users with a single, seamless view of a distributed name and data space. DFS also has enhanced data caching capability and consistency guarantees. While DFS satisfies several transparency requirements of ASCI, it lacks an archival back-end needed to support the petabytes of data generated by ASCI. To address this problem, we plan to exploit an extended definition of the DMIG DMAPI [9] specification to link DFS with an appropriate back-end HSM capable of satisfying the large storage requirements of ASCI. An HSM-DMAPI gateway will provide interoperability and communication between the DFS Server and the HSM (see Figure 3).

Another deficiency of DFS is that it does not currently satisfy the high-performance, parallel-access requirements of many ASCI applications and users. Overcoming the serial performance shortcomings of DFS will be challenging. We plan to investigate ways to expand the DFS interface for parallel transfer of data, where possible. While we hope to enhance DFS data transfer performance, we also need to provide other user interfaces capable of satisfying the performance requirements of ASCI. To satisfy transparency requirements, these storage interfaces will all need to operate within a common namespace shared by all user storage interfaces.

The strategy to allow all storage interfaces to view, and operate within, a single namespace is grounded in two architectural decisions. The first is that the namespaces of interworking sites will be linked together, thus allowing users to navigate a single global namespace. This linking will be accomplished by allowing the software that implements naming at each site to *link* leaves of its directory structure to points in other site namespaces. This approximates the idea of a UNIX™ mount, but more accurately mimics a UNIX soft link.

The second architectural decision concerns namespace consistency. Storage interfaces that have their own namespaces by design (e.g., DFS), must be integrated into the CSE such that the namespace provides a global namespace view. If interfaces are implemented as multiple namespaces joined to provide a single view, they must be kept consistent with other views. This presents a significant challenge. For example, back-ending DFS with an archival HSM as was done in [5] will not be enough. We require that the DFS namespace provide users with access to all data (given proper authorization) created by every interface in the CSE. This means that a file created by a non-DFS parallel interface will need to appear in the DFS interface's namespace.

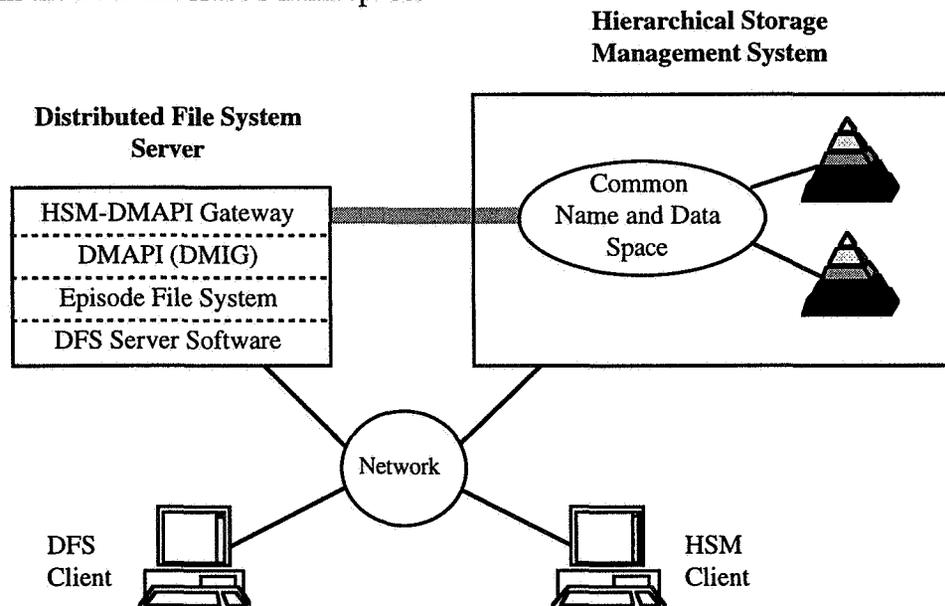


Figure 3. Integration of DFS and HSM Name and Data Space

The above architectural decisions lead to a design with location, access, and name transparencies, but do not necessarily satisfy requirements for administration or replication transparency. Administrative transparency implies the ability to assemble resources located across interworking sites into useful configurations, taking advantage of the strengths and bypassing the weaknesses of each site. Later phases of CSE design encompass linking the storage system management (SSM) internals of the server components of each site together and with all server components. This enables construction of hierarchies that span geographically separated sites. There are also plans for introducing future file and physical volume replication capabilities into the CSE that can span sites.

Achieving Site Autonomy

Satisfying site autonomy requirements in an architecture that links name spaces and joins administrative domains is difficult. Our design maximizes site autonomy within a single

system through incorporation of policy modules, and by allowing a site to continue functioning, albeit in degraded fashion, when disconnected from other participants in the CSE. Similarly, sites remaining in the CSE are allowed to operate in the absence of one or more sites.

Policy modules allow enforcement of SSM concerns that are likely to differ from site to site by isolating local rules or decisions into separate modules that may be implemented or modified by each site to suit its own needs. The CSE design provides for integration of policy modules by establishing well-defined interfaces to and from storage service components. Modification of the main components of the system is not required; only the policy modules need to change. An example is implementation of accounting. The CSE maintains a well-defined interface to an accounting policy module. As accounting events are encountered, policy module interfaces are used to communicate accounting information. The module that accepts this information will be customized by the site and will then perform site-specific actions appropriate to log, report, or ignore the information. The CSE design currently includes policy interfaces for accounting, scheduling, security, migration and purge and system management. The modeling of policies as general objects in distributed systems is explored in considerable detail in [10].

Architecting the system to allow independent operation, in the face of communication failure or other adverse events, relies on how server components of the CSE are linked. If servers are connected in a way that requires all information to be successfully exchanged over inter-site communication lines for a server to function, then autonomy will be lost. If servers are instead connected through simple remote location linking information, more autonomy is possible. The CSE uses the latter approach. Imagine two name server components residing within separate sites (see Figure 4). Each name server represents an autonomous directory structure that may have leaf nodes that are links or pointers to directory or file resources located in the other name server's structure. When traversing a pathname that crosses one of these links, the CSE is designed to realize that the next component is actually managed on a remote name server and contact that server for continued parsing of the path name.

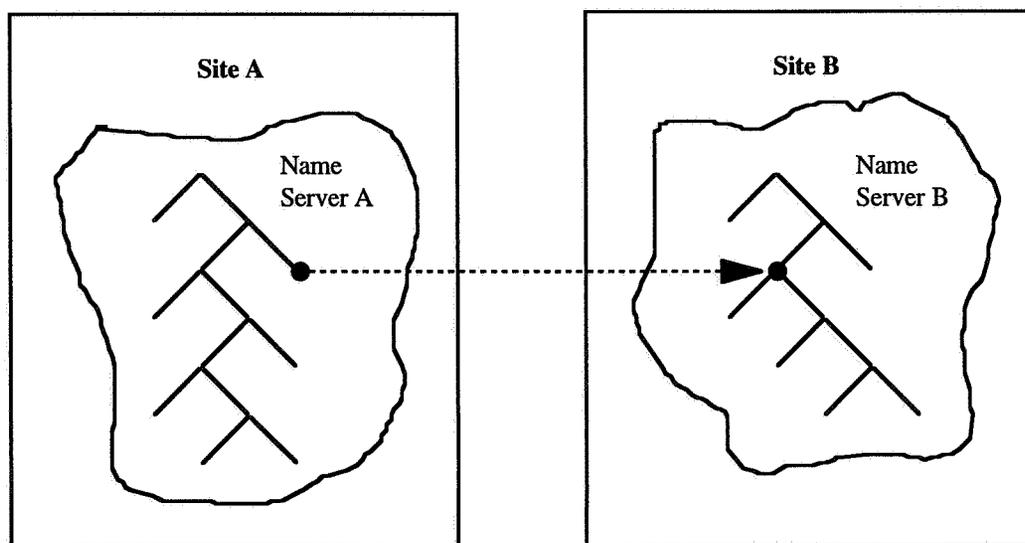


Figure 4. Linked Name Servers at Multiple Sites

Under the above scenario, one can imagine a situation where Name Server B is not reachable. Site A still has the ability to traverse all of its own namespace, but cannot access names located in Site B's space. In this case, depending on the distribution of data across sites, the absence of Site B will degrade access to data, even if names are accessible. This demonstrates a tradeoff between autonomy and administration transparency. Typically, as administrative transparency increases, site autonomy decreases. Striking a balance between these two requirements will affect the eventual implementation. Note that if an HSM is well designed and thus able to scale within a single site, many difficulties linking multiple sites will be moot. In the linked name server example above, one could have easily considered the two name servers to be at a single site, scaled because the namespace was too large to be supported by a single name server. Single-site name server scalability requires the ability of pathnames to span local name servers. Thus, solving this problem addresses many issues concerning multi-site name server cooperation.

Communication and Security Infrastructure

Because the ASCI environment is built on a wide-area fabric of distributed services, it is important that the CSE function as an integral part of this fabric. In our design, the HSM is constructed using this fabric to tie together its various distributed components. The ASCI PSE uses OSF's Distributed Computing Environment (DCE) as a convenient middleware layer to tie ASCI PSE components together. DCE provides services and tools that support the creation, use, and maintenance of distributed applications and servers in a heterogeneous computing environment. DCE also provides services that allow distributed applications and servers to interact securely with a collection of possibly heterogeneous computers, operating systems, and networks as if they were a single system. The CSE uses these services as the glue between each of its server components as well as for the mechanism linking HSMs with their clients and other components of the ASCI environment.

Because much of the work to be done within ASCI is classified and requires the use of *need-to-know* boundaries, authorization techniques will be critical to successful information sharing. The DCE Security Service maintains a replicated registry database that includes principals, users, groups, organizations, accounts and administrative policies. Sites within the CSE can make use of this security service by implementing calls to the service within their site-specific policy modules. Access Control List (ACL) entries in the database are used to define and grant permissions to an object. Any request by a user or application to use an object, such as an archival storage device, is accompanied by the requester's credentials which are checked against the ACL for that service.

Of equal importance to the authentication and authorization mechanisms provided by DCE is the security of data itself as they traverse communication networks. The security of WAN communications between distributed ASCI sites is provided by networks front-ended by high-performance encryptors, providing NSA-approved encryption of all data passing between classified ASCI sites.

Achieving Performance

As stated above, the transparencies inherent in the CSE architecture are not really useful if minimum performance needs of applications and users are not met. If performance is lacking, users will recognize the difference in speed when accessing data on remote systems, and as a result will find ways of working around the system to avoid delays, often at the expense of the system as a whole.

To support the performance requirements of ASCI, the CSE uses a scalable, parallel architecture to support high-performance data access. Under this design, sites can enhance performance by adding server components and/or wider stripes of data until a target performance level is achieved. For example, if a site requires a 100 megabyte/sec transfer rate for a file, but only has 20 megabyte/sec devices, the CSE allows the site to attain the desired transfer rate by storing data using a five-wide stripe across multiple devices at a site.

While the architecture we have chosen provides, in theory, the performance necessary to satisfy the requirements of ASCI, realizing this goal is highly dependent on additional advances in networking hardware, protocols, and encryption technology. Without gains in these areas matching the I/O performance gains seen in storage devices and platforms, the functionality required by ASCI will not be easily obtainable. The CSE architectural design and implementation will closely track these disciplines and react as necessary to integrate advances in these areas as they become available.

Implementation

The ASCI implementation strategy for a CSE focuses heavily on the High Performance Storage System (HPSS), an on-going effort to develop a scalable, high-performance, HSM for data-intensive applications and large-scale computers. Coordinated by IBM Government Systems, three of the principal developers for HPSS are also the three ASCI DP laboratories: Lawrence Livermore, Los Alamos, and Sandia National Laboratories. Oak Ridge National Laboratory, other federal agencies, and universities also participate in HPSS development. From the beginning, a major motivation for HPSS was to develop a high-speed storage system providing scalability to meet demands of new high-performance computer applications where vast amounts of data are generated, and to meet the needs of a national information infrastructure [11].

HPSS has a scalable, networked-centered architecture [12] and is based on the concepts of the IEEE Mass Storage System Reference Model, Version 5 [13]. The architecture includes a high-speed network for data transfer and a separate network for control. The control network uses OSF/DCE Remote Procedure Call technology. In an actual implementation, control and data transfer networks may be physically separate or shared [14]. Another key feature of HPSS is support for both parallel and sequential I/O. The parallel I/O architecture is designed to scale as technology improves by using data striping and multiple data movers [15]. HPSS was designed to support data transfers from hundreds of megabytes up to multiple gigabytes per second. File size scalability must meet the needs of billions of data sets, some potentially terabytes in size, for total storage capacities in petabytes.

The scalability features of HPSS are important for enabling distribution and cooperation of storage resources. The ability to introduce multiple, distributed servers as needed into an HPSS implementation is critical for both performance and autonomy reasons.

Distributable HPSS servers allow us to obtain a geographically distributed single-system view required by ASCII applications and users. Because HPSS supports network-attached peripherals [16,17] and third-party data transfer capabilities, new hardware can easily be added to provide incremental performance scalability. Each ASCII site maintains a degree of local autonomy and can tailor data rates and capacities needed for its local requirements and for resources shared across the Tri-Lab environment. The tailoring of storage resources and the access to them can be controlled and used through the HPSS Class of Service structures [18] and device hierarchy structures. Class of Service provides the control and management flexibility needed to implement a truly distributed storage hierarchy. HPSS also supports policy modules and uses a management-by-policy philosophy in its storage system management design.

The ASCII PSE working group defines a four-phase approach to realize a fully cooperative storage environment. These phases, described in the next sections, are:

- Independent HPSS systems linked via an encrypted WAN
- Independent HPSS systems with a single OSF/DCE DFS name space
- Cooperative distributed systems linked at the internal server level
- Cooperative distributed systems with single system/management view

Phase I

The first phase establishes independent autonomous HPSS systems at each of the Tri-Lab sites. Each site will operate a stand-alone HPSS system with the only linkage being a high speed encrypted WAN connecting each site (see Figure 5). Over this WAN, users will be able to use explicit interfaces such as FTP (assuming they are granted remote privileges) to communicate with remote sites. No transparency will be provided for remote access, but site autonomy will be more or less total. During this phase, significant planning and coordination will be undertaken to ensure that the site policies and DCE configurations at each site do not preclude any cooperation necessary when migrating to future implementation phases.

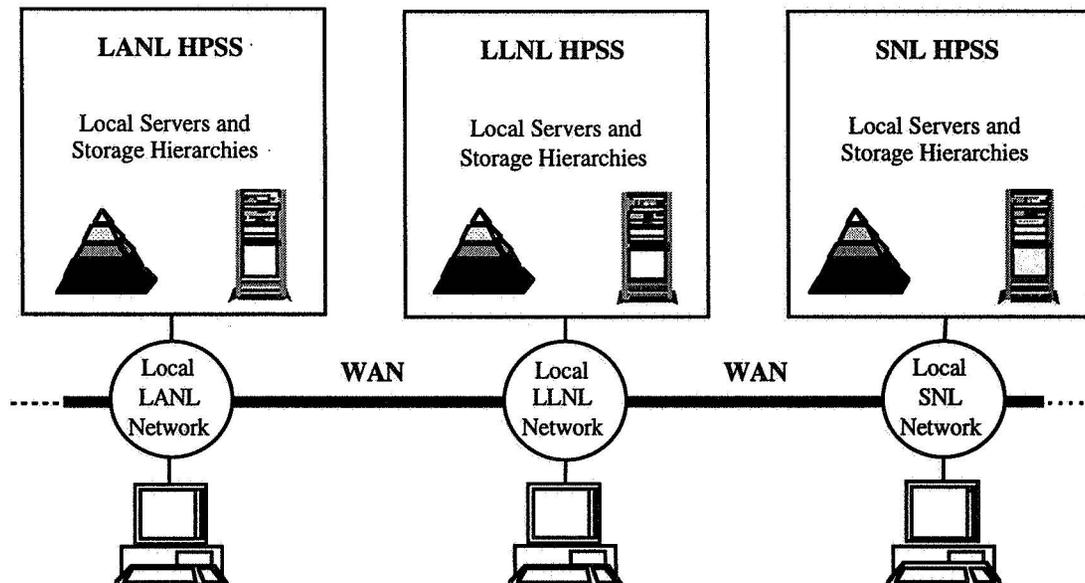


Figure 5. Phase I - Independent HPSS Systems

Phase II

In Phase II, the Tri-Lab sites will attain linkage of name spaces and gain some access transparency, using DFS. Each of the sites will support a DFS server or servers that are back-ended by HPSS (see Figure 6). The DFS namespace will export the view of each site's files to other sites that have linked their DFS servers to the global DFS space. Files created by DFS clients will be created and stored in the HPSS system that back-ends the DFS server where the new file is placed. Users will be able to access files at all sites using DFS, but will have to explicitly contact the location that stores a file to access the resource using any interface other than DFS. Thus in Phase II, the only interface that can see and access all of the storage resources at all sites without explicit login to remote sites will be DFS. Although the DFS interface provides functional transparency to users in this phase, users will be able to discern latency when accessing remote files. Note that data stored in parallel at a remote site will be accessible serially through DFS and other serial interfaces. Phase II introduces the first transparency aspects to the CSE while maintaining all of the autonomy provided in Phase I.

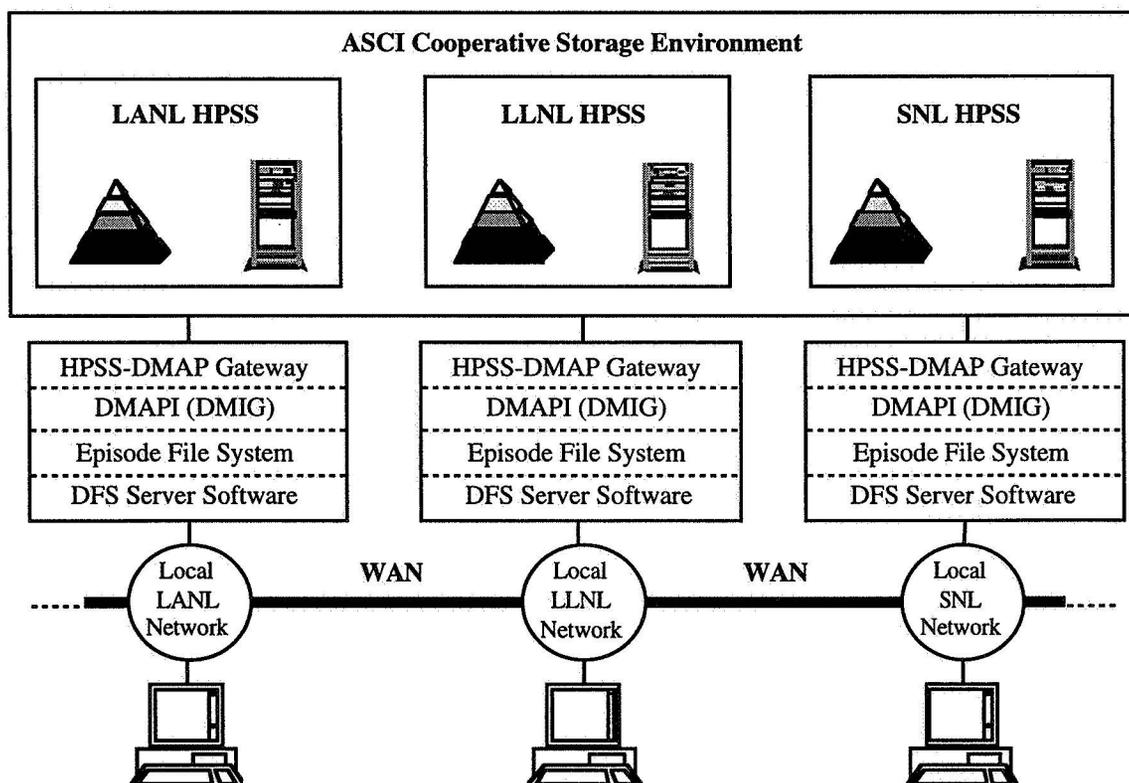


Figure 6. Phase II - Linked HPSS Namespaces with DFS

Phase III

Phase III builds on Phase II by linking the Name Servers of each site's HPSS system and allowing clients to seamlessly communicate with other sites for data access when necessary. The Phase III feature making this possible is implementation of links within a

namespace pointing to a directory or file in the namespace of a remote HPSS system (see Figure 7). Parsing a pathname that crosses a link between two sites will be performed transparently, as will remote data access. This provides the ability for all CSE user interfaces to automatically and transparently view or access the namespace of all Tri-Lab sites.

In Phase III, given adequate networking performance, many of our transparency requirements are met while still maintaining a high level of site autonomy. Because each site still runs its own HPSS system, it can function independently, possibly in degraded fashion, when other Tri-Lab sites are unreachable. In such a failure scenario, all accesses to files or names stored at remote sites will fail. Because users will be allowed to use remote resources, Tri-Lab sites will have some common aspects of storage system management. While this in-common management does not preclude use of unique policy modules at each site, it will be necessary to work out issues such as accounting for remote user accesses. While Phase III goes far toward the final vision of a single distributed HSM, it does not achieve administrative transparency.

Phase IV

Phase IV maintains all functionality of Phase III, but adds administrative transparency across the Tri-Lab sites. This addition will be made possible through linking and coordination of the Storage System Management components of each site. This allows an administrator at one site, given adequate privileges, to assemble storage resources existing at multiple sites into storage hierarchies that take optimum advantage of the array of capability and capacity devices distributed throughout the Tri-Lab environment. The addition of administrative transparency capabilities yields a single logical HSM from a management view, but results in some sacrifice of site autonomy as hierarchies are built requiring resources at multiple sites to all be available in order to provide service. The level of autonomy lost will be related to how much inter-site sharing of resources occurs.

How far the CSE implementation should proceed is a matter of debate. Accomplishing Phase III may satisfy those CSE requirements deemed most critical without sacrificing too much site autonomy. On the other hand, Phase IV, or a slightly enhanced version of Phase III implementing rudimentary administrative transparency, may provide ASCI with more optimal capabilities. A third perspective is to de-emphasize site autonomy. This position argues that a geographically distributed, yet single HPSS system is best for ASCI (note that Phase IV as presented is a *logically* single combination of independent HPSS systems). Time, experience, and the performance of remote data transfers will determine the proper choice.

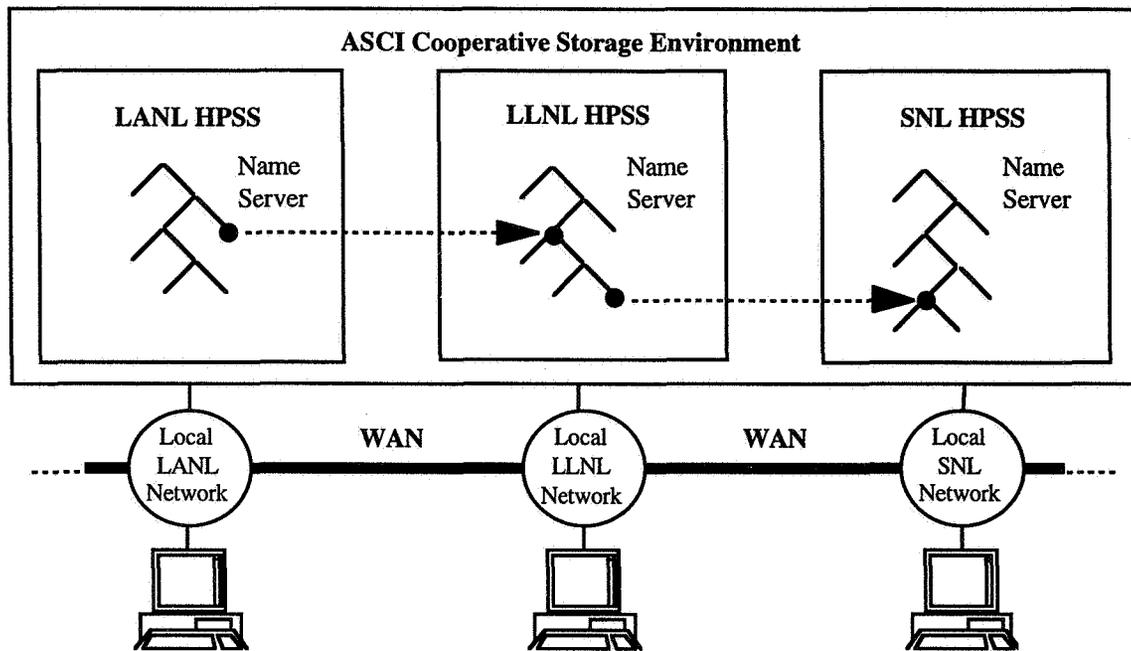


Figure 7. Phase III - Linked HPSS Servers Across Sites

Current Status

We plan to continue the long-standing DOE collaboration to develop HPSS with assistance from ASCI funding. HPSS Release 3, containing over 500 integration and system tests and approximately 200,000 lines of executable code, was released on June 30, 1996. Work continues in 1996 on the next deliveries of HPSS (known as Release 3+ and 4). In 1997, all three ASCI DP sites plan to deploy Release 3+ in production classified and unclassified computing environments. These three systems will initially function independently as described in the Phase I description above. Work on DFS/HPSS integration is on-going to meet the implementation described above as Phase II, as well as work to meet some of the scalability requirements of Phase III. Work is also underway at Lawrence Livermore to provide support in HPSS for the emerging MPI-IO interface standard [19,20] as a basis for parallel I/O portability in ASCI applications.

Conclusions

ASCI is an application-driven program. The fundamental goal of ASCI's Problem Solving Environment program is to give users the tools they need to do their job. This means ensuring that the power of the ASCI application/platform combination can be readily applied to the challenges of science-based stockpile stewardship. This will require the development of a balanced, supporting infrastructure far more capable than any available today. In particular, hierarchical storage management, archival storage, and parallel I/O must scale together with the growth in platform capability.

We have discussed the critical requirements that an ASCI cooperative model places on a storage infrastructure. ASCI will need terascale storage resources to be available from the desktop and viewed as a black-box. Users should not be concerned with how, where, or in

what format data is stored. Several transparencies must be provided to reduce or hide the complexity of system interaction and internal detail, but must be balanced such that efficiency is not lost in resource utilization or performance. There are critical site autonomy, communication, and security issues that must also be considered.

We described an architectural design and phased implementation plan for a distributed, cooperative storage environment (CSE) to achieve the necessary performance, user transparency, site autonomy, communication and security features needed to support ASCI requirements. The CSE provides a logically single, geographically distributed, storage infrastructure of semi-autonomous cooperating sites. The ASCI implementation strategy focuses heavily on integration of the High Performance Storage System (HPSS), a scalable, high-performance system for data-intensive applications and large-scale computers, and OSF/DCE DFS, a popular distributed file system. While DFS satisfies several transparency requirements of ASCI, it lacks the archival back-end and performance needed to support the petabytes of data generated by ASCI. We have shown how we plan to exploit the DMIG DMAPI interface to provide DFS with an appropriate back-end capable of satisfying ASCI's large storage requirements, and a phased implementation approach satisfying the cooperative storage requirements of ASCI.

References

- [1] Accelerated Strategic Computing Initiative (ASCI) Program Plan, Final Draft, DOE Defense Programs, May 1996.
- [2] R. W. Moore, "A Petabyte/Teraflops System: Meeting Future Data Needs," San Diego Supercomputer Center Gather/Scatter, Vol. 11, No. 4, December 1995.
- [3] B. Tierney, W. Johnston, L. Chen, H. Herzog, G. Hoo, G. Jin, J. Lee and D. Rotem, "Distributed Parallel Data Storage Systems: A Scalable Approach to High Speed Image Servers," *Proceedings of ACM Multimedia '94*, October 1994.
- [4] C. Antonelli and P. Honeyman, "Integrating Mass Storage and File Systems," Technical Report 93-2, University of Michigan Center for Information and Technology Integration, Ann Arbor, MI, April 15, 1993.
- [5] J. Nydick, K. Benninger, B. Bosley, J. Ellis, J. Goldick, C. Kirby, M. Levine, C. Maher and M. Mathis, "An AFS-based Mass Storage System at the Pittsburgh Supercomputer Center," *Proceedings of the Eleventh IEEE Symposium on Mass Storage Systems*, Monterey, CA, October 7-10, 1991.
- [6] B. Kobler, J. Berbert, P. Caulk and P. Hariharan, "Architecture and Design of Storage and Data Management for the NASA Earth Observing System Data and Information System (EOSDIS)," *Proceedings of the Fourteenth IEEE Computer Society Mass Storage Systems Symposium*, Monterey, CA, September 11-14, 1995.
- [7] E. Levy and A. Silberschatz, "Distributed File Systems: Concepts and Examples," *ACM Computing Surveys*, Vol. 22, No. 4, December, 1990.

- [8] K. Kazar, B. Leverett, O. Anderson, V. Apostolides, B. Bottos, S. Chutani, C. Everhart, W. Mason, S. Tu and E. Zayas, "DEcorum File System Functional Overview," *Summer USENIX Conference Proceedings*, Anaheim, CA, June 1990.
- [9] Data Management Interfaces Group, "Interface Specification for the Data Management Application Programming Interface," Version 2.3a Draft X/Open Submission, January 1996.
- [10] J. Moffett and M. Sloman, "The Representation of Policies as System Objects," *Proceedings of the Conference on Organizational Computer Systems (COCS '91)*, Atlanta, GA, November 5-8, 1991.
- [11] R. Coyne, H. Hulen, and R. Watson, "The High Performance Storage System," *Proceedings Supercomputing '93*, Portland, OR, November 15-19, 1993.
- [12] D. Teaff, R. Coyne and R. Watson, "The Architecture of the High Performance Storage System (HPSS)," *Fourth NASA GSFC Conference on Mass Storage Systems and Technologies*, College Park, MD, March 28-30, 1995.
- [13] R. Garrison, et al. (eds.), Reference Model for Open Storage Systems Interconnection: Mass Storage Reference Model Version 5, IEEE Storage System Standards Working Group (P1244), September 1994.
- [14] R. Hyer, R. Ruef and R. Watson, "High Performance Direct Network Data Transfers at the National Storage Laboratory," *Proceedings of the Twelfth IEEE Symposium on Mass Storage Systems*, Monterey, CA, April 26-29, 1993.
- [15] R. Watson and R. Coyne, "The Parallel I/O Architecture of the High Performance Storage System," *Proceedings of the Fourteenth IEEE Computer Society Mass Storage Systems Symposium*, Monterey, CA, September 11-14, 1995.
- [16] R. Van Meter, "A Brief Survey of Current Work on Network Attached Peripherals," abstract published in ACM Operating Systems Review, January 96, or available at <http://www.isi.edu/~rdv/netstation/nap-research/nap-extab/nap-extab.html>.
- [17] D. Wiltzius, "Network-attached peripherals (NAP) for HPSS/SIOF." http://www.llnl.gov/liv_comp/siof/siof_nap.html.
- [18] S. Louis, and D. Teaff, "Class of Service in the High Performance Storage System," *Third IFIP TC6 International Conference on Open Distributed Processing*, Brisbane, Australia, February 21-24, 1995.
- [19] T. Jones, R. Mark, J. Martin, J. May, E. Pierce and L. Stanberry, "An MPI-IO Interface to HPSS," *Fifth NASA GSFC Conference on Mass Storage Systems and Technologies*, College Park, MD, September 17-20, 1996.
- [20] MPI-IO Committee, MPI-IO: A Parallel File I/O Interface for MPI, Version 0.5. <http://lovelace.nas.nasa.gov/MPI-IO>.

53-82

83185

An MPI-IO Interface to HPSS¹

**Terry Jones, Richard Mark, Jeanne Martin
John May, Elsie Pierce, Linda Stanberry**
Lawrence Livermore National Laboratory
7000 East Avenue, L-561
Livermore, CA 94550
johnmay@llnl.gov
Tel: 510-423-8102
Fax: 510-422-4293

Abstract

This paper describes an implementation of the proposed MPI-IO [5] standard for parallel I/O. Our system uses third-party transfer to move data over an external network between the processors where it is used and the I/O devices where it resides. Data travels directly from source to destination, without the need for shuffling it among processors or funneling it through a central node. Our distributed server model lets multiple compute nodes share the burden of coordinating data transfers.

The system is built on the High Performance Storage System (HPSS) [2, 12, 14], and a prototype version runs on a Meiko CS-2 parallel computer.

1 Introduction

The Scalable I/O Facility (SIOF) project² is an effort to develop an I/O system for parallel computers that offers both high aggregate bandwidth and the ability to manage very large files [8]. To meet these needs, the SIOF project is developing a hardware infrastructure that will connect the processors in a parallel computer to multiple storage devices through a Fibre Channel network [3]. The project is also developing an application programming interface (API) that will give large scientific codes flexible, efficient access to the I/O system without forcing programmers to manage low-level details. This paper describes the implementation of the SIOF API software.

1.1 Background

Parallel programs use a number of strategies to manage large data sets. Most parallel computers offer a global file system that all the processors can access. Data typically travels over the internal communication network between the compute nodes and one or more I/O nodes, which manage a set of storage devices. This arrangement gives all the nodes access to all the files, but I/O traffic must compete with regular message traffic for

¹ This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under contract number W-7405-Eng-48. Specific funding came from the Gas and Oil Information Infrastructure initiative and the DOE Digital superLab project.

² SIOF should not be confused with a different project that has a similar name, the Scalable I/O Initiative [9].

access to the communication network. Depending on the configuration of the system, the I/O nodes may become a bottleneck to data transfer. Some parallel machines have local disks attached to each compute node, and programs may write separate files from each node to its local disk. This approach offers high aggregate bandwidth, since nodes transfer data through separate, dedicated channels. However, merging the data in these separate files or reading the files on a different set of nodes can be inconvenient.

The SIOF architecture uses a separate I/O network to connect each compute node to the storage devices. This allows I/O to proceed in parallel at a high aggregate bandwidth. It also lets a program treat data distributed over multiple devices as a single logical file. The system supports third-party transfers, so one compute node can orchestrate data transfers between a file and several processors. SIOF uses the High Performance Storage System (HPSS) [2, 12, 14] to manage distributed files and third-party transfers. HPSS is a joint development project of IBM and several U.S. national laboratories and supercomputer centers. Since the HPSS API is designed mainly for shared-memory systems, and since it requires programmers to specify many details of a parallel transfer, the SIOF project is developing a separate API for message-passing systems. This API is based on the proposed MPI-IO standard [5], which in turn is based on the popular MPI (Message-Passing Interface) standard [10]. Our initial implementation of the SIOF hardware and software runs on a Meiko CS-2 parallel computer.

1.2 The SIOF application programming interface

Several MPI-IO development efforts are now underway; however because the architecture of our underlying I/O system is unique, the SIOF implementation has some noteworthy features:

- Control of access to a given open file is centralized, but data transfer is distributed and direct.
- A *distributed server model* spreads the burden of controlling different open files among multiple compute nodes.
- When multiple processes participate in *collective* read or write operations, the system can determine dynamically how to group these requests for high throughput based on the transfer size, the distribution of data among the compute nodes and I/O devices, and other parameters.

The next two sections give the background to SIOF, HPSS, and MPI-IO. Section 4 examines the architecture of our MPI-IO implementation, and Section 5 describes how we manage collective I/O requests. Section 6 reports the current status of the project and our future plans for it. We conclude in Section 7 with a summary of the SIOF API.

2 The Scalable I/O Facility and HPSS

The SIOF project goal is to provide a “network-centered, scalable storage system that supports parallel I/O” [8]. To this end, SIOF is collaborating with HPSS developers to

extend the HPSS environment in two areas. First, it is addressing some issues (such as protocols and security) pertaining to network-attached peripherals [15]. Second, it is providing an MPI-IO interface to the HPSS client API, as described in the sections that follow.

The SIOF implementation uses a crosspoint-switched FC fabric that will connect the processors of a Meiko CS-2 directly to disk arrays, parallel tapes, and frame buffers. Each compute node is capable of independent I/O across the FC fabric so that all nodes may perform I/O in parallel. The SIOF API orchestrates coordinated accesses across the processors in a distributed computation, with each processor working on a part of a file. The envisioned architecture is shown in Figure 1.

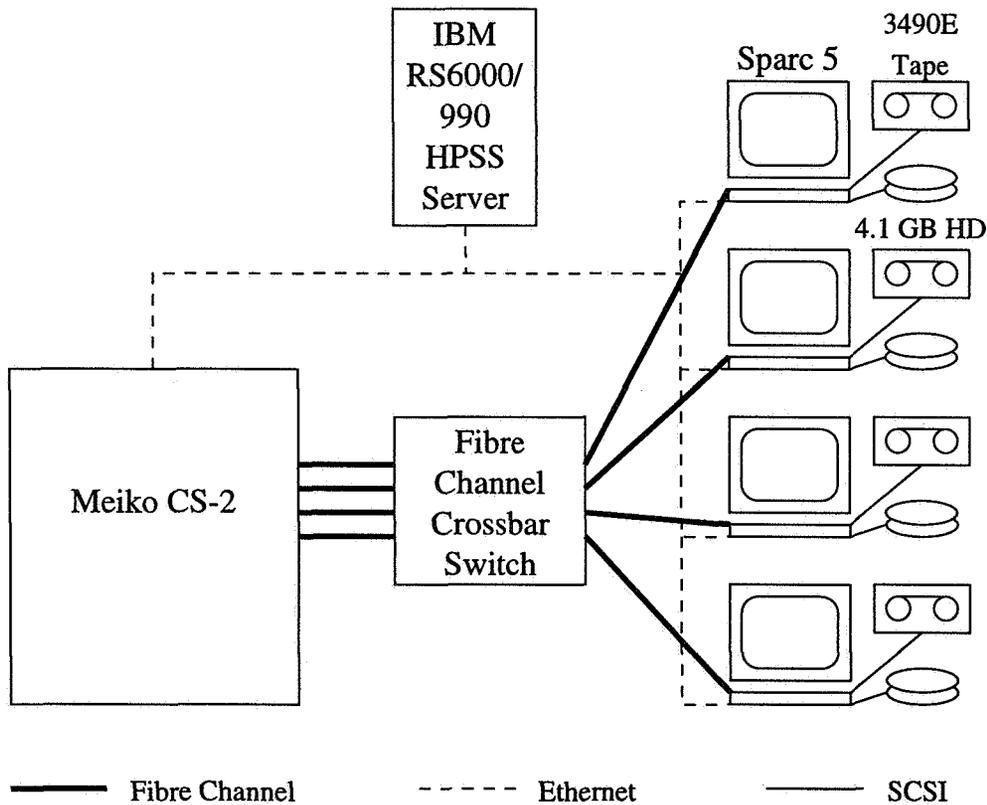


Figure 1: The Scalable I/O Facility architecture consists of a Meiko CS-2, an IBM RS6000 HPSS server, and a collection of tape drives and disk arrays. A Fibre Channel network connects the storage devices to the Meiko compute nodes.

The SIOF extensions rely on HPSS to achieve this implementation. HPSS is a standards-based, modular, hierarchical storage system that supports scalability in a variety of parallel environments. HPSS consists of multiple layers of interfaces that provide secure parallel access to the files of a storage system. The interfaces are implemented using DCE (Distributed Computing Environment) [6] and Transarc's Encina [13] transaction-based remote procedure calls.

The higher-level interfaces implement the administration (e.g., naming) and security (e.g., access control) of the storage system, while the lower-level interfaces implement the mechanics of the parallel data transfers required by file access commands. The interfaces of particular interest to the SIOF are the *data movers*.

For any given data transfer there is a mover on the application (client) side, and a corresponding mover on the HPSS side. These two movers determine the details of a given data transfer from a data structure called an IOD (I/O descriptor). This descriptor treats a data transfer as a mapping from the source of the transfer into a data transfer stream and a corresponding mapping from the transfer stream into the destination or sink of the transfer (see Figure 2).

In the simplest case, both the source and sink of the transfer are one contiguous block of bytes. But with distributed files and distributed applications, blocks may be discontinuous at either the source or the destination. The descriptive flexibility of the HPSS IOD allows a single transfer to consist of bytes striped across multiple nodes, multiple storage devices, or both. In each IOD, the transfer stream mappings from the source and to the sink are represented by a list of one or more descriptors, where each entry on the list describes a contiguous block of bytes.

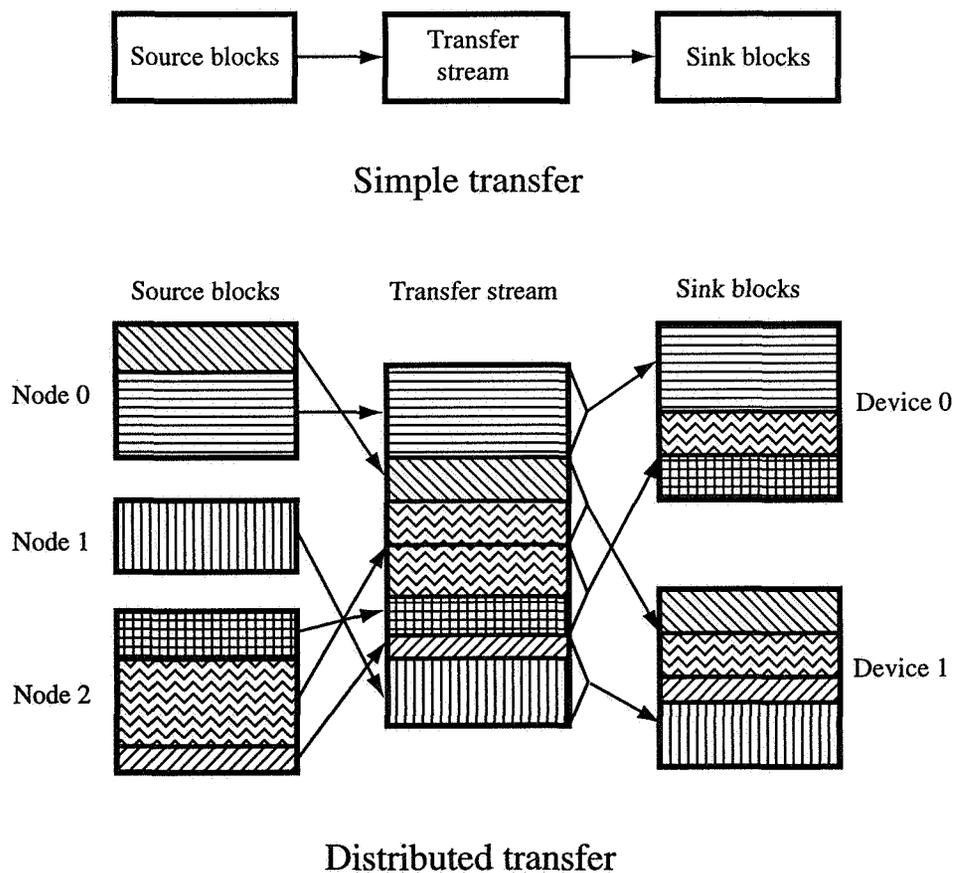


Figure 2: In a simple transfer, is mapped from a source into a transfer stream and then to a sink with no reordering. In a distributed transfer, HPSS can move blocks of any size from source to sink in any order or interleaving.

To initiate a simple contiguous transfer, a client application can call HPSS through Unix-like **read** and **write** commands. These interfaces construct an IOD for the transfer automatically. More complex transfers require the application to construct its own IOD with the necessary mappings and to deliver this IOD to HPSS through **readlist** and **writelist** commands.

Application programs using the SIOF API (rather than the HPSS API) do not create IODs directly. Instead, the SIOF API code creates IODs using the MPI datatypes (see Section 3) specified in MPI-IO **open**, **read**, and **write** operations. The motivation for hiding the details of the IOD construction is twofold: the client application can use the simpler MPI-like interface to describe the transfer, and the new interface layer introduces the possibility of optimizing transfers.

The HPSS architecture is also designed to allow for potential optimizations as a data transfer request is processed. The higher-level servers rewrite the client IOD and dispense one or more IODs to HPSS movers to complete a given transfer. In the process, HPSS uses its knowledge of how the file being accessed is distributed across storage devices, and which HPSS movers control those devices, to break the client IOD into component IODs to be distributed to the HPSS movers. Although HPSS does not currently reorder or combine client IODs, its design allows this in the future.

Note that although there is one logical mover for the client and another for HPSS for each transfer, there may in fact be multiple movers active on both sides: one per participating node on the client side, and one per device on the HPSS side. These movers are threads that are spawned when a transfer begins, and they terminate when the transfer is complete.

As shown above, the architecture of HPSS allows full generality of how source blocks are mapped into a data transfer and thence into sink blocks, but it is expected that the best performance will always be achieved when source and sink blocks match in size and number exactly. This will reduce contention (e.g., where more than one node is attempting to access the same device) and allow maximum parallelization of the transfer.

3 MPI-IO

In choosing an interface for SIOF that would work well in message passing programs, we examined several parallel file systems. We selected MPI-IO because it offers a good range of parallel I/O features and because it appears to have a good chance of becoming a widely-implemented standard.

MPI-IO was first developed by a group of researchers independently of the MPI Forum. Although the work used many ideas from MPI, it was not a formal part of the standard. However in April 1996, the MPI Forum voted to begin work on an I/O chapter, and the MPI-IO standard was presented as the initial proposal. As a result, while MPI-IO is now more likely to become a standard, its interface may change significantly as more people

contribute to it. In this paper, we refer to the version of MPI-IO that existed before the MPI Forum made any changes.

Like most parallel I/O libraries, MPI-IO supports *collective requests*, where multiple processes participate in an operation such as reading or writing a file. In many cases, an I/O system can gather collective requests from multiple nodes into a single I/O request. It can often complete this joint request more efficiently than a group of independent requests. In some implementations of parallel I/O systems, collective requests may perform an implicit barrier synchronization on the participating nodes. The synchronization allows a server to collect data from all the nodes participating in the operation before completing the operation. However, the MPI-IO standard does not require synchronization and warns users not to depend on it.

In a parallel environment, multiple processes can access a file simultaneously. Parallel processes often make interleaved accesses, and they may also access separate portions of the file. Some parallel file systems have an interface that is based on the POSIX [7] standard for file I/O, but this interface is designed for an environment where files are not shared by multiple processes at once (with the exception of pipes and their restricted access possibilities) [11]. Furthermore, POSIX file operations do not allow access to multiple discontinuous parts of the file in a single operation.

MPI uses user-defined and built-in datatypes to describe how data is laid out in a memory buffer. In MPI-IO, datatypes used in this way are called *buffer types*. MPI-IO also uses MPI datatypes to describe the partitioning of file data among processes. A *file type* describes a data pattern that can be repeated throughout the file or part of the file. A process can only access the file data that matches items in its file type. Data in areas not described by a process' file type (holes) can be accessed by other processes that use complementary file types.

MPI associates a datatype with each message. The length of the message is an integral number of occurrences of the datatype. This method of defining a message is more portable than specifying the message length in bytes. Similarly, MPI-IO defines a third datatype called an *elementary type* or *etype*. Both the buffer type and the file type contain an integral number of e-types. This allows offsets into a file to be expressed in e-types rather than bytes. Using MPI datatypes has the advantage of added flexibility and expressiveness, as well as portability.

4 SIOF API architecture

Having chosen MPI-IO as our application programming interface, we designed our implementation with several goals in mind:

- Make efficient use of I/O resources, including the storage devices, the external network, the processing nodes, and HPSS.
- Avoid creating bottlenecks that would limit the scalability of the I/O system.

- Minimize barrier synchronizations among the processes of the application, since these can slow down operation and a risk of deadlock.

This section describes how we designed the SIOF API to achieve these goals.

We consider an application to have one process per node, and in this description we assume that all of the application code executes in one thread per process. The SIOF API spawns several additional threads in each process that share its address space.

The thread executing the application is called the client thread, and each process spawns a server thread when the API is initialized. The client thread includes code that implements the interfaces of the MPI-IO functions. The server thread executes the code that issues requests to HPSS.

One server thread manages a given open file on behalf of all the nodes, and servers on different processors can manage different open files. This prevents any single node from having to manage all the open files. We call this aspect of the architecture the *distributed server model*. Conceptually, the server and client threads could be separate processes, since they share no data structures. However MPI cannot at present direct messages to different processes on the same node, so using MPI for communication requires the server and client to reside in the same process.

Any time a client thread needs to operate on a file, it sends a request via MPI to the server thread on the appropriate node. Each server maintains a table of the open files it manages. When a request arrives, the server looks up the HPSS file descriptor and other information about the file and then spawns a driver thread to issue the HPSS request. When this request is complete, the driver thread sends a response message to the client and then terminates. The client thread receives the message, and the original MPI-IO call returns a result to the application program.

4.1 Opening and closing a file

Opening a file in MPI-IO is always a collective operation, which means that all the nodes in the program (or a specific subset of them) participate. The nodes select a server by hashing the file name and other parameters to the **open** call to produce a node number. Since all the nodes must specify the same parameters to the call, they will all select the same node without needing to communicate with each other. The server's node number is stored in a local file table on each node for use in future requests.

Each node sends a request to the server as soon as it is ready; there is no barrier synchronization upon opening a file. When the server receives the first **open** request for a given file, it creates an entry in the file table and spawns a driver thread to call HPSS. Subsequent requests from other nodes to open the same file will find a matching request in the file table. If the HPSS **open** call has already completed, the server will send a reply containing the data from the completed (or possibly failed) call. If the HPSS call is still pending, the new request will be placed in a queue. As soon as the driver thread completes the HPSS call, it will send responses to the nodes with queued requests. This

arrangement guarantees that each **open** request generates exactly one call to HPSS, and requests from other nodes to open the same file share the results of this call.

Closing a file is also a collective operation, and the nodes again send individual requests to the server. This time, however, the server delays closing the HPSS file until all the requests have arrived. Therefore, closing a file is a synchronizing operation. This is necessary because the file cannot be closed until all the nodes are finished with it, and any errors that occur when HPSS closes the file must be reported to all participating nodes. Moreover, if the close operation does not synchronize, a node might treat a file as if it were closed and its buffers flushed when the file is in fact still open and handling requests from other nodes.

4.2 Reading and writing

Programs can read and write files collectively or independently, and they can intermix these operations freely on the same file (provided that all nodes that open a file participate in the collective operations).

For an independent read or write operation, the client first spawns a mover thread that will copy data between the memory buffer and the network channel to the storage device. When this thread has started, the client sends a read or write request to the server. The request includes the information that the server will need to construct an HPSS IOD (see Section 2). The server spawns a driver thread to issue the HPSS **readlist** or **writelist** call. HPSS transfers the data directly between the node and the storage device and then returns from the **readlist** or **writelist** call. Part of the return data is a structure called an IOR (I/O reply), which the driver thread sends back to the mover before terminating. The driver thread the IOR to its own record of the transfer, then returns status information to the client thread and terminates. The SIOF API code in the client thread transforms the status information into MPI-IO return data before finally returning from the MPI-IO call.

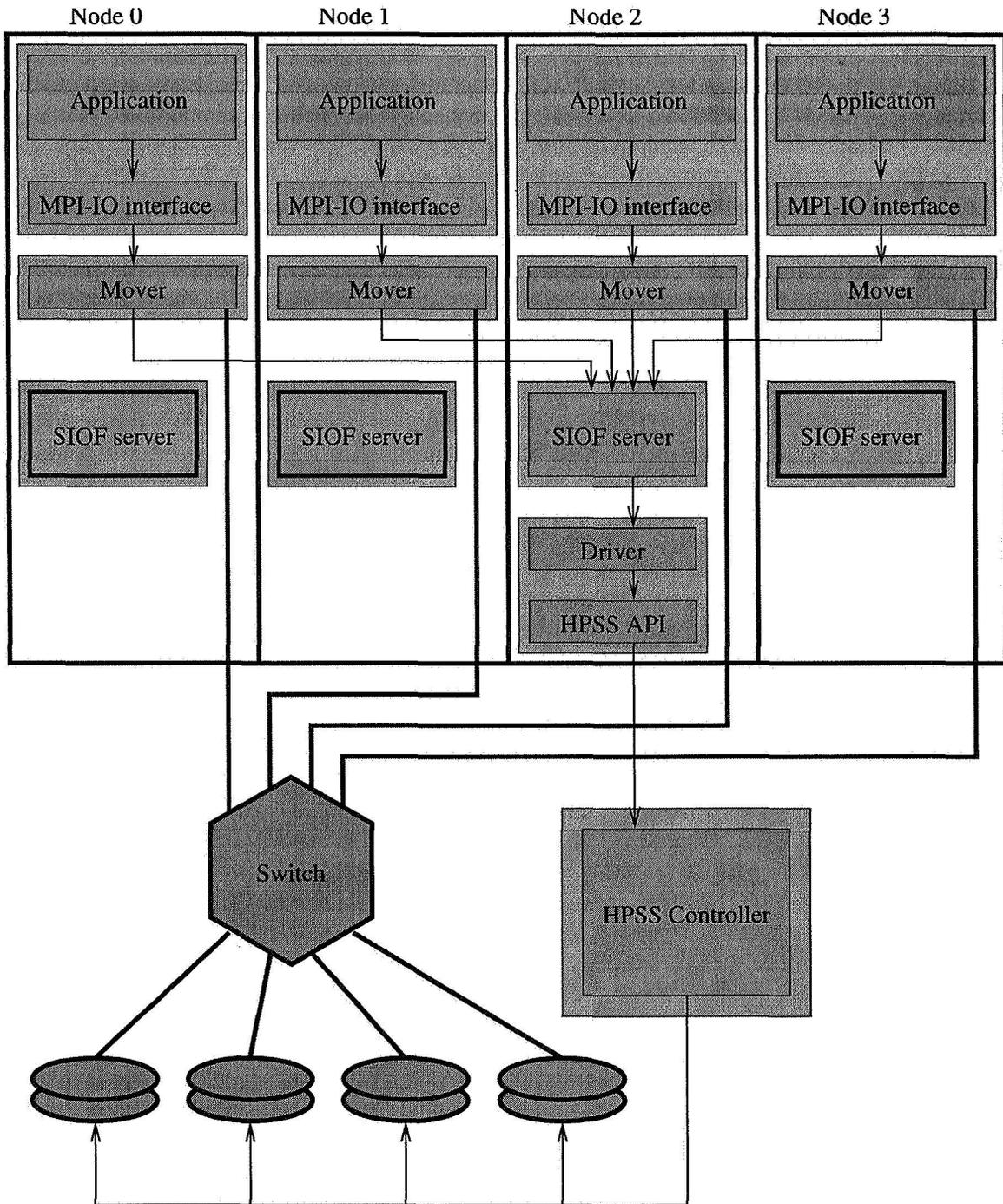


Figure 3: The SIOF API is implemented in several threads on each node (shown here as four large, vertical rectangles). Outer shaded boxes represent threads; inner boxes are functional modules within a thread. Boxes with heavy outlines show servers not participating in an operation. For read and write operations, control is centralized at a server thread, but data travels through separate, high-bandwidth channels between devices and compute nodes.

Collective operations require a few extra steps. The details appear in Section 5, but the main difference from independent operations is that the server may gather up several requests from different nodes and issue them together in a single HPSS call, as depicted in Figure 3.

4.3 File types and buffer types

Section 3 noted that MPI-IO programs can use file types and buffer types to access discontinuous regions of data. MPI-IO translates these datatypes into an internal format called a *chunk map*. A chunk map is a list of contiguous data blocks, and it contains only the information that the SIOF API needs from an MPI datatype to construct an IOD.

Because MPI specifies no functions for accessing the layout information in a datatype, the SIOF API code must explicitly read the internal data structures of the MPI implementation on which it is based (MPICH [1]). One reason for using chunk maps is to isolate the system-dependent code as much as possible, so most of the SIOF API code works with chunk maps rather than MPI datatype structures.

The SIOF API stores the chunk map of the file type for each node and each open file in the server thread's file table. When a file is read or written, the server constructs an HPSS IOD for the data to be transferred, with source and sink mappings for each contiguous chunk of data to be accessed. It passes this IOD to a single HPSS call.

Meanwhile, the mover thread parses the chunk map corresponding to its node's buffer type to determine which data to access in memory. The SIOF API does not compare buffer types with file types or decompose them with respect to each other; HPSS and the client mover thread can each behave as if the other is accessing a single, contiguous stream of data.

5 Managing collective operations

The SIOF API currently supports four types of data access: the independent **read** and **write** operations, and collective versions called **read-all** and **write-all**. Structuring the server to permit collective operations on reads and writes requires that several issues be addressed:

- How are collective operations implemented?
- How is the decision made to dispatch them?
- What optimizations are available for collective operations?

This section discusses these three issues.

5.1 Collective implementation

A group of client nodes initiates a collective operation by sending requests to the server. The server queues these requests as they arrive in a *data access list*, which it traverses either after the receipt of a client's message for a file operation or after a predetermined period of time has elapsed, whichever comes first. As the server traverses the list, it updates a *dispatch priority* for each pending collective operation. The dispatch priority determines when the server will initiate the data access; if the priority is over a predetermined threshold, the server spawns a thread to issue the HPSS **readlist** or **writelist** call. If a collective write request includes overlapping file accesses by different nodes, the server constructs an IOD that resolves the conflict in a well-defined way.

The data access list also records the number of outstanding clients, which is needed to handle cases where the server dispatches a request before all clients have checked in. The number of clients is initially the number of nodes that have jointly opened the same file, but if two or more dispatches are used for the same operation, it will be the number of clients remaining for the operation (i.e., the number not already checked in and previously dispatched).

5.2 Determining dispatch priority

How the dispatch priority is determined will have a strong effect on performance and utilization of the I/O system. For example, one can imagine a scenario in which 15 clients of a 16-client application check in at nearly the same time, but the 16th client checks in much later. In such a scenario, it may be more efficient to dispatch the 15 pending requests as a group and wait for the 16th request separately. On the other hand, issuing a request too soon will reduce the ability of the SIOF API library to amortize latency costs involved in setting up a data access. A number of factors may play a part in determining the dispatch priority. At the present, our implementation for MPI-IO **read-all** and **write-all** operations blocks until all client nodes have checked in. However, we plan to investigate several algorithms to determine their effect on utilization and performance. These algorithms will consider, to varying degrees, the time since a request was first issued, information on which clients have checked in, the transfer size, the granularity of the file types, and whether the access is to tape or disk.

5.3 Optimizations

The architecture of the SIOF API makes several optimizations feasible. These include:

- Asynchronous operation.
- Grouping accesses on the same storage device.
- Grouping accesses on the same processor.
- Coalescing small accesses.

The first optimization reduces a server's sensitivity to the latency of HPSS calls. By spawning a thread for each such call, the server can handle multiple requests concurrently. (However independent requests for access to the same file are serialized to preserve atomicity.)

Grouping accesses to the same storage device can help improve cache performance. For example, certain decompositions of matrices among processors can produce requests for small, interleaved chunks of data [4]. By constructing IODs so that requests for sequential data appear in order, the server can increase the probability of cache hits on a disk. On the other hand, sending small blocks of data between one disk and multiple nodes in round-robin order may produce excessive switching latency in the external network. In that case, it may be better to group requests so that data residing on one node is accessed sequentially. Performance tuning will help us determine how best to arrange the parts of a collective request.

Even if there is no locality to be exploited in a collective operation, grouping requests from multiple nodes into a single **readlist** or **writelist** call can amortize one-time expenses incurred in I/O operations, such as the cost of an RPC transaction between the parallel computer and the HPSS controller.

6 Current status and future work

We demonstrated a prototype of the SIOF API at the Supercomputing '95 conference. This version included independent and collective reading and writing with independent file pointer and explicit file offsets. The prototype supported file types and buffer types using any MPI datatype. We are currently working on a production quality version of the SIOF API that will implement the full MPI-IO specification. We plan to include this interface in a future release of HPSS.

Meanwhile, the MPI-IO specification itself is changing. The MPI Forum continues to work on a standard for parallel I/O, and we plan to incorporate whatever changes they make to MPI-IO.

In addition to the interface changes, we are also considering structural changes to the SIOF API that will improve its efficiency and robustness. The robustness changes will mainly involve implementing the MPI exception handling mechanism.

To improve efficiency, we would like to reduce the amount of processing that the server threads do. For example, we are considering whether independent read and write requests could be sent directly from the requesting node to the HPSS server instead of being routed through the server node for the open file. Implementing this change would require finding a way for multiple nodes to access the same HPSS file correctly and efficiently. Another change we are considering is to use the **MPI_Reduce** command to combine requests from multiple nodes for a single collective operation. At present, when a group of nodes issues a collective write request, for example, the server receives messages from each requesting node and builds up a description of the operation incrementally. As an alternative, **MPI_Reduce** could use a customized combiner function to distribute the work of creating a collective request over many nodes. The

server node would then receive a single, merged request instead of many separate parts. The reduction in memory management and message handing activity on the server node would be significant, but this approach would cause collective operations to synchronize the nodes. For loosely-synchronous programs, the synchronization delays could outweigh other gains. Therefore, we will have to consider carefully the merits of implementing this strategy.

7 Conclusion

The SIOF API is a new implementation of the proposed MPI-IO standard. It is designed as a high-level user interface for the HPSS file system, and its initial implementation is on a Meiko CS-2 parallel computer. Because HPSS supports third-party transfers over an external network, our implementation can transfer data in parallel between processors and storage devices while presenting a global view of the file system that all nodes can access. Our distributed server model spreads the burden of coordinating data transfers over multiple nodes. Control of a given open file is centralized, but data transfer can proceed in parallel. We believe this combination of features will offer the high aggregate I/O bandwidth for large data transfers that many parallel scientific codes need.

References

- [1] P. Bridges, N. Doss, W. Gropp, E. Karrells, E. Lusk and A. Skjellum, Users' Guide to MPICH, A Portable Implementation of MPI. <http://www.mcs.anl.gov/mpi/mpi/mpiuserguide/paper.html>.
- [2] R. Coyne, H. Hulen, and R. Watson, The High Performance Storage System, in *Proceedings of Supercomputing '93*, November 1993.
3. Fibre Channel Association, *Fibre Channel: Connection to the Future*, Austin, Texas, 1994.
- [4] D. Kotz and N. Nieuwejaar, Dynamic file-access characteristics of a production parallel scientific workload, in *Proceedings of Supercomputing '94*, November 1994.
- [5] MPI-IO Committee, MPI-IO: A Parallel File I/O Interface for MPI, Version 0.5. <http://lovelace.nas.nasa.gov/MPI-IO>.
- [6] Open Software Foundation, *OSF DCE Application Development Reference*, Prentice-Hall, Englewood Cliffs, N.J., 1993.
- [7] *Portable Operating System Interface (POSIX)—Part 1: System application programming interface (API)*. IEEE Standard 1003.1-1990.
- [8] Scalable I/O Facility. http://www.llnl.gov/liv_comp/siof/siof.html.
- [9] Scalable I/O Initiative. <http://www.cacr.caltech.edu/SIO/>.

- [10] M. Snir, S. W. Otto, S. Huss-Lederman, D. W. Walker, and J. Dongarra, *MPI: The Complete Reference*, MIT Press, Cambridge, Mass., 1996.
- [11] W. R. Stevens, *Unix Network Programming*, Prentice-Hall, Englewood Cliffs, N.J., 1990.
- [12] D. Teaff, R. W. Watson, and R. A. Coyne, The architecture of the High Performance Storage System (HPSS), in *Proceedings of the Goddard Conference on Mass Storage and Technologies*, March 1995.
- [13] Transarc Corporation, Encina product information. <http://www.encina.com/Public/ProdServ/Product/Encina>.
- [14] R. Watson and R. Coyne, The parallel I/O architecture of the High Performance Storage System (HPSS), in *IEEE Symposium on Mass Storage Systems*, September 1995.
- [15] D. Wiltzius, Network-attached peripherals (NAP) for HPSS/SIOF. http://www.llnl.gov/liv_comp/siof/siof_nap.html.

A Proposed Application Programming Interface for a Physical Volume Repository

Merritt Jones
The MITRE Corporation
1820 Dolley Madison Blvd.
McLean VA 22102
MERRITT@mitre.org (Merritt E Jones)
Tel: 703-883-5471
Fax: 703-883-5230

Joel Williams
Systems Engineering and Security
7474 Greenway Center Drive
Greenbelt MD 20770
joel.williams@ses-inc.com (Joel Williams)
Tel: 301-441-3694
Fax: 301-441-3697

Richard Wrenn
Digital Equipment Corporation
Colorado Springs CO 80919
wrenn@cookie.enet.dec.com (Rich Wrenn)
Tel: 719-548-2887
Fax: 719-548-6330

Introduction

The IEEE Storage System Standards Working Group (SSSWG) has developed the *Reference Model for Open Storage Systems Interconnection, Mass Storage System Reference Model Version 5*. This document, dated September 8, 1994, provides the framework for a series of standards for application and user interfaces to open storage systems. More recently, the SSSWG has been developing Application Programming Interfaces (APIs) for the individual components defined by the model. The API for the Physical Volume Repository is the most fully developed, but work is being done on APIs for the Physical Volume Library and for the Mover also. The SSSWG meets every other month, and meetings are open to all interested parties. Further information on the SSSWG may be found at <http://www.arl.mil/IEEE/ssswg.html>.

The Physical Volume Repository (PVR) is responsible for managing the storage of removable media cartridges and for mounting and dismounting these cartridges onto drives. This document describes a model which defines a Physical Volume Repository, and gives a brief summary of the Application Programming Interface (API) which the IEEE Storage Systems Standards Working Group (SSSWG) is proposing as the standard interface for the PVR.

The IEEE Reference Model

The Reference Model is described in [1]. What follows is a very brief overview.

The Reference Model determines the following key definitions:

- *Store*: an addressable storage space, either physical or virtual
 - Physical attributes defined by the media type
 - Virtual attributes defined by the client request
- *Device*: A set of media access points (for data access) and mount points (for physical access).
- *Physical Volume*: The recording medium accessible without intervening load operations.
- *Cartridge*: A set of physical volumes or cartridges.

It also defines the following modules

- *Physical Volume Repository (PVR)*: It sees cartridges and drive mount points, and its major operation is to *mount* cartridges.
- *Physical Volume Library (PVL)*: It makes the volume to cartridge mapping and causes the PVR to mount cartridges. Its major operation is to *mount* physical volumes.
- *Virtual Storage Server (VSS)*: It creates virtual stores and performs the store to volume mapping. Its major operation is to create and manage virtual stores.
- *Mover (MVR)*: It manages data transfer and is designed in particular to manage high-speed data transfer. Its major operation is to *load* media to media access points and to perform data transfer.

Figure 1 depicts the IEEE Reference Model

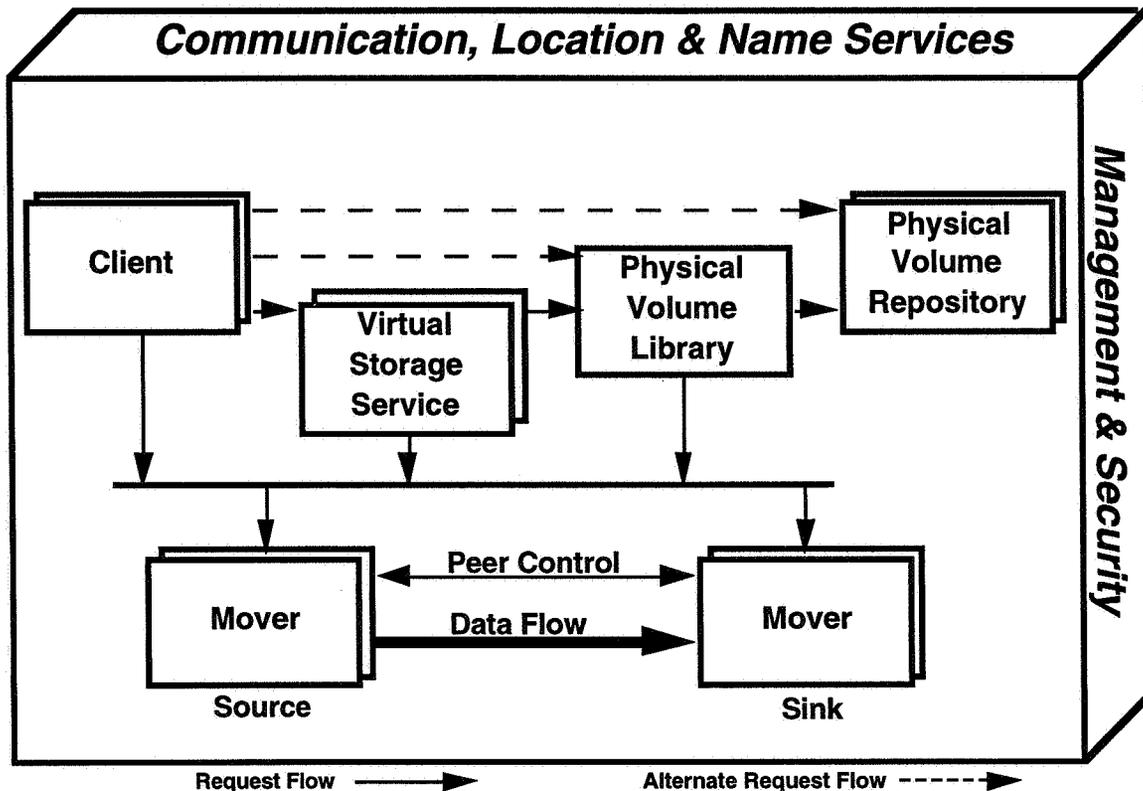


Figure 1

The PVR System

The PVR model and API define a consistent interface that client applications can use to interact with a variety of media-handling systems. The model and API that are described in this paper are based on a paper written by Rich Wrenn [2]. The PVR services are realized by servers which exist for each PVR within an enterprise. That is, the PVR server for various vendors' robotic systems, and humans performing manual operations, will be implemented uniquely. However, with a standard interface implemented, the interface presented to the client applications and the functions provided are exactly the same, regardless of the server implementation. According to this standards-based model, multiple PVR servers can exist on a single node, within a cluster, or within the enterprise. It is also possible for multiple client applications to share a PVR provided within the node, cluster, and enterprise.

Definition of a PVR

The physical volume repository is defined in the following way.

PVR Objects:

- One or more storage locations organized into partition objects
- Zero or more cartridge objects
- One or more media location domains for grouping cartridges

- Zero or more drive objects
- Zero or more device location domains for grouping drives or dependent PVRs
- Zero or more area objects for staging cartridges prior to mounts
- Zero or more port objects for ejecting cartridges from or injecting cartridges into the PVR
- One or more transfer mechanisms, either mechanical or human, operating in the same context, and capable of moving cartridges between their storage locations, drives, and ports
- One or more controllers which command the transfer mechanism within a single context

PVR Operations

- Mounting cartridges on drives and dismounting cartridges from drives
- Injecting and ejecting cartridges through ports. The PVR provides a controlled interface which defines directives that perform the above operations, and which allow object attributes to be accessed and modified.

Note that PVR does not access the drive data path and has no knowledge of the contents of cartridges.

Definition of the PVR Objects

Figure 1 depicts the relationships between the objects of the PVR. Child objects of the PVR are top level partitions, ports, drives, device-location domains, media-location domains, and cartridges. Partitions are arranged in a hierarchy, with the lowest level partition containing cartridges.

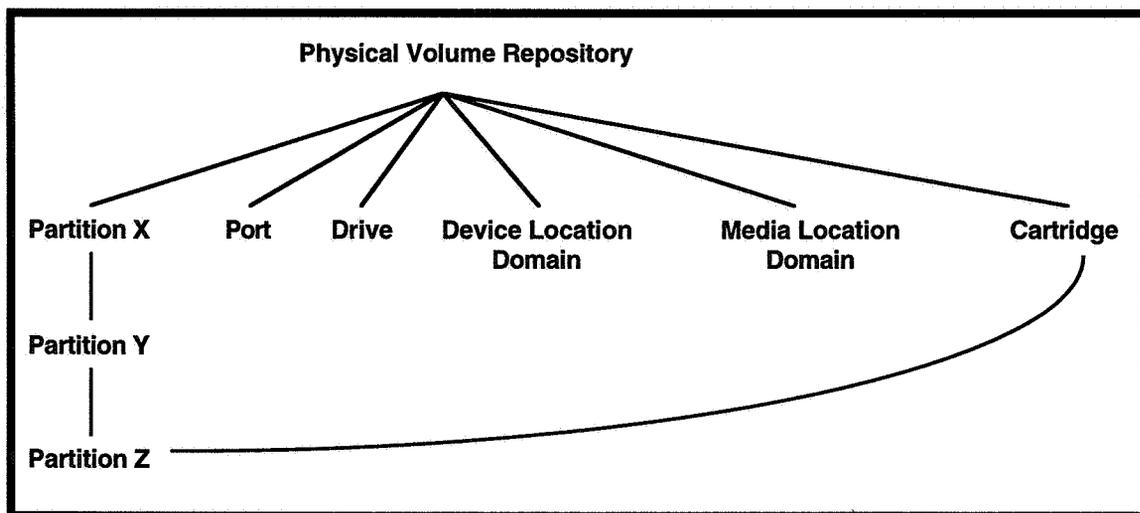


Figure 2

Magazine loaders are modeled through the concept of dependent and independent PVRs, so that a device into which cartridges may be mounted may be either a drive or a dependent PVR.

PVRs which are not serviced by other PVRs are defined as independent PVRs. An independent PVR is serviced by a tender who is responsible for transferring cartridges between the PVR's ports and the outside world. The tender must be able to remove cartridges which have been ejected through an outport and enter cartridges which have been injected through an inport.

PVRs which are serviced by other PVRs are defined as dependent PVRs. Dependent PVRs are not serviced by tenders. A dependent PVR has, by definition, one cartridge slot which can be mounted by another PVR. This cartridge slot is the location at which a compound cartridge (magazine) is mounted and dismounted.

The Partition Object

A partition describes the physical characteristics of a collection of media storage locations. It has finite capacity and stores media of exactly one cartridge type. The PVR cartridge storage locations are physically grouped and have equal characteristics with regard to location, cartridge type, and disaster protection. A top-level partition is a child of one, and only one PVR. Partitions may be created in a hierarchical fashion with a partition having a number of child partitions. A partition may contain one or more MLDs which may contain PVR cartridges.

The Media Location Domain (MLD)

The media location domain describes a logically grouped set of PVR cartridges with equivalent attributes located under a single top-level partition. It has finite capacity and stores PVR cartridges of exactly one type.

The Device Location Domain (DLD)

A device location domain (DLD) is a logical collection of PVR drives or dependent PVRs which have equal static attributes and may be used interchangeably. DLDs are characterized by connectivity, location, management policies, media types supported, and disaster protection. A DLD is a member of one and only one PVR.

The PVR Cartridge

The PVR cartridge is the PVR's view of the cartridge, which, as noted above, does not include any knowledge of the contents of the cartridge. Each cartridge resides in an MLD within a PVR and is referenced by PVR cartridge name. The PVR catalogs the exact physical location used to store the cartridge within a partition.

The PVR Drive

The drive is the PVR's view of the drive--essentially a place where it mounts and dismounts cartridges. A PVR drive name identifies the drives within a DLD. Each drive object resides in a DLD within a PVR. A PVR drive is a member of one and only one DLD.

The Area Object

An area is an optional, temporary location used to optimize mounting of cartridges into devices. Two types of areas are defined: stage areas and scratch areas. Cartridges are moved from their home location in the MLD to an area with either a stage or scratch directive in anticipation of a subsequent mount or mount scratch directive, respectively. This has the potential to reduce the apparent mount time of the PVR cartridge.

The Port Object

Ports are the physical portals through which cartridges are inserted and removed from PVRs. PVR ports are only associated with independent PVRs and are associated with globally named stations such that they are visible to external services, such as a transportation service. A single physical portal may function both as an inport and an outport.

The Task Object

PVR directives cause the instantiation of tasks. All tasks operate asynchronously from the thread which issued the directive, and there is a directive that allows for that thread to wait for the completion of the task. Directives are also provided which allow one to check on the status of a task or to cancel it.

Object Status

In general, each PVR object may be in one of three states:

- *Enabled*: The object is available for use.
- *Disabled*: The object is not available for use.
- *Error*: The object is in error.

Routines exist to *enable* or *disable* objects, and when an object is to be reconfigured or deleted, it must first be disabled.

Basic Directives

All objects (except cartridge and task objects) have four basic directives: *create*, *delete*, *set*, and *show*.

Create creates the object and specifies some of its attributes; *Delete* deletes the object, providing it is in the *disabled* state, and contains no references to other objects; *Set* changes or initializes a list of the object's attributes; *Show* displays a list of the object's attributes.

Cartridges have *set* and *show* directives, but no *create* and *delete* attributes. The *enter* and *inject* directives, which introduce a cartridge into its proper place in the PVR, correspond to the *create* directives for other objects, while the *eject* directive, which ejects the cartridge from the PVR, corresponds to the *delete* directives for other objects.

Tasks are created when a directive is issued which causes a task to begin.

Access Control

In the full implementation of the IEEE Reference Model, the primary clients of the PVR would be Storage Management and the Physical Volume Library.

The PVR server enforces access control through access control lists for all of its objects. The PVR, media location domain, device location domain, and task objects each have access control lists (ACL). Access control for each of the partition and PVR port objects is governed by the access control list on the PVR. Access control for each of the PVR cartridge and PVR area objects is governed by the access control list on the associated media location domain, and access control for each of the PVR drive objects is governed by the access control list on the associated device location domain.

Sharing the PVR by Non-Cooperating Clients

The PVR cartridges within an MLD have equivalent attributes, and are stored in locations such that all locations within the MLD yield equivalent cartridge attributes. Similarly, the devices within a DLD have equivalent attributes and reside in locations such that all locations within the DLD yield equivalent device attributes. MLDs, as well as DLDs, are mutually exclusive (non-overlapping).

The MLD and DLD concepts greatly simplify mount operations. In addition, the access lists on the MLDs and DLDs may be used to partition the PVR for use by non-cooperating clients. Each client would only see its collection of MLDs and DLDs, so that contention for drives and cartridges could not occur, and the possibility of deadlock is eliminated. The robot or human operator, however would be shared because it operates within a single context.

Access Control Rights

Within each of the access control lists, the following list of seven basic rights is used, with additional object specific rights defined as needed.

- *Show* is used for the sole purpose of limiting who can show the instance's attributes.
- *Set* is used primarily to control who can change or initialize attributes of instances governed by the access control list.
- *Control* is used for the sole purpose of limiting who can change the instance's access control list and other attributes (such as owner) which control access to the object.
- *Delete* is used primarily to control who can delete instances governed by the access control list.
- *Execute* is used for several purposes relating to action directives issued to the instance.
- *Read* is used to control the ability to read data contained in the instance.
- *Write* is used to control the ability to write data on the instance. It is also used to control who can change the membership of the instance and who can create subordinate objects of the instance.

Device Selection

The client applications can make three types of mount requests to the PVR:

- Explicit Device Selection (EDS): The client application directs the PVR to mount the PVR cartridge object onto a specific device.
- Automatic Device Selection (ADS): The client application directs the PVR to mount the PVR cartridge onto any device within a device location domain (DLD). The PVR selects a device and mounts the PVR cartridge onto that device. Finally, the PVR replies to the client application with the name of the device that it selected.
- Automatic Volume Recognition (AVR): The PVR is directed to mount the PVR cartridge onto any device within a device location domain (DLD) and the PVR selects a qualified device. The PVR does not return the name of the selected device to the client application. The client is required to determine the selected device through alternative means.

Populating PVRs with Cartridges

A PVR is populated with PVR cartridges in a two-step process. Either step may be performed first, but both steps must be completed before the PVR cartridge becomes available for use.

- *Inject* is a PVR directive issued by the PVR client which logically creates the PVR cartridge and assigns it to an MLD. The PVR cartridge name is provided by the client through the inject directive. It is not necessary for the PVR cartridge to be physically present for the inject to succeed.
 - If the PVR cartridge name is in the PVR and is in the *injected* or *available* state, reject the directive.
 - If the PVR cartridge name is in the PVR and is in the *entered* state, move the cartridge to the proper MLD and change the PVR cartridge state to *available*.
 - If the PVR cartridge is not in the PVR, create a PVR cartridge object placing it in the *injected* state. The PVR will then issue a request to a PVR tender that the PVR cartridge be physically *entered* through the PVR inport.
- *Enter* is an asynchronous action wherein a PVR cartridge is physically inserted into the PVR. The PVR cartridge name is provided when the cartridge is entered - manually or via a PVR vision system.
 - If the PVR cartridge name is in the PVR and is in the *entered* or *available* state, reject the cartridge.
 - If the PVR cartridge name is in the PVR and is in the *injected* state, move the cartridge to the proper MLD and change the PVR cartridge state to *available*.
 - If the PVR cartridge is not in the PVR, create a PVR cartridge object placing it in the *entered* state.

Object Attributes

Each object has four classes of attributes:

- *Identifier* attributes are used to name the object. Objects have a single identifier which is sometimes referred to as its primary identifier. Primary identifiers are specified during the creation of the object, or in the case of cartridges, when the cartridge is entered.
- *Characteristic* attributes describe the object and can be modified with the *set* directive. Some characteristic attributes must be specified when the object is created while others are optional and have default values. Frequently, the object must be in the *disabled* state in order for a characteristic attribute to be set.
- *Status* attributes describe the objects current status and its relationship to other objects cannot be modified with the set directive. In all cases, status attributes have default values.

- *Counter* attributes count events or actions. Initial values of counter attributes may be optionally specified on the object's *create* directive, but counter attributes cannot be modified with the *set* directive. Creation time is considered a counter attribute. On object creation, if creation time is not specified, it is set to the current time. For other counter attributes, if no initial value is specified, the attribute is set to zero during object creation.

Events

PVR objects generate events; in fact each counter attribute except creation time corresponds to an event. Typical events are *access denied* and *access granted*. A mechanism is provided to poll the PVR for a list of its events.

Characteristics of the Application Programming Interface

The Application Programming Interface which is being put forward as a proposed standard is a C-language interface which may be implemented in a distributed environment, such as the Distributed Computing Environment (DCE). References to objects are generally made through handles, which are opaque types containing information that allows the server to optimize access to the object. Each directive returns 0 on success and -1 on failure. In the case of failure, it sets the status variable to give further information (similar to the UNIX *errno*).

The following sections summarize the Application Programming Interface. It is fully documented in [3]. In addition to the directives listed below, there are convenience functions which free memory, handles which the server has allocated for the client, directives which list an object's members, and directives which enable or disable objects.

Handle Directives

The following routines take as input a name, or a name and another handle, and create a handle to the named object. Note that the task name is a 32-bit unsigned integer.

```
int pvr_area_get_handle (pvr_t pvr, wchar_t *area_name, pvr_area_t
    *area, pvr_status_t *status);

int pvr_dld_get_handle (pvr_t pvr, wchar_t *dld_name, pvr_dld_t *dld,
    pvr_status_t *status);

int pvr_drive_get_handle (pvr_t pvr, wchar_t *drive_name, pvr_drive_t
    *drive, pvr_status_t *status);

int pvr_get_handle (wchar_t *pvr_name, pvr_t *pvr_h, pvr_status_t
    *status);

int pvr_mld_get_handle (pvr_t pvr, wchar_t *mld_name, pvr_mld_t *mld,
    pvr_status_t *status);

int pvr_part_get_handle (pvr_t pvr, wchar_t *part_name, pvr_part_t
    *part, pvr_status_t *status);

int pvr_port_get_handle (pvr_t pvr, wchar_t *port_name, pvr_port_t
    *port, pvr_status_t *status);
```

```
int pvr_task_get_handle (pvr_t pvr, mss_unsigned32_t task_id,
    pvr_task_t *task, pvr_status_t *status);
```

Basic Create and Delete Directives

The following create and delete directives are available. The create directive supplies a linked list of attributes to be initialized upon creation. The error id identifies the first attribute, if any, that was in error.

```
int pvr_create (pvr_t pvr, pvr_id_attr_t *attribute_list, pvr_attr_t
    *error_id, pvr_status_t *status);
int pvr_delete (pvr_t pvr, pvr_status_t *status);

int pvr_area_create (pvr_area_t area, pvr_id_attr_t *attribute_list,
    pvr_attr_t *error_id, pvr_status_t *status);
int pvr_area_delete (pvr_area_t area, pvr_status_t *status);

int pvr_dld_create (pvr_dld_t dld, pvr_id_attr_t *attribute_list,
    pvr_attr_t *error_id, pvr_status_t *status);
int pvr_dld_delete (pvr_dld_t dld, pvr_status_t *status);

int pvr_drive_create (pvr_drive_t drive, pvr_id_attr_t
    *attribute_list, pvr_attr_t *error_id, pvr_status_t *status);
int pvr_drive_delete (pvr_drive_t drive, pvr_status_t *status);

int pvr_mld_create (pvr_mld_t mld, pvr_id_attr_t *attribute_list,
    pvr_attr_t *error_id, pvr_status_t *status);
int pvr_mld_delete (pvr_mld_t mld, pvr_status_t *status);

int pvr_port_create (pvr_port_t port, pvr_id_attr_t *attribute_list,
    pvr_attr_t *error_id, pvr_status_t *status);
int pvr_port_delete (pvr_port_t port, pvr_status_t *status);

int pvr_part_create (pvr_part_t part, pvr_id_attr_t *attribute_list,
    pvr_attr_t *error_id, pvr_status_t *status);
int pvr_part_delete (pvr_part_t part, pvr_status_t *status);

int pvr_task_delete (pvr_task_t task, pvr_status_t *status);
```

Basic Set and Show Directives

The set directives change or initialize the attributes in the linked list of attributes. Error identifies the first attribute, if any, that was in error. The show directive takes a linked list of attribute identifiers and creates a linked list giving the current state of the attributes of the object.

```
int pvr_set (pvr_t pvr, pvr_id_attr_t *attribute_list, pvr_attr_t
    *error_id, pvr_status_t *status); int pvr_show (pvr_t pvr,
    pvr_attr_id_request_t *input_list,
    pvr_id_attr_t **output_list , pvr_o_flag_t flag, pvr_status_t
    *status);
```

```

int pvr_area_set (pvr_area_t area, pvr_id_attr_t *attribute_list,
    pvr_attr_t *error_id, pvr_status_t *status);
int pvr_area_show (pvr_area_t area, pvr_attr_id_request_t
    *input_list, pvr_id_attr_t **output_list, pvr_o_flag_t *flag,
    pvr_status_t *status);

int pvr_cart_set (pvr_t pvr, wchar_t *name, pvr_id_attr_t
    *attribute_list, pvr_attr_t *error_id,
int pvr_cart_show (pvr_t pvr, wchar_t *name, pvr_attr_id_request_t
    *input_list, pvr_id_attr_t **output_list, pvr_o_flag_t *flag,
    pvr_status_t *status);

int pvr_cart_set (pvr_t pvr, wchar_t *name, pvr_id_attr_t
    *attribute_list, pvr_attr_t *error_id,
int pvr_cart_show (pvr_t pvr, wchar_t *name, pvr_attr_id_request_t
    *input_list, pvr_id_attr_t **output_list, pvr_o_flag_t *flag,
    pvr_status_t *status);

int pvr_drive_set (pvr_drive_t drive, pvr_id_attr_t *attribute_list,
    pvr_attr_t *error_id, pvr_status_t *status);
int pvr_drive_show (pvr_drive_t drive, pvr_attr_id_request_t
    *input_list, pvr_id_attr_t **output_list, pvr_o_flag_t *flag,
    pvr_status_t *status);

int pvr_mld_set (pvr_mld_t mld, pvr_id_attr_t *attribute_list,
    pvr_attr_t *error_id, pvr_status_t *status);
int pvr_mld_show (pvr_mld_t mld, pvr_attr_id_request_t *input_list,
    pvr_id_attr_t **output_list, pvr_o_flag_t *flag, pvr_status_t
    *status);

int pvr_part_set (pvr_part_t part, pvr_id_attr_t *attribute_list,
    pvr_attr_t *error_id, pvr_status_t *status);
int pvr_part_show (pvr_part_t part, pvr_attr_id_request_t
    *input_list, pvr_id_attr_t **output_list, pvr_o_flag_t *flag,
    pvr_status_t *status);

int pvr_port_set (pvr_port_t port, pvr_id_attr_t *attribute_list,
    pvr_attr_t *error_id, pvr_status_t *status);
int pvr_port_show (pvr_port_t port, pvr_attr_id_request_t
    *input_list, pvr_id_attr_t **output_list, pvr_o_flag_t *flag,
    pvr_status_t *status);

int pvr_task_set (pvr_task_t task, pvr_id_attr_t *attribute_list,
    pvr_status_t *status);
int pvr_task_show (pvr_task_t task, pvr_attr_id_request_t
    *input_list, pvr_id_attr_t **output_list, pvr_o_flag_t *flag,
    pvr_status_t *status);

```

Additional Area Directives

The *flush* directive returns all cartridges from the area to their home positions. The *stage* and *scratch* directives put the list of cartridges into the stage and scratch area respectively.

The *list* directive lists the cartridges in an area. The *mount scratch* directive mounts from the scratch area.

```
int pvr_area_flush (pvr_area_t area, mss_unsigned32_t *task,
    pvr_status_t *status);
int pvr_area_stage (pvr_area_t area, pvr_string_list_t *cart_list,
    mss_unsigned32_t *task, pvr_status_t *status);
int pvr_area_scratch (pvr_area_t area, pvr_string_list_t *cart_list,
    mss_unsigned32_t *task, pvr_status_t *status);
int pvr_area_list (pvr_area_t area, pvr_string_list_t **output_list,
    pvr_status_t *status);
int pvr_area_mount_scratch (pvr_area_t area, pvr_mount_type_t type,
    wchar_t *device_spec, wchar_t *info, mss_unsigned32_t *task,
    pvr_status_t *status);
```

Additional Cartridge Directives

The *enter* and *inject* directives place the cartridge into the PVR. The *eject* sends it out of the PVR. The *relocate* directive places the cartridge in a different location. There are different kinds of *mount* directives, and one *dismount* directive.

```
int pvr_cart_eject (pvr_t pvr, wchar_t *cart_name, wchar_t *port,
    mss_unsigned16_t *cell, pvr_status_t *status);

int pvr_cart_enter (pvr_t pvr, wchar_t *cart_name, wchar_t *port,
    pvr_status_t *status);

int pvr_cart_inject (pvr_t pvr, wchar_t *cart_name, wchar_t *port,
    mss_unsigned16_t *cell, wchar_t *mld, pvr_status_t *status);

int pvr_cart_relocate (pvr_t pvr, wchar_t *cart_name, wchar_t *mld,
    pvr_status_t *status);

int pvr_cart_bind_mount (pvr_t pvr, wchar_t *cart_name,
    pvr_comp_cart_list_t *list, pvr_mount_type_t type, wchar_t
    *device, mss_enum_boolean_t read_only, wchar_t *info,
    mss_unsigned32_t *task, pvr_status_t *status);

int pvr_cart_bind_mount_scratch (pvr_t pvr, mss_unsigned16_t
    quantity, wchar_t *area, pvr_mount_type_t type, wchar_t *device,
    wchar_t *info, mss_unsigned32_t *task, pvr_status_t *status);

int pvr_cart_mount (pvr_t pvr, wchar_t *cart_name, mss_unsigned16_t
    side, pvr_mount_type_t type, wchar_t *device, mss_enum_boolean_t
    read_only, wchar_t *info, mss_unsigned32_t *task, pvr_status_t
    *status);

int pvr_cart_dismount (pvr_t pvr, wchar_t *cart_name, wchar_t *area,
    mss_unsigned32_t *task, pvr_status_t *status);
```

Additional PVR Directives

The `pvr_ack_mount` acknowledges that the client has detected that a mount or mount scratch operation has completed. The client provides the name of the device and cartridge that has been mounted

```
int pvr_ack_mount (pvr_task_t task, wchar_t *device_name, wchar_t
    *cart_name, mss_unsigned16_t *side, mss_enum_boolean_t process_check,
    pvr_status_t *status);
```

The `pvr_update_mount_status` function is sent to a dependent PVR to update the cartridge name that was mounted.

```
int pvr_update_mount_status (pvr_t pvr, wchar_t *mounted_cart_name,
    pvr_status_t *status);
```

Additional Drive Directive

The following directive enables the client to configure a drive.

```
int pvr_drive_configure (pvr_drive_t drive, wchar_t *node_name, wchar_t
    *local_drive_name, mss_enum_boolean_t *excluded_flag, pvr_status_t
    *status);
```

Additional MLD Directive

The following directive lists the cartridges within an MLD.

```
int pvr_mld_list (pvr_mld_t mld, pvr_cart_state_t *state,
    pvr_slot_list_t **carts, pvr_status_t *status);
```

Additional Partition Directives

The `pvr_part_copy` directive streamlines configuration of partitions by allowing one to be a copy of another. The `pvr_part_inventory` directive starts a task which inventories the cartridges of a partition. The `pvr_part_list` directive provides a list of the cartridges in a partition.

```
int pvr_part_copy (pvr_part_t part, pvr_id_attr_t *attribute_list,
    mss_enum_boolean_t descendent_flag, wchar_t *old_part_spec,
    pvr_status_t *status);
```

```
int pvr_part_inventory (pvr_part_t part, mss_unsigned32_t start,
    mss_unsigned32_t count, mss_unsigned32_t *task, pvr_status_t
    *status);
```

```
int pvr_part_list (pvr_part_t part, pvr_cart_state_t *state,
    pvr_slot_list_t **carts, pvr_status_t *status);
```

Additional Task Directives

Directives are provided to manage tasks within the PVR. Tasks execute asynchronously, and the `pvr_task_wait` function allows the calling thread to wait for the task to complete. The `pvr_task_cancel`, `pvr_task_pause`, and `pvr_task_resume` allow for task management.

```
int pvr_task_cancel (pvr_task_t task, pvr_reason_code_t code,
                    pvr_status_t *status);
int pvr_task_pause (pvr_task_t task, pvr_status_t *status);
int pvr_task_resume (pvr_task_t task, pvr_status_t *status);
int pvr_task_wait (pvr_task_t task, pvr_task_completion_status_t
                  *task_status, pvr_status_t *status);
```

Polling Directives

There is a collection of polling directives which allows the client to poll one instance or all instances of an object for a specified list of events.

```
int pvr_poll_events (pvr_t pvr, time_t start_time, pvr_event_request_t
                    *input_list, pvr_id_event_t **output_list, pvr_status_t *status);
int pvr_drive_poll_events (pvr_drive_t drive, time_t start_time,
                           pvr_event_request_t *input_list, pvr_id_event_t **output_list,
                           pvr_status_t *status);
int pvr_all_drive_poll_events (pvr_t pvr, time_t start_time,
                               pvr_event_request_t *input_list, pvr_id_event_t **output_list,
                               pvr_status_t *status);
int pvr_port_poll_events (pvr_port_t port, time_t start_time,
                          pvr_event_request_t *input_list, pvr_id_event_t **output_list,
                          pvr_status_t *status);
int pvr_all_port_poll_events (pvr_t pvr, time_t start_time,
                              pvr_event_request_t *input_list, pvr_id_event_t **output_list,
                              pvr_status_t *status);
int pvr_area_poll_events (pvr_area_t area, time_t start_time,
                          pvr_event_request_t *input_list, pvr_id_event_t **output_list,
                          pvr_status_t *status);
int pvr_all_area_poll_events (pvr_t pvr, time_t start_time,
                              pvr_event_request_t *input_list, pvr_id_event_t **output_list,
                              pvr_status_t *status);
int pvr_task_poll_events (pvr_task_t task, time_t start_time,
                          pvr_event_request_t *input_list, pvr_id_event_t **output_list,
                          pvr_status_t *status);
int pvr_all_task_poll_events (pvr_t pvr, time_t start_time,
                              pvr_event_request_t *input_list, pvr_id_event_t **output_list,
                              pvr_status_t *status);
int pvr_dld_poll_events (pvr_dld_t dld, time_t start_time,
                         pvr_event_request_t *input_list, pvr_id_event_t **output_list,
                         pvr_status_t *status);
int pvr_all_dld_poll_events (pvr_t pvr, time_t start_time,
                             pvr_event_request_t *input_list, pvr_id_event_t **output_list,
                             pvr_status_t *status);
```

```
int pvr_dld_poll_events (pvr_dld_t dld, time_t start_time,  
    pvr_event_request_t *input_list, pvr_id_event_t **output_list,  
    pvr_status_t *status);  
int pvr_all_dld_poll_events (pvr_t pvr, time_t start_time,  
    pvr_event_request_t *input_list, pvr_id_event_t **output_list,  
    pvr_status_t *status);  
int pvr_mld_poll_events (pvr_mld_t mld, time_t start_time,  
    pvr_event_request_t *input_list, pvr_id_event_t **output_list,  
    pvr_status_t *status);  
int pvr_all_mld_poll_events (pvr_t pvr, time_t start_time,  
    pvr_event_request_t *input_list, pvr_id_event_t **output_list,  
    pvr_status_t *status);
```

References

Links to the following documents may be found at <http://www.arl.mil/IEEE/ssswg.html>.

1. *Reference Model for Open Storage Systems Interconnection*, IEEE Storage System Standards Working Group, Project 1244, September 8, 1994.
2. *IEEE Storage System Standards Physical Volume Repository Model (proposed)*
3. *The Physical Volume Application Programming Interface*

A Global Distributed Storage Architecture

55-82
73187

Dr. Nemo M. Lionikis
Michael F. Shields
Department of Defense
9800 Savage Road
Fort Meade, MD 20755
nemo@romulus.ncsc.mil
mfs@romulus.ncsc.mil
Tel: 301-688-9509
Fax: 301-688-9599

I. Introduction

NSA architects and planners have come to realize that to gain the maximum benefit from, and keep pace with, emerging technologies, we must move to a radically different computing architecture. The compute complex of the future will be a distributed heterogeneous environment, where, to a much greater extent than today, network-based services are invoked to obtain resources. Among the rewards of implementing the services-based view are that it insulates the user from much of the complexity of our multi-platform, networked, computer and storage environment and hides its diverse underlying implementation details. In this paper, we will describe one of the fundamental services being built in our envisioned infrastructure; a global, distributed archive with near-real-time access characteristics. Our approach for adapting mass storage services to this infrastructure will become clear as the service is discussed.

II. High Level Architecture

As a world-wide organization, NSA's storage and retrieval services must provide for rapid, efficient, and user-driven data access from any node in our organization. Storage services must be accessible yet secure, scalable, reliable, cost effective, and manageable. The technologies used to implement storage must be commercial-off-the-shelf (COTS) wherever possible and the user interface to these services must be clear and simple. Moreover, a key requirement of the services is that they must support the notion of near-real-time access to data.

Because traditional file-based solutions, with their induced latency, are inadequate to meet the near real-time processing requirements being levied today by our users, we are developing the Byte Stream Storage and Transfer Service. The user sees the Byte Stream Storage and Transfer Service as a globally distributed archive with near-real-time access. The service is intended as a mechanism that allows a user to access and manipulate data streams. It is a critical feature of the stream service that while a producer is creating a stream at one location, a consumer, possibly at a geographically remote location, can

begin to access the producer's data. One of our design goals is that no matter where users are located, a consumer can begin accessing data within seconds of its creation.

One of the most radical aspects of the proposed stream service is the assumption of all storage management by the service. There is no concept of an "archived" stream. Once data has been written into the service, the user has one, and only one, view of it. The user sees "a stream", not "a local disk copy" or "an archived copy", each with its own interface involving different commands and even operator intervention to gain access. No knowledge of data location is required on the part of a user. No special commands to access storage are required. No special commands to transfer data to the processing system are required. No thought, beyond initial system configuration, is given to availability of space. No application code is required to handle file boundaries and file names for a stream of data. These mechanisms are created once and for all in the service and then applied consistently to every stream. Users must only know the name of the stream they wish to access and the service will find and deliver the data.

A user-level application that processes live, non-burst signals should be able to work with a data abstraction that models the "stream-oriented" nature of these signals. The notion of a Byte Stream Storage and Transfer Service was devised to support such a data abstraction. A by-product of adopting the stream data abstraction is that it supports the notion of near-real-time processing of live data quite naturally. When moving a byte stream, we do not assume that the entire stream is present or, in fact, that the entire stream even exists yet. We cannot think in terms of transferring the entire stream to a specified host and processing it. Rather, we are constantly transferring bytes of the stream as they are created. The concept of a service that moves and stores streams is not *apriori* necessary, but its advantages are huge. One cannot overstate the value of a single, universally accepted abstraction for a byte stream, captured in a stream service. Not having such a service requires producing distinct, possibly incompatible, file-based solutions for each new production data flow, with all of the attendant naming, storage, movement, administration, accounting, and maintenance issues that the new solutions would demand.

Internally, the byte-stream service is a set of geographically distributed relay/storage hubs (Figure 1), that cooperate with each other and with interface software running within a stream consumer or producer process, to accomplish the movement and storage of data. The hubs are connected via a network and control software within the hubs communicates via standard protocols (TCP/IP or UDP/IP). Hubs are logical entities that may consist of several systems. A stream might reside within a single hub or be distributed among multiple hubs. Multiple copies of pieces of the stream may exist in different hubs. A consumer will receive a copy from the nearest hub. There are no coherency issues because a stream can be written only once (archive semantics). There are, however, issues of deleting extra copies when they become old or inactive and the

stream service must institute policies to manage this. In general, one copy will be labeled for retention and all other copies will be considered cached and can be deleted.

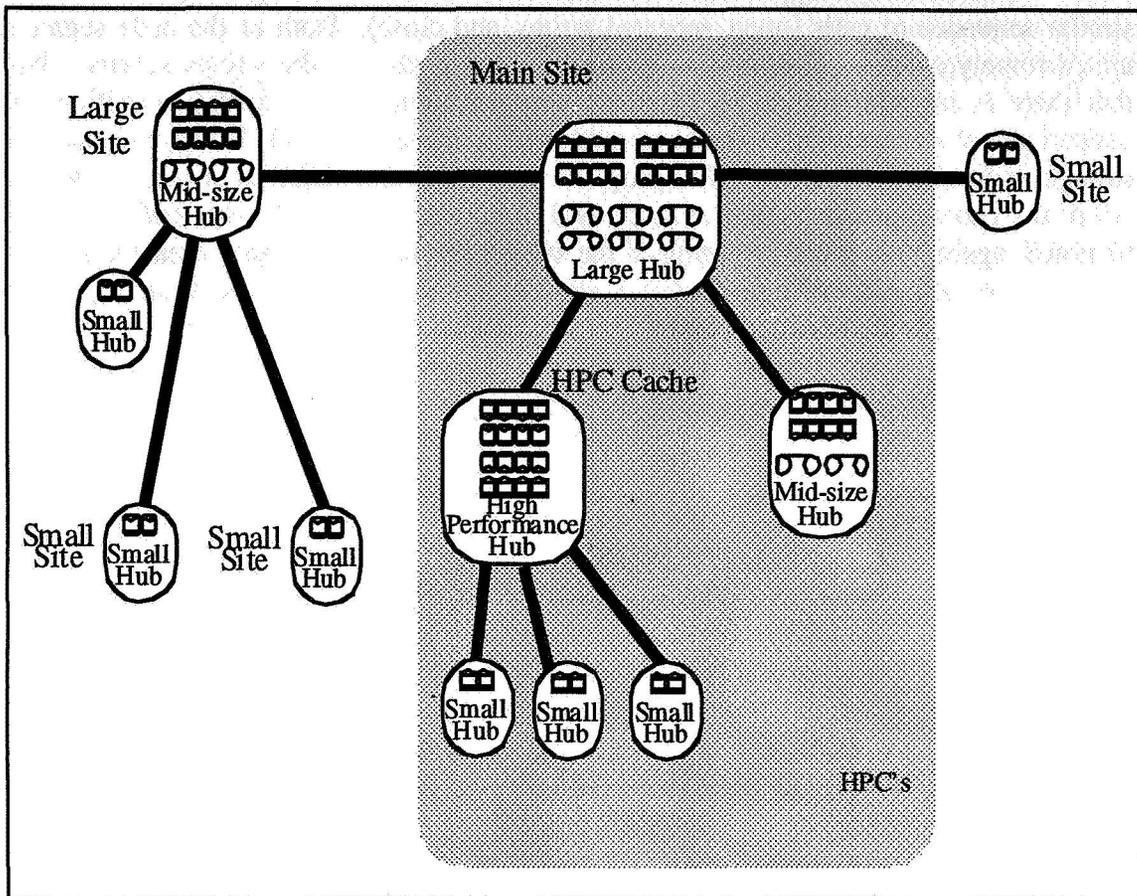


Figure 1: High Level Architecture of the Global Archive

The stream service interface will mimic the POSIX system call I/O interface, with common UNIX extension, using the C-language binding. There are at least two compelling reasons for doing this. First, the POSIX system call interface has been used in countless settings and has proven its versatility. It is safe to assume that the interface will support both current and future requirements. Second, developers are already familiar with the interface, so, learning to use the stream service should not be an onerous task. Having said this, it must be pointed out that the stream service interface will be a mixture of file and network semantics. This is because it is desirable to allow a developer to use such calls as `creat()`, `open()`, `close()`, `read()`, `write()`, and `lseek()` from the file domain. It is also necessary, though, to provide the capabilities of `select()` from the network domain in order to support the abstraction of a near-real-time stream. The actual interface will consist of a library of subroutines containing at least `yopen()`, `ycreat()`, `yclose()`, `yread()`, `ywrite()`, `ylseek()`, and `yselect()`.

Figures 2 and 3 provide simple examples of how these routines can be used to read or write a stream. To read, a user application will open a stream, referring to it by a name. The application then seeks to the position of interest and repeatedly reads and processes data. When done, the application will close the stream. Writing a stream will be a similar sequence of calls (open, repeated writes, and close). Both of the code segments are extremely simple and dramatically illustrate the virtues of the stream service. Note that there is no reference to location, no concern about file boundary conditions, no concern about storage. There is also no notion of whether the data is being obtained from storage or from a live source. The program only requires a name to access the stream. All of the general problems of movement and storage are handled transparently. It should be noted, again, that one assumption of the stream service is that, once created, a stream cannot be edited. In order to modify a stream, it must be read, processed, and a new stream created for the resultant output.

```

stream_id      sid;
char*          buffer[M A X ];
int            bytes_read;

sid = yopen("stream_name", YO_RDONLY);
ylseek(sid, POSITION, YSEEK_SET);
while ( (bytes_read = yread(sid, buffer, M A X)) != ERROR ) {
    /* Process the bytes read from the stream ..... */
}
yclose(sid)

```

Figure 2: Reading a byte stream

```

stream_id      sid;
char*          buffer[M A X ];
int            bytes_written;

sid = yopen("stream_name", YO_CREAT | YO_WRONLY);
while ( NOTDONE ) {
    /* Get data and perform processing ..... */
    bytes_written = ywrite(sid, buffer, M A X );
}
yclose(sid)

```

Figure 3: Creating and writing a byte stream

When a user process wishes to write a stream, it begins by calling `yopen`. Internally, the service interface software establishes communications with its local hub (Figure 4). When writing begins, an agent is started on the hub and is connected to the interface software in the user process. Data then passes through the interface to the agent on the hub which caches the data on disk. As the cache fills, data may be moved by the service to storage systems within the hub for short-term retention. At this point a stream (potentially, but not necessarily, live) is being captured and stored. Note that storage is not a direct concern of the user process.

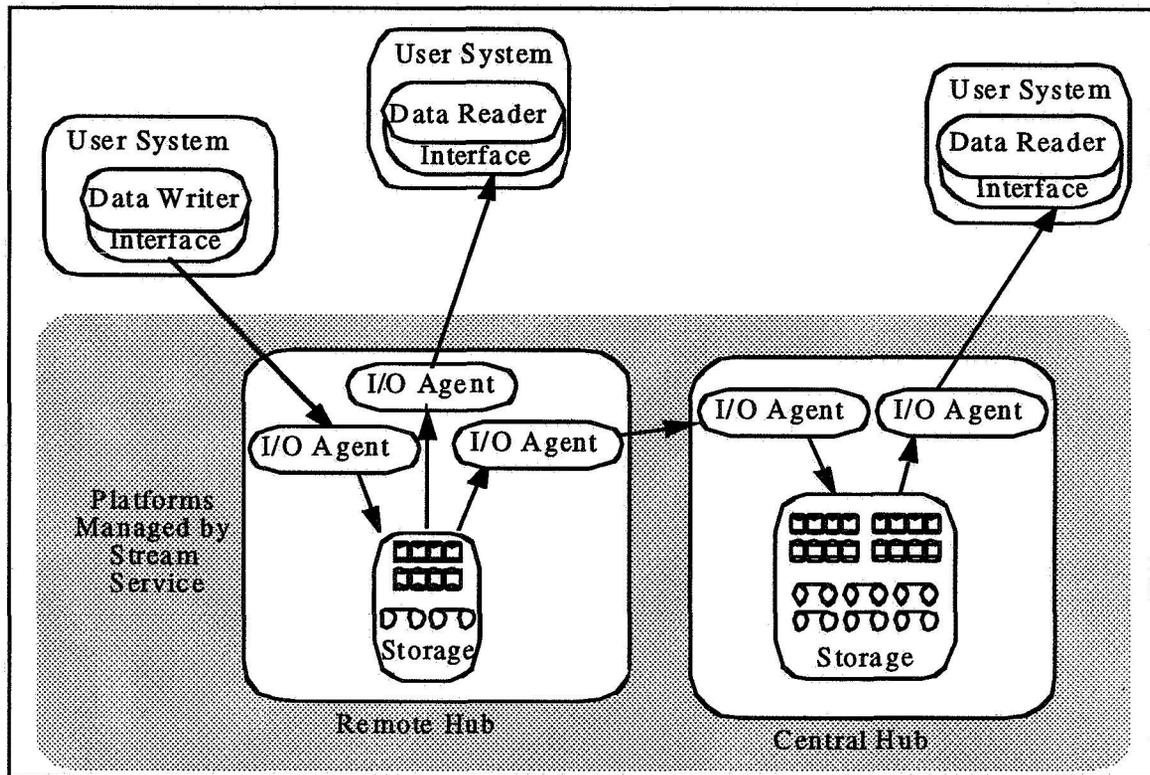


Figure 4: Data Flow Examples

When a user process wishes to read a stream, it begins by calling `yopen`, and, once again, the service interface software establishes communication with its local hub. When `yread` is called, the local hub determines if the desired data is present. If not, the hub finds a remote hub that has the desired data, and requests a transfer from the remote hub to the local hub. Now, with data present in the local hub, a connection is established from an agent in the hub to the interface software in the user process, and data is forwarded. As the data arrives from a remote hub, it is cached in the local hub. As the cache fills, data moves to a storage system for retention. Note that the reading process may or may not be receiving live data, and is unaware and unconcerned as to whether the data originated at a local or a remote location. In fact, all storage details are hidden from the user.

A near-real-time flow is established when a stream is being produced at the same time that it is being consumed. Should a communication outage occur between hubs, data will not be lost because the hub that is local to the stream producer will continue to cache and store data. Of course, a network outage between the producing system and its local hub will cause a data loss if the buffering capability of the producing system is exceeded. Consumers and producers can run on the storage platform. In this case, the network will be circumvented and we expect to observe reading and writing at very near disk speeds.

One of the great advantages of the service described here is that accessing stored data is exactly the same as accessing live data. It is the responsibility of the local hub to discover where pieces of a stream are stored. If a stream has been moved from cache to storage, the hub will ensure that it is drawn into cache again with forwarding identical to the near-real-time case.

It is common to associate related information with a byte stream. A follow-on development, the Annotated Stream Service, built on top of the Byte Stream Storage and Transfer Service, supports this notion. An annotated stream consists of several byte streams, one being a data stream and the remainder being annotation streams. The Annotated Stream Service provides a mechanism for a stream writer to associate annotations with specific points in a data stream. For a stream reader, the service synchronizes the reading of the annotations with the reading of the data. The internal form of an annotation is chosen by the application developer. The service merely provides a framework for the association, storage, and synchronized delivery of the data and the annotations. As with the byte stream service, all of this is done while still preserving a simple "open, close, read, write" interface.

III. Storage Strategy

Guiding Principles: NSA has adopted a COTS, to the maximum extent possible, approach to any Mass Storage requirement. As a direct result of this policy, we have carefully approached the global distributed storage architecture steered by previous work in developing a scalable set of disk and tape components, subsystems and systems matched to specific requirements. Significant consideration is given to performance, functionality, and cost, with a keen eye on system level reliability. To the maximum extent possible, we strive to achieve vendor independence and network connectivity; wherever possible, we desire data sharing and products which facilitate technology insertion. Finally, remote monitoring is key to overall system viability.

Product Considerations:

Disk Storage: For both large and small nodes the disk subsystems are almost always specified to be RAID devices. While the majority of the current set of disk subsystems are SCSI-2 F/W, our high-end nodes will require fibrechannel speeds. The ability to remote the arrays beyond today's cable limits greatly enhances our physical layout potential. In addition, the ability to connect large arrays to multiple servers enhances our reliability, shareability, and control. NSA has relied heavily on shared network disk arrays within our supercomputer complexes and has urged industry to develop products of this class. To achieve the desired performance and flexibility for the individual nodes of this architecture, extremely large network RAID arrays are a must. The disk arrays must be platform independent, reducing reliance on any single vendor.

Robotic Tape Storage: Our larger nodes require robotic tape libraries which range from tens of terabytes up to multiple petabytes. They are sized to match specific user needs from a performance, capacity, and user access perspective. We envision each node to have multiple tape libraries, matched to the specific type of stream data. Our goal is to make our distributed library infrastructure transparent to the user. While certain data types lend themselves to very large capacity libraries, others do not. As such, our experience with the current set of storage management software offerings forces us to adopt a multiple library strategy. The majority of today's products use commercial relational database management system (RDBMS) products to manage the files stored in the libraries and this artifact must be accommodated in the overall architecture. Most of the products evaluated to date are limited to the tens of millions of files. Large files (>150 MB) are ideally suited to high performance helical drives which can deliver petabyte class individual libraries. However, small file (15 MB) mass storage libraries will outpace the RDBMS' ability to scale to the 100 million file mark. Because the stream service controls file creation, large files should be the norm. While these numbers are not exact for today's storage software offerings, they are representative of the challenge that system architects face in designing a multi-node hub. The vendor community can deliver hardware that easily scales into the multiple petabyte range today; however, the storage software lacks the maturity, performance, and ability to service this class of system. Although the smaller nodes are disk only, they will still require high-performance robotic tape backup systems.

Storage Software: There are two common cross vendor categories of storage software in wide use today at NSA, Hierarchical Storage Management (HSM) and Virtual File System (VFS) software. Of the two, the latter is most widely used. HSMs classically are major computer systems (processors, disk, robotic tape) that are network connected to multiple client systems. Both VFS and HSM are primarily skewed towards the operational paradigm of store with infrequent retrieves. While performance is dependent on multiple factors and is highly dependent upon the network connectivity, VFS systems generally deliver higher performance than HSMs. VFSs today use large UNIX servers with RAID arrays and manage 7-40 TB robotic tape libraries. They too are network

connected to multiple client systems, but do not possess the full range of archive functionality of the HSM. However, they interact with almost any client and provide a file system view to that client; hence they are very easy to install and are widely used by a diverse population due to their simple interface. To support the distributed storage architecture, our large node will be based on multiple VFS storage systems. Emerging multimedia software products easily embrace this technology which further enhances its role in our infrastructure. Finally, the multiple library approach facilitates technology insertion for the physical components that make up the storage library allowing for the migration of data to be performed as a background job as older drive technology is retired.

MetaData: The most difficult element of the storage system is the metadata system. With multiple, disparate libraries connected to the large node, and several nodes in a hub within the archive, transparent access by a diverse population is facilitated by this critical element. Its importance has been recognized by the Mass Storage Community as evidenced by the IEEE sponsoring a yearly MetaData Conference. The ability to manage hundreds of millions to billions of files can only be done by a carefully designed metadata system. NSA has taken the approach of a distributed metadata system for its scientific processing complex; however, to scale to the numbers of files needed for the future, significant breakthroughs are needed. Suffice it to say, that the integration and use of metadata and its storage will need to be accomplished. Scalability here is fundamental to the success of this endeavor. This paper will not address metadata.

IV. Initial Development Plans

The architecture discussed above will be implemented incrementally. The intent of the initial configuration is to present users with the first view of the stream service/distributed archive and validate the concepts contained in the architecture.

Near Term Plans: Initially, a single-system, mid-sized hub will be built. The hub will employ a medium performance UNIX server with tens to hundreds of gigabytes of disk cache and a single robotic tape library. The storage software will be Virtual File System based. The hub will run early increments of the stream service software and will be used to validate many of the concepts of the architecture. After the first hub has been built and tested, a second, large, two-system hub will be built. The hub will consist of two identical high-end UNIX servers, each with a large size RAID disk array and one or more robotic tape libraries. Both high-end and medium performance/capacity libraries may be employed and, again, the storage software will be Virtual File System based. The systems will have multiple network connections of differing performance levels (FDDI, ATM, and Ethernet). The two-system hub will run a follow-on increment of the stream service that manages the multiple storage platforms. Inter-hub data transfers will be based on a static policy. A mix of user workstations which mirror the current

infrastructure will complete the near term test configuration (Figure 5). Using this configuration, we intend to evaluate the user interface, desired functionality, initial scalability, overall reliability, as well as subsystem, software, and system reliability. We will focus on the adequacy of the specific technologies chosen, calibrate performance choke points and scalability considerations. As a result of our analyses, the overall architecture will be modified, if necessary, and the lessons learned will be incorporated into our long term plans.

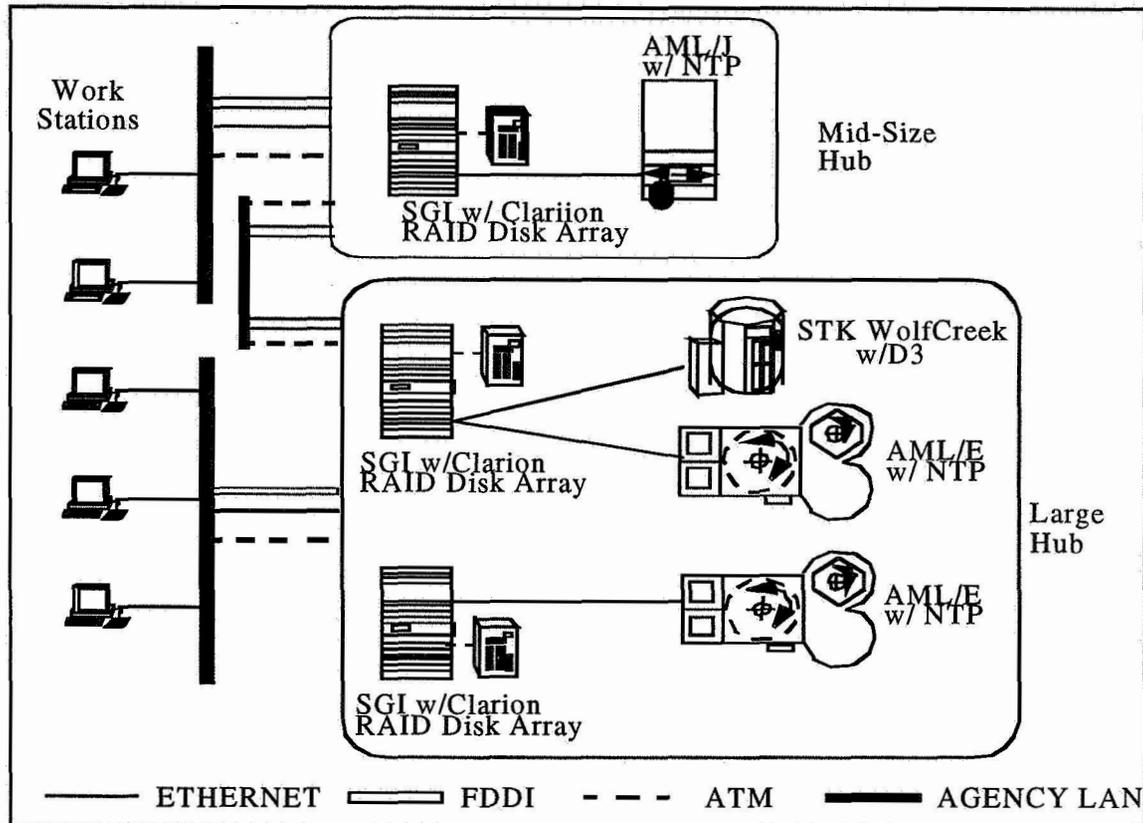


Figure 5: Complete Near Term Test

Longer Term Plans: While this area is highly dependent upon the prior phase and its success, several features are already slated for implementation in the longer term. Among these are:

- Inclusion of a wide area network (WAN) connected hub
- Inclusion of bandwidth management and flow control between hubs across the WAN
- Increasing the numbers/scales of hubs and MSS libraries
- Evaluation of metadata system approaches and their scalability

Other areas under consideration, even though they are merely “on the drawing board”, include:

- Expansion of the server area to include Massively Parallel Processors (MPPs)
- Inclusion of Web-based user access
- Inclusion of MultiMedia into the test set

V. Conclusions

In summary, this paper has been an attempt to present a brief overview of the architecture for a global distributed archive with near-real-time access characteristics and the strategy for use of mass storage systems within that architecture. The instantiation of the architecture is clearly a long term project that must be approached incrementally. As such, it is vital that the interface to the archive be implemented early on and that the archive be expanded and improved transparently to early users, behind this interface. Although we would not minimize the challenge of the long term development, we hope that the tremendous benefits to be gained by building such an archive are evident from this brief exposition.

56-82
83188

Petabyte Class Storage at Jefferson Lab (CEBAF)

Rita Chambers, Mark Davis
Jefferson Lab Computer Center
12000 Jefferson Ave.
Newport News VA 23606
chambers@cebaf.gov
davis@cebaf.gov
Tel: 757-269-7514
Fax: 757-269-7053

Abstract

By 1997, the Thomas Jefferson National Accelerator Facility will collect over one Terabyte of raw information per day of Accelerator operation from three concurrently operating Experimental Halls. When post-processing is included, roughly 250 TB of raw and formatted experimental data will be generated each year. By the year 2000, a total of one Petabyte will be stored on-line.

Critical to the experimental program at Jefferson Lab (JLab) is the networking and computational capability to collect, store, retrieve, and reconstruct data on this scale. The design criteria include support of a raw data stream of 10-12 MB/second from Experimental Hall B, which will operate the CEBAF Large Acceptance Spectrometer (CLAS). Keeping up with this data stream implies design strategies that provide storage guarantees during accelerator operation, minimize the number of times data is buffered, allow seamless access to specific data sets for the researcher, synchronize data retrievals with the scheduling of postprocessing calculations on the data reconstruction CPU farms, as well as support the site capability to perform data reconstruction and reduction at the same overall rate at which new data is being collected.

The current implementation employs state-of-the-art StorageTek Redwood tape drives and robotics library integrated with the Open Storage Manager (OSM) Hierarchical Storage Management software (Computer Associates, International), the use of Fibre Channel RAID disks dual-ported between Sun Microsystems SMP servers, and a network-based interface to a 10,000 SPECint92 data processing CPU farm. Issues of efficiency, scalability, and manageability will become critical to meet the year 2000 requirements for a Petabyte of near-line storage interfaced to over 30,000 SPECint92 of data processing power.

Introduction

The Thomas Jefferson National Accelerator Facility (formerly CEBAF, the Continuous Electron Beam Accelerator Facility, and now known as Jefferson Lab), located in Newport News, Virginia, operates a 4 GeV continuous wave electron beam accelerator, with the capability to drive fixed target experiments in nuclear physics simultaneously in three Experimental Halls. By 1997, when all three halls are under production operation, the data generation capability of the experiments, including both raw and reconstructed

data, is expected to approach 250 TB per year. By the year 2000, a total of one Petabyte of data will be stored on-line for access to users on both Local and Wide Area Networks.

In this paper, we outline some of the major design decisions and strategies employed in the development of an automated facility which can collect raw experimental results from three separate data acquisition operations, plus serve this information to a 10K+ SPECint92 batch data reconstruction farm. The central mass storage system must also store output from the data reconstruction and analysis process, provide intuitive access to files associated with specific data runs and phases of the analysis, plus provide a data export capability for transport of the summary information back to the researcher's home institution where final analysis steps will be performed.

Some of the most critical decision points in the process require coordination in the design of both the on-line and off-line phases of the operation. The size of the individual data run, which represents a specific running period for each spectrometer, naming conventions for raw data files and associated calibration, target mapping, and other auxiliary files, and the methods used by each experimental hall to funnel data to the central mass storage system must be anticipated in the design of the off-line data handling capability. In some cases, particularly in the size of the raw data file, limitations and optimizations in the off-line process will influence operations during the data acquisition stages. This paper will summarize the basic assumptions in the development of the data handling operation, including considerations in designing the data path for both on-line and off-line operations. We provide a description of the current evolution of the design, status of the current production operation serving one Experimental Hall, plus anticipate the challenges ahead as we scale the operation to support a Petabyte-class data storage requirement.

Data Handling Requirements

Inherent in the design of the data handling operation at Jefferson Lab is the requirement for an automated, "hands-off" operation. Physicists historically have run experiments with their hands "on the wheel" -- actively managing and monitoring the experiment itself while manually loading multiple small tape units to store the generated raw data. In this mode, the volume of the output and the success of the operation are immediately apparent. The researcher is responsible both to develop effective tracking and logging systems as well as to determine and resolve problems encountered in the experimental and data storage facets of the operation. When designing for the collection of 1 TB per day of raw data, it is immediately apparent that this classic mode of operation will not scale: people time is expensive; the task of recording and tracking large numbers of potentially large files is daunting. A 2 GB raw data file, for instance, represents less than 3 minutes of operation of the CEBAF Large Acceptance Spectrometer (CLAS) in Experimental Hall B. Manual logging methods developed when data rates were on the order of Kilobytes per second become unmanageable when data is being generated in Megabytes per second. Part of the design consequently is to meet the human requirements for visual verification of the success of the data storage operation, to develop intuitive methods to locate specific data runs and associated files, and to implement robust strategies to withstand interruptions in the central storage capability without affecting the on-line data acquisition (DAQ) process.

Due to the large scale of the operation, efficiency is tantamount. Critical to the design is the effort to minimize the number of times that raw data must be copied on its path to the central mass storage facility. The analysis of early designs, in fact, revealed up to four separate copies of the raw data file on its way to a robotics silo: DAQ to disk; disk to disk via network to the tape storage server; a re-copy to disk buffers required by some tape management applications; and a final copy to tape in the robotics library. Recopying 1 TB of data incurs large costs in both time and hardware and could significantly increase the resources required. Furthermore, data reconstruction processes in nuclear physics on the average make two to three passes through the original raw data. While a true data reduction in this phase of the analysis is desirable, "reduced" data may, in fact, equal or even exceed the size of the raw input in some instances. There may be many reduction stages in the final creation of a Data Summary Tape (DST) sufficiently small to be transported to the home institution for final analysis. Consequently, efficient algorithms to coordinate the use of tape transports and disk pool areas, to optimize network and batch node performance with central and local disk buffers, to manage on-line storage of output information anticipated for near term re-access, and to vault experimental results to be maintained 10 years or more in off-line storage, are essential in maximizing the use of expensive resources (tape, disk, CPU, network).

The overall design must also meet the requirements of three separate experimental operations, and, in fact, arbitrate resources between the halls, isolating them from each other. JLab's three experimental halls each impose a different set of requirements and timelines for computational support, and in many cases make use of a variety of procedures and standards in the operation of their experiments. The data transport, storage, and post-processing requirements for Experimental Hall B, due to begin production during FY97 (10/1/96-9/30/97), significantly surpass the standard operational requirements for Halls A and C. While planning has focused on meeting the technical challenges posed by the collection and processing of approximately 1 Terabyte per day of raw data (after the compression phase in the data acquisition process) from the CEBAF Large Acceptance Spectrometer (CLAS) in Hall B, a data stream equivalent to approximately 10-12 Megabytes per second, the plans must also provide viable solutions for the lower data rates projected for Halls A and C. The other two halls, which begin operation in an earlier time frame than Hall B, generally incur lower data rates (1-2 MB/second) and while eventually requiring a separate data path from Hall B, can in fact be used to test, tune, and refine the solution for Hall B.

A summary of the basic data storage requirements and timelines for the three halls illustrates that the data handling requirements for Hall B are an order of magnitude greater than for the other two halls:

Hall	Test Runs	Production	Event Size	Events/Sec	MB/Sec	Data/Day
C	Complete	Current	1 KB	200 - 2000	0.2 - 2	1 - 100 GB*
A	2Q96	4Q96	1 KB	200 - 2000	0.2 - 2	1 - 100 GB *
B	4Q96	1Q97	10 KB	1000	10	1 TB

* Approximately 1-25 GB/day under normal operation. Exceptions to this rate are the Hall A Helium parity experiment which will run at 10 KHz (10 MB/sec or about 1 TB/day) for a few months, and Visual Compton Scattering and other experiments which are expected to collect data at 2 KHz (2 MB/sec) with peaks up to 10 KHz.

The data reconstruction requirements for Halls A and C are estimated as 1/10 those of Hall B. Hall B estimates that each Byte of data will require on the order of 1000 instructions to reconstruct. At 10 KB/event and 1000 events/second, this is roughly equivalent to 10K MIPS (or 10K SPECint92). Using this projection plus anticipated increases in data rates, CPU and data resources must be implemented in the following scale:

FY	CPU (MIPS)	Disk (GB)	Near-Line Tape (TB)
96	2 K	100	5
97	10 K	500	150
98	20 K	1000	300
99	30 K	2000	1200

Meeting the usability, efficiency, and flexibility requirements outlined above imposes a special set of challenges for the modest budget and staff dedicated to this operation. The design described below consequently makes heavy use of commercial software applications and standard off the shelf hardware. The selection of hardware and software components has stressed cross-platform capabilities so that components can be replaced and/or upgraded as needed without major redesign of the facility. Project management has stressed the close involvement of users from all three Experimental Halls in addition to Computer Center personnel, who will implement and manage the central mass store, in order to insure that the design meets both the technical and human aspects of the overall requirement.

Factors in the Design of the Data Path

Several factors were evaluated in the design of the data path for both the on-line and off-line operations. In almost all cases, the final decisions represent trade-offs in terms of cost, efficiency, and robustness. With a small staff, plus some input and assistance from physics users, and limited budgets, simplicity is key.

Factors in Designing the On-Line Data Path

All three Experimental Halls use some combination of VME and Fastbus technology to collect raw data from one, and in some cases, two spectrometers. Hall B has developed an event building capability which employs the Asynchronous Transfer Mode (ATM) network technology to collect, sort, format, and compress the data points from each physics event. Details of their algorithm are provided in reference [1]. VME single board computers serve as readout controllers (ROCs) to collect data from the electronic crates; control and data messages are passed in the 53-byte ATM cells over OC-3 connections (155 Mbps) between the ROCs and an on-line SMP (Symmetric Multiprocessing) farm processor (OLFP), a UNIX server. Formatted event data must then be transported to a

mass storage library for eventual replay to the off-line batch farms. Some local backup tape capability is desired both for convenience and redundancy.

Design decisions specifically related to the data path of the on-line physics events include (a) the location and number of robotics libraries required to collect the raw data, (b) the network implementation over which to transmit the information, (c) the number, speed, and capacity of the tape transports to employ, and (d) the number of copies of the raw data to be stored.

Robotics Libraries: Considerations such as redundancy and the need for visual feedback from storage operations led to the evaluation of implementing dual robotics libraries to support the on-line operation. In this model, a smaller robotics library would be located in the experimental area (the "Counting House" where the DAQ systems reside) so that tape storage of on-line information could continue even in the event that network connections to the central site were interrupted. With modern Hierarchical Storage Management (HSM) software, data loaded to the Counting House silo could be migrated in background to a higher capacity central silo used to feed the off-line batch farms. This arrangement, while providing good redundancy and failover capabilities, plus fulfilling the human need to keep the raw data of the running experiment "local", in fact results in one extra copy step, greater complexity in managing the location of the data and in freeing sufficient storage space for real time operations, plus most importantly doubles the cost of the operation. Locating all tapes (including duplicate copies if required) within one central silo (or silo-complex) insures that the data is where it is needed when it is needed and provides a central single point to expand when capacity requires. After evaluation of the options, simpler, cheaper solutions for redundancy and feedback can be implemented with graphical monitoring utilities to provide visual feedback to the researcher, and local disks and lower cost tape drives on the DAQ systems for buffering and emergency archives. This solution does require that some mechanism provide for the uncontested use of central tape drives for on-line operations. This is particularly critical for the high data rates for Hall B, where local buffers could quickly overflow if the real time operation waited on lower priority off-line use of the central drives. The decision to employ an HSM file management approach posed a problem in that most HSM applications do not provide tape allocation capabilities. Consequently the design of a local customized tape staging application must incorporate the capability to insure that a tape transport is immediately available for selected real time processes.

Network Medium: Viable network transports for the raw data include FDDI (Fiber Distributed Data Interface), ATM, HiPPI (High-Performance Parallel Interface), and Fast Ethernet (100BaseT). While the costs of ATM may prove to be lower than the more mature FDDI and HiPPI standards, many vendor offerings are unproven and still groping for a standard. Fast Ethernet (100BaseT) provides both higher capacity and cost effectiveness but may not meet the high speed throughput requirements of Hall B. Decisions regarding switching versus routing must insure that signals from the three halls do not interfere with each other, yet are not degraded by-latency overheads.

Tape Transports: A major design decision was whether to use multiple lower speed/capacity tape transports (possibly DLT) or a fewer number of high end drives (Redwood--11.1 MB/sec., 50 GB cartridges; Ampex--15 MB/sec., 165 GB cassettes, etc.). The IBM Magstar Drive (9 MB/sec, 10 GB cartridges) offered a midrange choice

in terms of cost and capacity, with relatively high end throughput (at least approaching the 10-12 MB/second data stream expected from Hall B). Employing multiple, lower cost drives has the advantage that losing one or even several drives has minimal impact on the overall operation, plus increases the possibility that the researcher can in fact read raw data tapes at the home institution. The disadvantage to this model is the increase in complexity in terms of fanning the data stream out to multiple drives as well as the significant increase in sheer floor space required to store tapes (both in near-line and off-line locations). The use of a higher throughput, higher density tape solution reduces the complexity of the algorithm, reduces the cost of tapes as well as storage, plus provides the throughput capacity required to “catch up” after scheduled and unscheduled interruptions.

Data Copies: It is interesting to note that the cost of generating the 300 TB possible to store in the laboratory’s STK 4410 robotics silo, given site estimates of running costs, is many tens of millions of dollars (more depending on the volume of data collected from the lower intensity halls, Halls A and C, as well as the amount of processing required to produce any reconstructed and/or analyzed results). Consequently, the issue of whether duplicate copies of the raw data should be kept for all experimental runs is truly both a cost and research critical decision. Assuming Hall B produces approximately 50 GB per hour of operation running 125-150 days per year, the annual cost to save one copy of the raw data stream is on the order of \$300K in tape costs alone. The cost, per copy, then is less than 1% of the overall generation cost. On the other hand, \$300K, let alone \$600K (assuming two copies), plus of course the original investment in additional \$100K+ tape units, is a significant impact on tight experimental budgets. A survey of other energy research laboratories indicates that keeping duplicate copies of raw data is by no means universal even in far lower data rate environments, that total loss of a raw data tape is rare, and that the loss of some small percentage of an experiment’s data would be unlikely to affect the overall results. This decision is still under consideration at Jefferson Lab and may be affected as much by budget restrictions as risk analysis.

Factors in Designing the Off-Line Data Path

Using the Hall B estimate of 10K SPECint92 to “reduce” (in many cases, just “reconstruct”) the data from the CLAS spectrometer, the laboratory will require an off-line batch-mode CPU farm consisting of on the order of 50 CPUs ready for production operation during 1Q97. The final configuration of the farm and the supporting software will depend on several factors, including relative costs and performance of a range of processors, size/speed/cost of local node disks and central RAID subsystems, size of input raw data files as well as output files, and the complexity of the software algorithm to coordinate pre- and post-staging of data files with the data reconstruction job. The basic assumption in the design is that the first pass reconstructed data is approximately the same size as the input data. An actual reduction of the output data, to 10% for example, would drastically reduce the overall cost of the implementation, plus have significant impact on the overall throughput of the facility.

The data reconstruction operations on JLab’s data involve a model of “trivial parallelism.” One executable designed for an experiment can be used repeatedly against event after event either in sequence or in parallel to generate reconstructed events. Consequently the design decisions involve at what granularity to fan out events to a series

of CPUs, making the basic assumption that a “pizza-box” style of post-processing will most likely cost less than the use of one, very high end multi-processing system. A PVM-approach (Parallel Virtual Machine), for instance, would use a “master” server to fan out single events to a series of CPUs each running the same code, collecting output back on the master node. Alternatively, blocks of events can be handled in a series of automatically generated batch jobs, with the naming of the output files used as a method to “collect” the results back into sequential order.

During off-line post-processing, raw data files must be retrieved from the central mass store via an automated multi-job generation process that loads required files “just in time” for batch processing and returns output to required locations (tape silo and/or on-line storage). Design decisions specifically related to the data path of the off-line processing include (a) how the researcher will access required files; (b) the algorithm and path used to pre- and post-stage files for the running batch job; and (c) the algorithm and implementation to allocate resources according to laboratory planning.

Data Access: One primary goal in the design is that access to the files associated with specific data runs should be reasonably intuitive to the end user. One method to implement this is of course to use the concept of the UNIX file system itself as a way to catalog files. The use of meaningful directory and file naming conventions then allows reasonable access, even without metadata, to specific file sets. Commercial HSM and other volume management applications support this access mode by implementing virtual file systems where only a portion of the files actually reside on-line. The use of standard HSM-style “file migration”, where on-line water marks and recent use heuristics define which files are maintained on-line, provided one possibility to support the file system for JLab’s experimental data. A disadvantage of this approach, however, is that files to be retrieved must first be “migrated” from tape to the local file system before they can be used. In the case of feeding an off-line post-processing CPU farm, the required location, for performance reasons, may very well be on dedicated central staging areas and/or local batch node disks, as opposed to the “cataloging file system,” thus necessitating at least one extra file copy operation to locate the file where it is needed. A variant of file migration is the use of a file “stub”, or marker to the actual tape location of the file, provided by some file management utilities. In some implementations, restrictions in the relocatability of stubs can pose a problem for expanding, dynamic file systems. In addition to intuitive access to the raw data and related output files, researchers must have the capability to store additional metadata related to both runs and data reduction phases. This requirement, however, calls for a database-oriented capability above and beyond the management of the virtual file system alone.

Staging Algorithms: Probably the most critical decisions for the overall design of the off-line batch farm revolve around how to make the input file, either a raw data file or the output from an earlier phase of data reduction, accessible to the batch job that eventually uses it. The question involves not only decisions regarding synchronization and job priorities, but from a design perspective even the anticipated input and output file sizes and how they may perform in either local or central staging models. Although data files could in theory be directly loaded from the tape silo to local disks on individual farm nodes, two limitations argue for an initial central staging area: (a) limitations in the I/O performance of the individual farm nodes. Although CPU alternatives exist with the required I/O performance, this will mandate higher performance and hence higher cost

systems for the farm; (b) processor/tape utilization. Ideally, CPUs should not be idle while the next data file is loading, and the use of high cost, high performance tape drives should be optimized around efficient staging algorithms. De-coupling the two phases by means of central buffering best accomplishes each goal without compromising the other.

A variety of staging models can be considered. A real argument for smaller input files exists in both the current limitations in many UNIX operating systems for files less than 2 GB as well as the possibility to use inexpensive disks local to the batch node for actual input/output file storage. Considering that a 2 GB raw data file represents less than 3 minutes of beam operation in Hall B, such a limitation involves some level of inefficiency in terms of opening and closing files during the data acquisition process plus dramatically increases the number of associated raw data files. Substituting either a 25 GB (~30 minutes of operation) or 50 GB (~ one hour of operation) size for the raw data file may be more efficient from a DAQ perspective but effectively rules out truly local disk storage from a cost consideration and incurs the performance penalties of NFS or other network file access. Just to complicate the formula are considerations such as the time to “cold start” the farm, the time to “warm start” the farm after a brief interruption (e.g. take advantage of the files already pre-staged), plus considerations of the researcher’s intent for longer term on-line storage of the associated output files. A PVM approach can solve a large file requirement by fanning out events from a large input file to the individual batch node, but network performance must be considered as well as the increased coding complexity for the researcher. Moreover, the entire model changes drastically if data reduction actually accomplishes a significant reduction in output file size during early processing cycles.

Resource Allocation: All resources required during the experimental process are allocated to an experimental collaboration according to laboratory planning, from the hours of scheduled beam time to the staging of input files for allocated use of the off-line batch farm. The bottom line for researchers, however, is how much CPU time they are getting to post-process the data collected during their on-line operation. The design of the algorithms to “feed the farm” must provide the best overall site throughput as far as quantity of data processed, yet accomplish this within the guidelines of allocation strategies mandated by the laboratory. In this central silo model, the design of tape drive allocation and staging algorithms must first of all meet the requirements to insure the uncontested use of storage mechanisms by real time operations. Beyond this, the focus must be on a fair share allocation of the farm resources; tape staging serves only the purpose of fueling the correct mix of jobs. The challenge in a fully automated system is at what point in the process to implement a fair share algorithm to achieve the overall allocation strategies of the laboratory--if during the tape loading stage, how can we determine in advance which files should be loaded to achieve the desired mix in the set of running jobs; if during the job submission stage, how can we insure that the required files will be available at the time to run? And what percent utilization can we hope to achieve with high performance, high cost tape transports? The prototyping and testing of these various models will be essential in selecting the optimal design given the specific CPU, I/O, and network parameters at hand.

Current Implementation and Status

The On-Line Model

The implementation in progress to handle the data flow from Jefferson Lab's experimental program includes the selection of the Open Storage Manager (OSM, Computer Associates, International) HSM software integrated with a StorageTek robotics library and Redwood helical scan tape drives. Fast Ethernet provides the base for the experimental network, carrying all data for the 1-2 MB/sec data streams of Halls A and C (See **Figure 1**, "High Speed LAN"). The use of a Fast Ethernet switched architecture provides enhanced performance by protecting raw data streams from outside interference. A developmental ATM switch will be used to prototype the potential use of ATM for the farm network fabric. The higher intensity data collection of Experimental Hall B (as well as potentially some experiments in Hall A) will use the network for control signals only, moving the raw data via dual-ported Fibre Channel RAID connected to both the Experimental Hall event-building CPU and the Computer Center data server. The StorageTek Redwood drives were selected for both their high performance (current benchmarks indicate an 11.1 MB/sec. throughput) and high density (50 GB cartridges are available). Due to the large volume of data anticipated, effectively ruling out the option of performing off-site data reduction, the advantages of these high performance/high density options for Jefferson Lab outweighed the disadvantage that the home institutions will most likely not be able to build similar environments. Furthermore, there was some advantage to the tape cartridge design of the StorageTek (STK) Redwood transports which contain no takeup reel, cutting the storage space required in half. One factor in the selection of the OSM software was the existence of a customized extension called *OSMcp*, developed at the Deutsches Elektronen-Synchrotron (DESY) which solved the relocation issue with stubs as well as provided exactly the data flow model required: the direct copy to a designated location (either disk-to-tape or tape-to-disk) using a stub file as a reference only. Furthermore, by use of UNIX protections on the stub files alone, the integrity of master copies of the raw data files, which should never be modified and rarely deleted, can be protected.

Figure 2, "High End Data Flow", illustrates the path to be taken by Hall B experimental data. Raw data collected during the Fastbus/VME-based data acquisition process will be channeled through an ATM switch to the Hall B SMP system (Data Acquisition Symmetric Multi-Processing, DA-SMP). This system will use the multiple input data streams to build events, locating the raw data files created on the dual-ported Fibre Channel RAID subsystem. The DA-SMP system will toggle between two separate RAID partitions and as it fills one, will signal the Computer Center data server (CC-SMP, a Sun

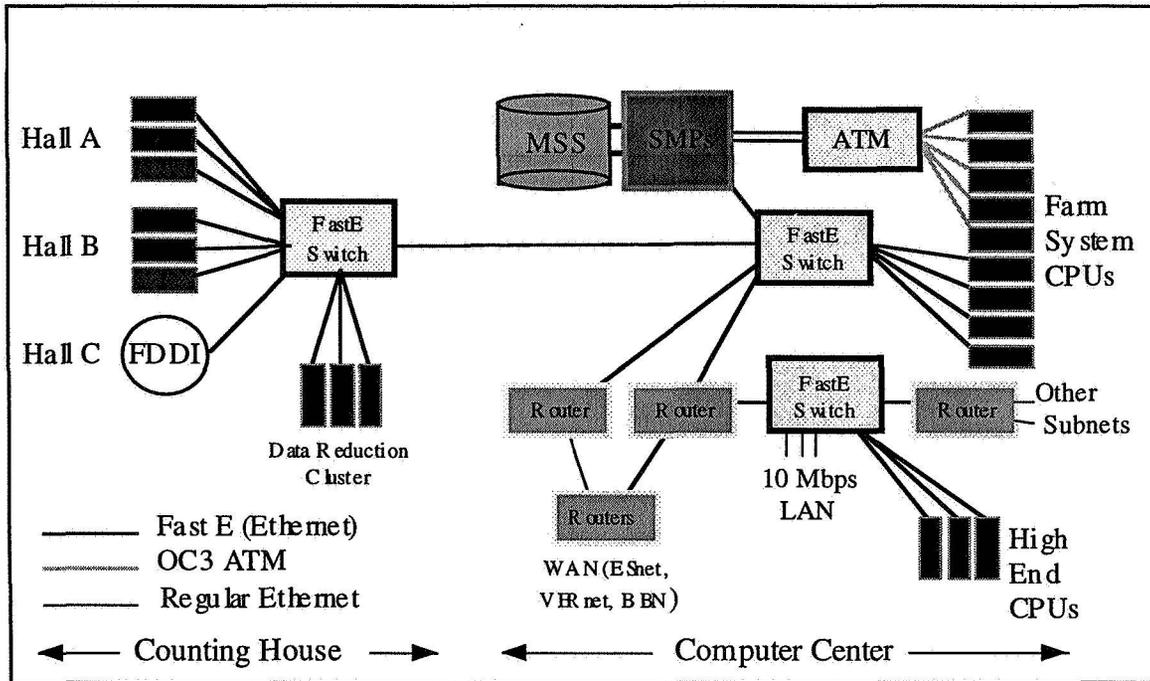


Figure 1: High Speed LAN

Enterprise 4000, 1 km away) via a network call, to begin moving the files on that partition to the STK silo. The DA-SMP will then resume data collection on the alternate RAID partition. This alternate writing then reading of the RAID subsystem, isolates the data transport from the network, and effectively enhances the throughput of the two data storage servers in that data does not have to pass twice through each system as required for a network-based transfer. Current testing of this model using a lower performance Sun 1000, including the use of the OSMcp utility, has resulted in transfers in the range of 9 MB/second, closing in on the performance goal of the maximum rating for the Redwood drives, 11.1 MB/sec. Configuring high performance components of the architecture (Redwood drives, RAID, network interface) with a separate SBus is expected to yield the remaining required throughput. While initial testing has involved only one RAID subsystem divided into two partitions and one Computer Center storage CPU, the solution can scale with the addition of multiple separate RAID units and storage servers. The CC-SMP will use the OSMcp utility to move the raw data directly to tape, leaving "stubs" in predefined directories corresponding to the specific Hall/Spectrometer and experiment. The stubs, which appear to the user as regular UNIX files, are essentially pointers to the actual file locations on tape, as stored in the OSM database. The OSM software currently interfaces to two Redwood tape drives retrieving tapes from an STK 4410 robotics library (6000 tape maximum capacity, 140 robotic exchanges per hour). With the 50 GB cartridges available for these drives, the current maximum capacity of the silo is 300 TB. An aggregate of 30 MB/second of data throughput must be supported for post-processing to keep pace with the rate of new data collection (e.g. 10 MB/sec each for: new data in; raw data out; processed data in). Depending on the final efficiency factors realized, 6-8 Redwood tape transports will be required to support this level of throughput.

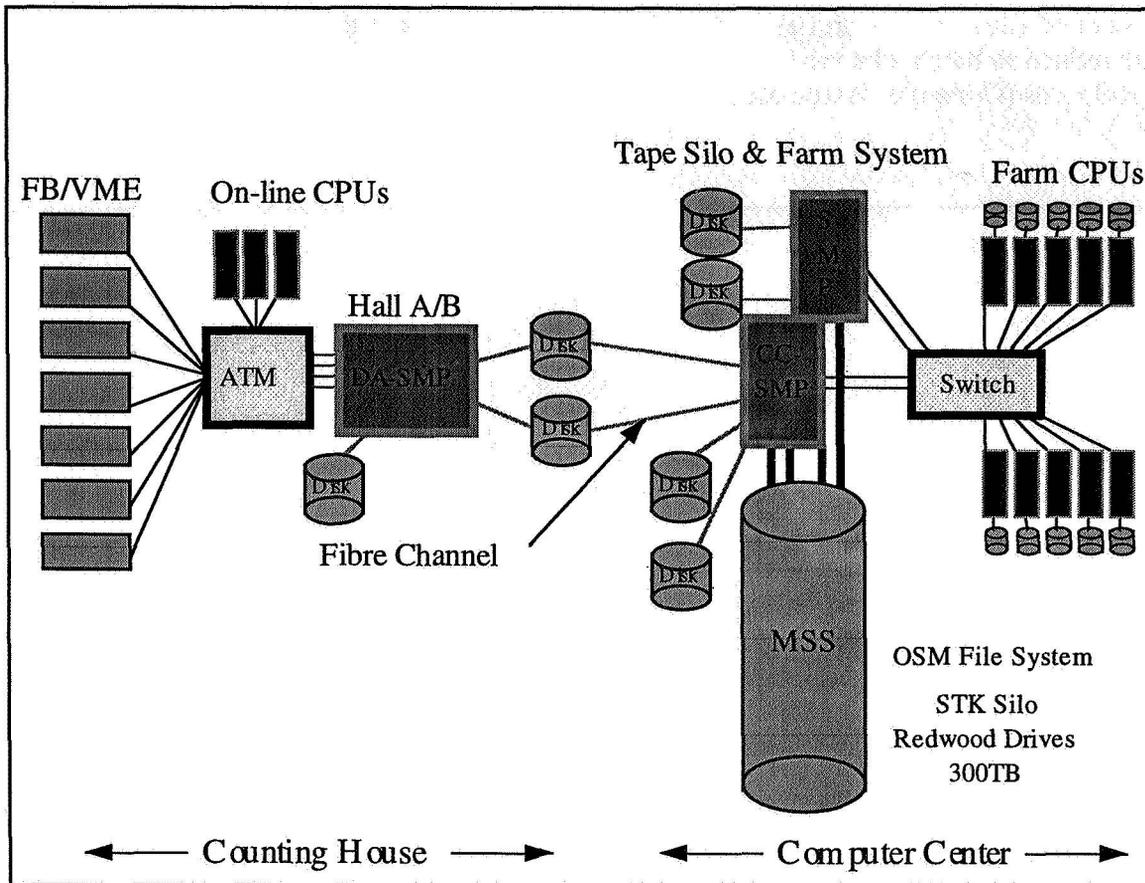


Figure 2: High End Data Flow

Halls C and A will most likely use a network-based data path, also making use of the OSMcp capability. A simple, automated process, parallel to the existing local Exabyte tape copy procedure, has been implemented for current Hall C production. The procedure transfers the data (via remote copies as opposed to NFS) from the Hall C Data cluster to a staging area directly on the Computer Center storage server, and then subsequently OSMcp's the files, leaving stubs in a separate file system. A user utility provides the reverse capability to retrieve data files for post-processing on the existing Data Reduction cluster (3 HP 9000/735s) located in the Counting House. Exabyte and 4mm DAT autoloaders available on this cluster currently provide an export capability for the experimental user.

The Off-Line Model

The Load Sharing Facility (LSF, Platform Computing) has been selected to provide the batch management software base for the off-line CPU farms. The initial design work has begun with the concept of a simple round-robin approach, using LSF to channel jobs to an array of "pizza-box" CPUs (low end, low cost, headless UNIX systems), each with a local disk to hold executable and output files, most likely reading input files from a central tape staging area. While the use of PVM is not ruled out entirely, a coarse grained parallelism with each individual job (submitted, however, in batches) processing one complete run is attractive due to the simplified coding and testing required on the part of the experimental user. A locally developed customized application to track runs and

associated files, to pre- and post-stage files in coordination with OSM, and to generate data reduction batch jobs via LSF is under design and will most likely make use of the OpenIngres (Computer Associates) relational database software.

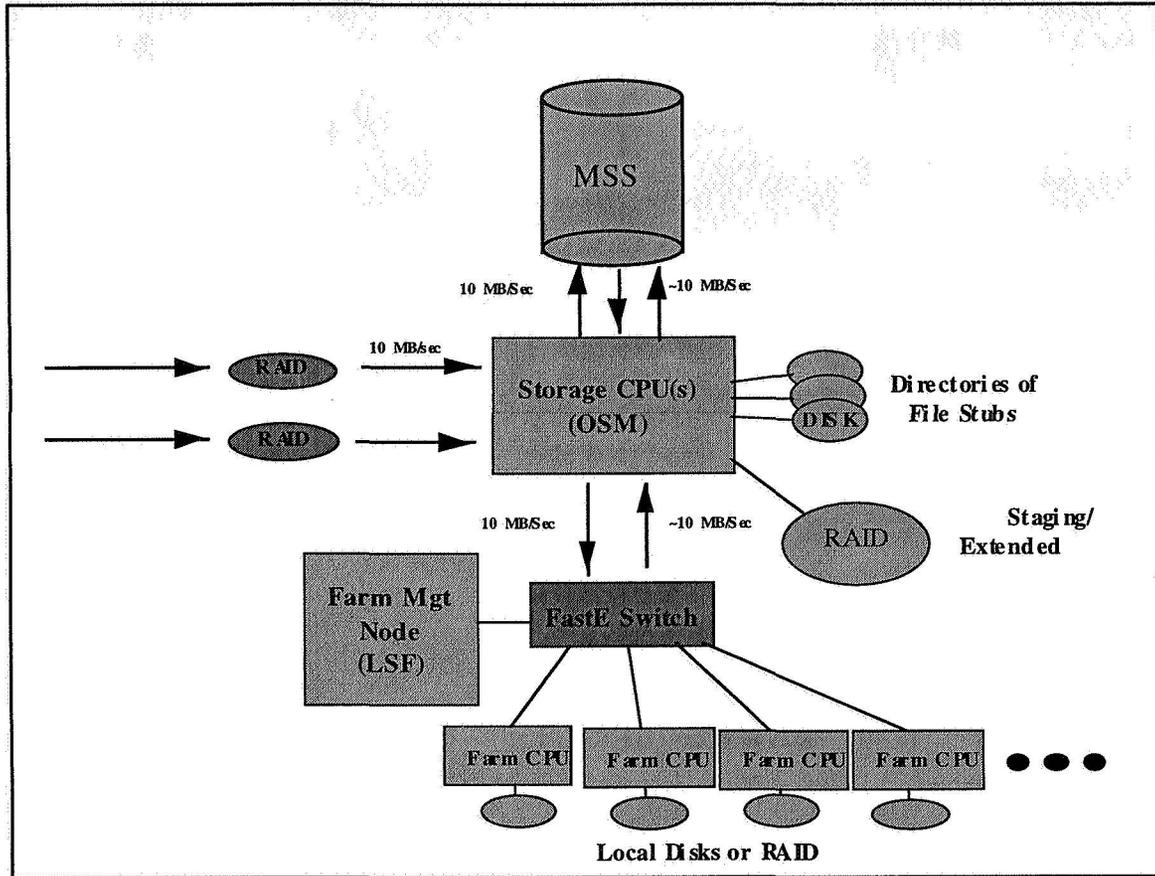


Figure 3: Batch Farm Data Flow

Figure 3, "Batch Farm Data Flow", illustrates the general path of data to and from the data reconstruction CPU farm. Batch jobs, managed by LSF, must coordinate both the retrieval (reverse OSMcp) of raw data and other auxiliary files (calibration files, trigger maps, executables, etc.) as well as the storage of generated output files. OSM is currently limited to a maximum file size of 2 GB. This, plus the economics of local disk storage mandates a maximum raw data file size of 2 GB at the current time. Work is in progress to simulate a variety of alternatives of central and/or local file staging to determine the optimal model considering both data throughput as well as cost. Critical calculations include the number of high end tape transports that will be required to provide both the necessary redundancy for on-line operations as well as sufficient aggregate throughput to pre- and post-stage data for the off-line farms. The right mix and volume of central and local RAID and/or striped disks must address cost, performance, and throughput, as well as simplicity in the algorithm to manage the farm.

Challenges Ahead

Several major challenges face the development team in the near future; the first is to develop the customized software extensions to the file and batch management layers provided by the commercial applications, OSM and LSF. A task force composed of Computer Center staff, Hall, and User representatives is nearing the end of the requirements phase for such a suite of applications. The functionality envisioned will replace manual and even electronic methods to manage the progress of an experiment through the pipeline required from raw data file, through reconstruction phases, to a final Data Summary Tape or even Micro-DST, and provide Jefferson Lab's collaborations with web-accessible, graphical tools to manage and track the entire reconstruction process. Results of prototyping the various data staging models will be critical to the final design of both the disk pool areas and staging algorithms. The developmental milestones include completion of both requirements and design reviews by October, 1996, and the release of a Beta version in early 1997. The procurement of required farm hardware will be concurrent to the coding and implementation phases of the software development. Production experimentation for Halls C and A, and early calibration runs for Experimental Hall B are expected to remain within the data handling capabilities of the existing script-based methods to store and retrieve sets of files from the central mass storage library.

Efficiency and scalability are challenges for every massive data handling operation. Experiences from other similar data collection and replay environments suggests that planning should anticipate overall efficiency rates of no more than 50%, building in sufficient tape, disk, and CPU resources to avoid bottlenecks at any point in the operation. The hard reality in today's research environment is that this approach costs real money--money that from the experimentalist's perspective reduces the amount of research that can be done. In JLab's setting, the hundreds of thousands of dollars that can be saved by tight management of high end disk and tape devices translates into new spectrometer equipment and additional beam time -- more "physics"! The goal, consequently, must be to develop finely tuned, smart systems that can anticipate scheduled requirements, manage resources closely, recover quickly from interruptions, and scale by modular upgrades. Right now, Hall B anticipates a data rate of 10-12 MB/second. Given history, that no doubt will ramp up not down. With the potential of two or three Experimental Halls running simultaneous, high intensity experiments, the initial infrastructure must anticipate an eventual doubling or tripling of the original design goals. Modularization of both hardware and software implementations must allow the upgrade and/or replacement of any one component, from the addition of multiple storage servers and farm nodes, or expansion of on-line storage pools, to the possibility of replacing magnetic-based near-line solutions with other future technologies. The useful lifetime of Jefferson Lab's raw data sets may be 10 years or more. In today's technology, that can represent two or even three implementation life cycles. Today's solutions must prepare not only for tomorrow's requirements but also lay the framework to build with future tools.

Conclusions

Designing for large data handling projects in today's computational environments involves the coordination of network design, tape and disk pool modeling, simulation of processing flows, as well as the detailed consideration of end user requirements and

interfaces. The goal of Jefferson Lab's design to support the experimental data handling requirements of the laboratory is to employ modular hardware and software solutions that will scale to meet anticipated future requirements. We have chosen to employ commercial software foundations extended by locally developed applications to coordinate different components of the system. While the current design will provide the immediate capability to handle all facets of collecting and post-processing a new data stream of 10-12 MB/second, our objective must include the scalability to survive not only considerable expansion of the anticipated load, but significant changes in the technological alternatives available.

Acknowledgments

We would like to acknowledge the many individuals at Jefferson Lab whose combined contributions have resulted in the implementation in progress to provide the data handling required for the experimental program; in particular: Dave Doughty (Christopher Newport University), Lan Fan (College of William & Mary), Larry Dennis (Florida State University), David Hibler (Christopher Newport University), Andy Kowalski (JLab), Sandy Philpott (JLab), Dave Rackley (JLab), Roy Whitney (JLab), and Elliott Wolin (College of William & Mary). This work was supported by Department of Energy contract DE-AC05-84ER40150.

References

- 1 D. Doughty et al, "Event Building in Future DAQ Architectures Using ATM Networks", Proceedings of the Computing for High Energy Physics Conference (CHEP95), Rio de Janeiro, Brazil, 1995.

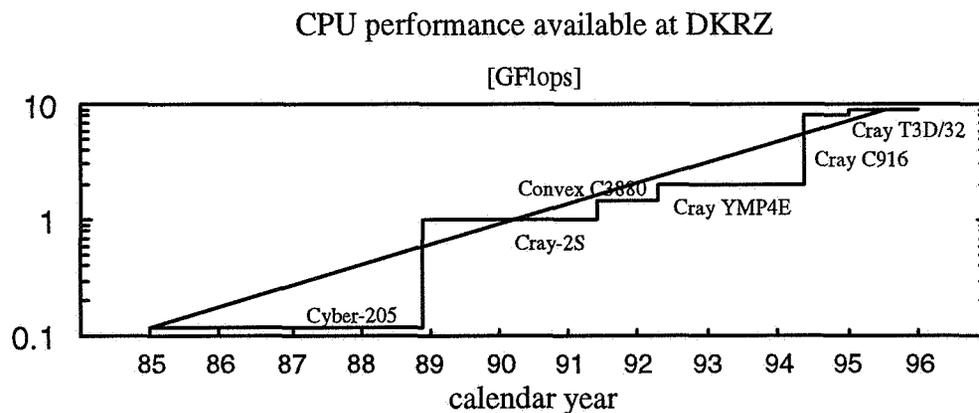
DKRZ Workload Analysis

Hartmut Fichtel
 Deutsches Klimarechenzentrum GmbH
 Bundesstr. 55
 D-20146 Hamburg
 Germany
 e-mail: fichtel@dkrz.de
 phone: +49(40)41173 220
 fax: +49(40)41173 174

Introduction

DKRZ is a primarily federally funded institution to provide the German climate research community with the necessary computing resources to perform numerically highly complex climate simulations. This task implies a virtually unlimited requirement for installed compute power at the outset; moreover, directly correlated with available compute power are the corresponding data services requirements, both in terms of static capacity and dynamic data access.

The following diagram shows the development of installed compute power at DKRZ from 1985 to 1996.



This local development seems to be well synchronized with the overall technological development of supercomputer technology showing an average increase of almost a factor of two every 2 years.

Tightly coupled to the compute power dedicated to climate simulations is the data rate in terms of long term storage. As a general rule, it turns out that **1 Flops compute power** (sustained) generates **1 KByte of data** per year which is of sufficient interest to justify long term storage. So, the above diagram can also be interpreted as the development of actual annual long term storage rates if the y-axis is replaced by [TByte].

There are two more interesting rules of thumb defining the overall correlation between available compute cycles and the resulting requirements for the data management system which have been remarkably constant in the past. The second rule refers to data

generation rates as opposed to long term storage: roughly 25 % of the total data generated will be stored for less than a year and thus does not contribute to long term storage requirements. These rates refer to climate model output only which is the dominating sort of data at DKRZ. Other types (e.g. observational data for model validation, general system backups etc.) display different characteristics. The third rule defines the resulting overall data access requirements: for every byte generated there will be 3 bytes accessed, so model data turns out to be relatively active.

With these three rules not changing and the expectation that continuous funding will be available, it is rather easy to estimate the future requirements for the data management system which is summarized in the following table. Actually, the data intensity will partially change: the major existing applications double their storage intensity by decreasing the archival interval from 12 to 6 hours simulation time. Other uncertainties result from new applications, e.g., atmospheric chemistry.

	95	96	97	98	99	2000
CPU performance [GFlops]	9	9	20	20	35	50
data generation rate [GByte/day]	40	80	150	150	300	400
data archival rate [TByte/year]	10	20	40	40	80	100
required data archival capacity [TByte]	20	40	80	120	200	300
average data moved [GByte/day]	160	320	600	600	1200	1600
average transfer rate [MByte/s]	1.6	3.2	6	6	12	16
required peak transfer rate [MByte/s]	16	32	60	60	120	160

The immediate conclusions are that current technology supports both the required robotic capacities and transfer rates for disk and tape devices. On the other hand, it has to be noted that current technology fails for required network performance and in particular for the aggregate performance of the existing migration software.

Detailed Statistics

Reasonable estimates for future data management system requirements can be made from some global parameters and the rules described if the applications are well known and do not change considerably over time. These facts are insufficient, though, both to define the detailed structure needed for future systems and to identify possible bottlenecks in existing ones. More details are needed for these purposes.

Central file service at DKRZ is based on the well known UniTree product running on a Convex C3800 hardware platform. The built-in statistics and logging features of standard UniTree are relatively poorly developed, so some additional instrumenting code has been added to generate timestamped event-driven messages to analyze the dynamic behavior of the system. About 8 MByte of raw data thus generated per day is preprocessed to combine the various event-driven messages into a single transaction record resulting in 2 additional MByte per day of compressed transaction information.

The logging and analysis concentrates on these three key areas of the system:

- client interface via the network
- disk cache
- tape devices and robotic systems

Client interface

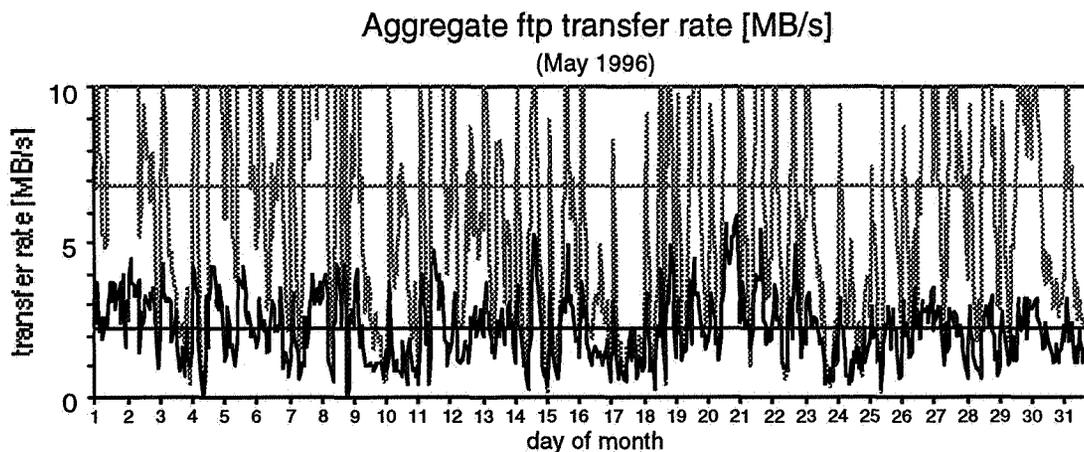
The client interface for normal data access is restricted to **ftp** since both **rcp** and **nfs** (besides the well known performance and security deficiencies of **nfs**) lack the ability to adequately control the access to the Unitree families which is required to choose the proper tape technology typically as a function of file size and possibly other file attributes. So it was decided to modify **uftp** to log the major transaction events along with the relevant parameters:

- start of transfer and possibly cache hit
- stage to disk if necessary
- end of transfer

The relevant parameters additional to the time stamp are

- ftp operation code
- path name, capability, user and group id
- family, number of copies, host and network
- file size and transfer times (disk and possibly tape)

This collected information allows a variety of different parametric statistics like number of requests and data transferred per given time interval with associated variances. An example is the following diagram which shows the **ftp** access pattern with the number of parallel transfers and the effective transfer rate averaged over one-hour intervals.



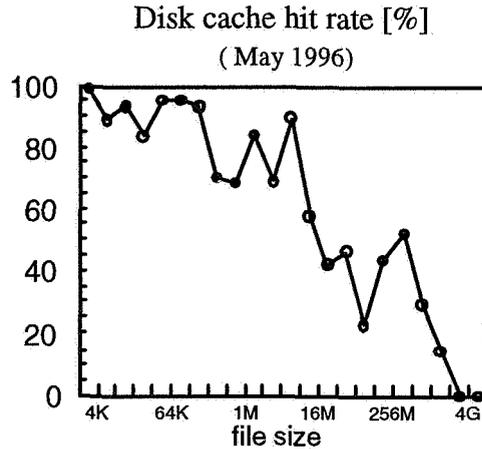
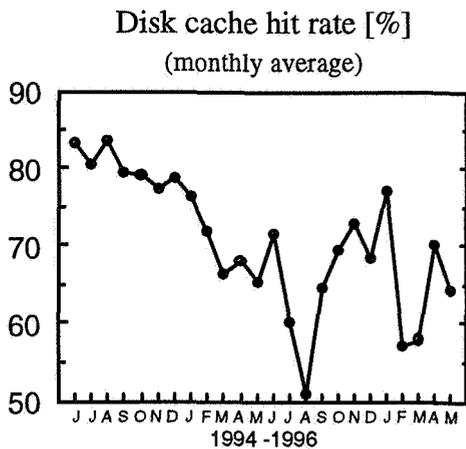
The 2 straight lines are the averages per month with 6.8 parallel accesses and a total of 5.7 TB transferred by 116K requests. If the time interval is shortened to ever smaller

values it can be expected that due to the bursty nature of the traffic particularly in prime times the observed peak rates will reach a maximum either defined by the maximum load or the performance limit of the system. The current hardware platform consists of a Convex C3840 with 1 GByte central memory, 12 disk channels (IPI and SCSI), 14 tape channels (BMX and SCSI) on the server side. HiPPI-switch connections to the major clients, and typically no disk bottlenecks on the client side. In this hardware environment the highest burst rate ever noted was around 16 MB/s which gives rise to the suspicion that the current overall bottleneck in terms of performance is induced by software overhead. This approach can generally be nicely used for benchmarks during the running system without shutting out user operations totally, e.g., to validate hardware or software upgrades and reconfigurations.

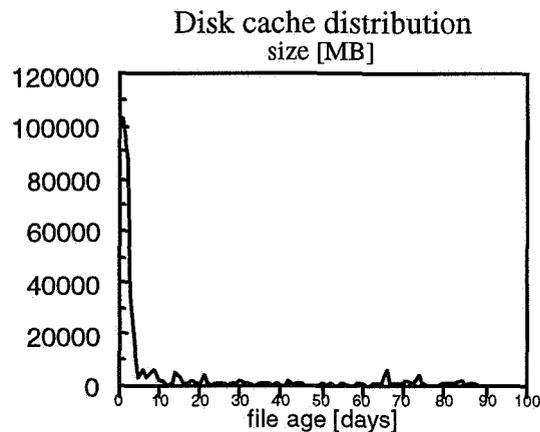
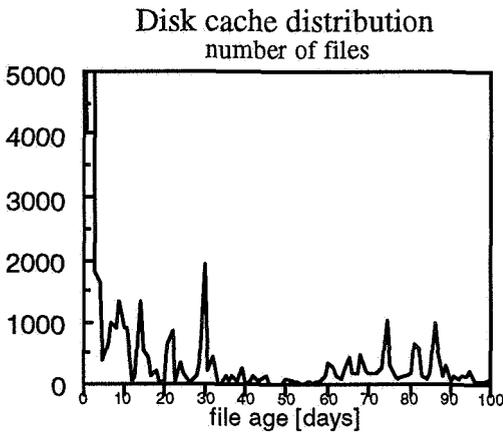
It should be noted though that averages over larger time intervals are only of limited value since the frequency distributions are far from normal. It is typical for mass storage systems that the number of transactions is dominated by small files, but the bulk of the data flowing is caused by a relatively small number of large transfers. The specific distribution of course is application dependent. Additional to plotting system parameters by file size it is worthwhile to also study other distributions, e.g. by host and network.

Disk cache

Probably the most important factor to performance is the disk cache since the cache hit pattern dominates the turnaround time as seen by client access requests. The overall disk cache hit rate development over time for the last 2 years is plotted in the following diagram. Due to the increasing load on the system this global cache hit rate has been decreasing almost monotonically from 85% down to 50% over the course of a year which tendency could be reversed by a major disk cache upgrade in mid 1995. In two steps 48 Elite-9 SCSI disk drives with 8 SCSI channels have been added to the existing IPI devices resulting in a total disk capacity of 550 GByte. It is interesting to note that this hardware upgrade did not take immediate effect but took several months instead. The reason is that it is easy to fill up the cache immediately, but it takes very long to put the files on disk which have the most positive influence on the overall hit rate, i.e. as many small active files as possible. This is also the explanation for the backward developments e.g. in February 1996: due to drive failures the contents of complete partitions have been lost with a resulting loss of a large number of small file copies.



Although the global hit rate has decreased to 65% the situation is still much better for small files as the hit rate versus file size distribution shows. The next 2 diagrams show a snapshot of the cache file distribution (both number of files and aggregate size as function of age). Although the Unitree disk cache purging algorithm is not very versatile it turns out that the old files do not add considerably to the total size since they tend to be small.



Another valuable information to derive from the transactions logged with the capability identifying single files is the working set which denotes the locality of the user load.

Archival storage

The core of the mass storage system is the archival storage typically consisting of tape devices integrated into large robotic capacities. Tape technologies differ considerably in their technological parameters and cost, and furthermore the tape handling software in question may or may not take advantage of the various features they provide. It is thus not sufficient to compare different tape technologies just by their published features or even by comparative tests under the native operating system, since mass storage systems often handle tape devices independently and in different fashion. In particular it is not sufficient to compare streaming transfer rates because for realistic file distributions

transfer is not the only fraction of the operation contributing to elapsed time, and it may not even be the dominating one.

Currently there are 2 robotic systems and 3 tape technologies in production use at the DKRZ: STK 4400 libraries with both 3490 and D3/Redwood drives as well as Metrum RSS-600 with VHS drives. In order to evaluate this technology logging information from **tapesrvr** and **pdmsrvr** define the various events constituting a complete tape transaction:

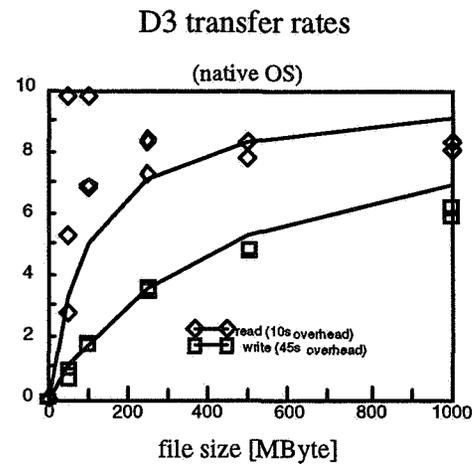
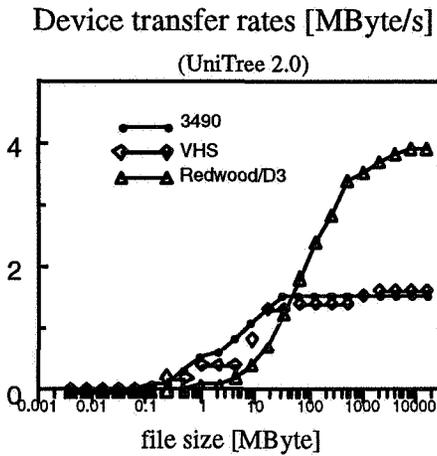
- receipt of request from **disksrvr**, possibly wait time in queue
- mount request issued to **pdmsrvr**
- tape positioned to requested block offset
- transfer complete, possibly „lazy wait“ on drive
- dismount request issued to **pdmsrvr**
- tape dismounted

The parameters additional to the timing information of the various events listed above include:

- stage or mig request
- client or repacker request
- capability
- family, location, copy, and possibly fragment
- VSN, tapetype, physical unit
- block offset
- file size

The analysis of this device oriented information enables a wide range of useful statistics, like the impact of internal reorganization (repacking) on overall system performance for client initiated requests, device specific performance for mount/dismount, positioning and transfer operations, queuing states for available devices, effects of system parameters like the „lazy wait“ feature on mount/dismount overhead, and many more.

An example is the following diagram which shows tape transfer performance as measured in the running Unitree system for 3490, VHS, and Redwood/D3 tape technologies as a function of request size. It turns out that VHS is relatively performing best by reaching 80% of the streaming rate already for moderately sized files of 30-40 MByte, whereas 3490 does not exceed about 50% of the peak rate. Relatively disappointing is D3 performance which saturates at about 40% of the streaming rate needing even larger files in the GByte range. It is assumed that the reason for this poor performance again is caused by the inefficient server-server communication scheme since similar tests under the native operating system show that the D3 streaming rate of almost 11 MByte/s can indeed be reached for files of 1 GByte and more.



The different read/write behavior is caused by the overhead of tapemark processing which is higher for writing files. The measured write results follow the theoretical curve very closely, the read results for small files are perturbed by the read-ahead capability of the D3 device, a feature which cannot be typically made use in a mass storage system.

Conclusions

The UniTree product as it is released emits a rather large amount of logging information into various log files, but this data typically is meant for operator information in difficult operational situations or directly for debugging purposes. It is strongly advised that the existing logging functionality in the standard release is advanced by adding messages with relevant parameters for every major event in the course of executing file or device oriented requests as described.

This upgrade would be relatively easy in terms of implementation effort. The benefit is a complete sequence of transaction descriptions generated by external user requests or by internal administration commands. This collection of transaction records can be used for intensive statistics and performance evaluations. It is furthermore a perfect source to drive realistic system simulations to study the effects of possible hardware or software changes.

Acknowledgments

This project has been initiated and in most parts designed by DKRZ. The implementation of actual changes in the UniTree server code has been performed by Ingolf Sessler, a consultant to the Hamburg office of Convex Computer. We appreciate the open and ongoing cooperation.

A New Generic Indexing Technology

58-82

83190

Michael Freeston

CCSE/Alexandria, University of California

Santa Barbara CA93106

freeston@alexandria.sdc.ucsb.edu

tel: 805-893-8589

fax: 805-893-3045

Abstract

There has been no fundamental change in the dynamic indexing methods supporting database systems since the invention of the B-tree twenty-five years ago. And yet the whole classical approach to dynamic database indexing has long since become inappropriate and increasingly inadequate. We are moving rapidly from the conventional one-dimensional world of fixed-structure text and numbers to a multi-dimensional world of variable structures, objects and images, in space and time.

But, even before leaving the confines of conventional database indexing, the situation is highly unsatisfactory. In fact, our research has led us to question the basic assumptions of conventional database indexing. We have spent the past ten years studying the properties of multi-dimensional indexing methods, and in this paper we draw the strands of a number of developments together - some quite old, some very new, to show how we now have the basis for a new *generic* indexing technology for the next generation of database systems.

Multi-dimensional indexing

There has been no fundamental change in the dynamic indexing methods supporting database systems since the invention of the B-tree twenty-five years ago. And yet the whole classical approach to dynamic database indexing has long since become inappropriate and increasingly inadequate. We are moving rapidly from the conventional one-dimensional world of fixed-structure text and numbers to a multi-dimensional world of variable structures, objects and images, in space and time.

But, even before leaving the confines of conventional (i.e. relational) database indexing, the situation is highly unsatisfactory. In fact, our research has lead us to question the basic assumptions of conventional database indexing. The concept of a set of one-dimensional primary and secondary indexes, tuned to a particular query pattern, emerged in the days of menu-driven applications. This concept has not evolved in any way to match the flexibility of 4GL query languages. And today two new factors make such tuning increasingly impractical: internal system inferencing - where even the system itself cannot predict the

progression of queries - and an increasing number of very large scale applications where the overhead of periodic re-indexing is unacceptable .

Of course, menu-driven applications remain, but there are many conventional database applications today where much more flexibility is needed. Users who browse through a database in an unpredictable way find system performance equally unpredictable. We believe that:

1. index design should be guided by a natural principle which is intuitively understood by users: the more information which is given to the system to guide a query, the faster should be the response.
2. the *average* response time to the set of all possible instantiations of a query is a much better guide to user acceptability than the *fastest* time obtainable with one particular instantiation pattern.

To achieve this kind of performance flexibility without incurring massive additional update overheads and increased software complexity requires some form of multi-dimensional indexing. Most recent research on multi-dimensional indexing has been motivated by the need for better spatial access methods. This has tended to obscure the much more general attraction of multi-dimensional indexing, which was clearly appreciated by earlier researchers in the field [1] [2].

The original inventors of the B-tree [3] already had the next step in mind: an index which would generalize the properties of the B-tree to n dimensions i.e. an index on n attributes of a record instead of one. Ideally, such an index should have the property that, if values are specified for m out of n key attributes (a partial match query), then the time taken to find all the records matching this combination should be the same, whichever combination of m from n is chosen. The attraction of such an index is that it narrows the search space evenly over all the indexed attributes, and so behaves as if there were an index on each attribute - while avoiding all the update complication of secondary indexes. It also greatly reduces the need to re-index - or multiply-index - large data sets on different attribute combinations.

To achieve this, the index must be symmetrical in n dimensions. There is no longer a directly defined ordering between the individual records according to their (single key) attribute values. Each record must be viewed as a point in an n -dimensional data space, which is the Cartesian product of the domains of the n index attributes. An n -dimensional generalization of the B-tree must recursively partition this data space into sub-spaces or regions in such a way that the properties of the B-tree are preserved, as far as is topologically possible. Specifically, the number of nodes encountered in an exact-match search of the tree (assuming no duplicates), or a single update, should be logarithmic in the total data occupancy of the tree; and the occupancy of each data or index region should not fall below a fixed minimum.

The challenge has therefore been to generalize the B-tree to n dimensions *while preserving all of its worst-case characteristics*. But a solution has proved very elusive. Alternative approaches of every kind have been proposed (see, for example, [4] [5]), but all have been vulnerable to pathological cases.

In the past this was largely an academic issue. But now it is becoming a matter of life and death. Large databases control mission-critical and non-stop applications of every kind - from fly-by-wire aircraft control systems to electricity supply load balancing. There is no room any more for systems which work very well most of the time. They *must* be completely predictable *all* the time.

This consideration provides justification enough for commercial database companies to cling to the B-tree. It may be inflexible, but it has the essential qualities of being totally predictable and fully dynamic. Whatever advantages an alternative may have, it must demonstrate these same qualities before it has any hope of replacing the B-tree. For these reasons, and a number of others - even more compelling - which we give below, we believe that our recent discovery of a general solution of the n -dimensional B-tree problem [6] is a major breakthrough in dynamic indexing techniques.

Principles of a new generic indexing technology

As a result of ten years research in this area, we have come to appreciate the fundamental importance of a solution of the n -dimensional B-tree problem. This is essentially because *any* data structure which can be expressed as a tree can be represented as a point in an n -dimensional data space at the deepest level of a set of recursively nested data spaces i.e. data spaces containing points which represent data spaces at the next deeper level of recursion.

For example, conventional relational n -tuples can be represented as points in a single, n -dimensional data space. But if the relation contains a nested attribute, then the discriminator which distinguishes between instances of the nested attribute is the index key to the data space of nested attribute instances. This observation has been the basis of our long-standing hypothesis that an indexing system could be developed of sufficiently wide applicability to deserve being called *generic*. It has been the prime motivation for our attempts to develop a multi-dimensional index technique based on the following principles:

1. the index should be fully dynamic i.e. performance should not degrade with time;

2. performance should be fully independent of the data distribution;
3. the exact-match access path length, and times for single insertion, update and deletion should be at worst logarithmic in the size of the data set;
4. there should be a guaranteed fixed minimum in the ratio of the size of the data set to the size of the memory needed to store it;
5. there should be a fixed ratio between the maximum size of the index and the size of the data set (and this ratio should be $\ll 1$);
6. the index should represent a recursive partitioning of the data space (and *not* a recursive partitioning of the points in the space);
7. the partitions should be regular binary partitions of the *domains* of the data space;
8. the partitioning scheme should allow one partition region of the data space to *enclose* another;
9. the representation of the partitioning scheme should allow variable-length, binary keys;
10. the index keys should be generated dynamically.

The first five of these principles are satisfied by a B*-tree. Although it has never been proved that these principles cannot be satisfied by some data structure other than a tree, no such structure has yet been devised (with the possible exception of the hB-tree [7] which, strictly speaking, is not a tree, but a directed acyclic graph). So, in practice, we assume that these principles dictate that the basic indexing structure should be a 'grow and post tree' [8].

Principle six is not satisfied by a B-tree, which recursively partitions the *points* in the data space, not the data space itself. This choice may not be immediately obvious, since there are simple and effective techniques for mapping an n-dimensional point on to a linear order - using, for example, Z (Morton, or Peano) order [9] . Then an ordinary B-tree can be used, and the worst-case characteristics of the B-tree are automatically inherited. There is the additional practical advantage that it can be immediately applied in any system which supports B-trees.

But this approach is too restricted for a generic technology, which must be able to efficiently access extended spatial objects as well as points. The rectangular cover of an n -dimensional spatial object can always be mapped to a $2n$ -dimensional point, but neighborhood relationships are lost in such a transformation. An alternative approach [9] resolves the object cover into a set of subspaces, each represented by a unique (e.g. Peano) code. This allows a recursive linear partitioning of these subspaces, in the same way as for points in a conventional B-tree. But the method violates principle three because, in general, the update of a single object cover requires the update of all its constituent parts. This introduces the uncontrollable access and update characteristics we want to avoid. (Other well-known methods such as the R-tree and the R+-tree suffer from similar problems).

There are two other major reasons for the decision to partition the data space itself, rather than the objects within it. The first is that it is otherwise not possible to contract the representation of a sparsely occupied dataspace to a set of occupied subspaces. Comparative studies by [5] have clearly shown this to be a very significant factor in the efficiency of spatial range queries.

The second is that, in a data space representation, each of the subspaces in the recursive sequence enclosing a point or spatial object can be defined by a key which is a prefix of that of the next subspace in the sequence. This is the motivation for principle seven: by adopting regular (or *strict*) binary partitioning of the data space, each subspace can be assigned an extremely compact binary key with this prefix property. In fact, a much more compact representation than this is possible, since each subspace can be uniquely represented by a key which defines it *relative* to the subspace which encloses it at the next higher recursion level.

It is therefore possible to map the recursive partitioning of the data space to a tree-structured index in which:

- (a) index tree level l corresponds to recursion level l .
- (b) each branch node represents a subspace s , and the set of keys in the node represents the set of subspaces into which it is partitioned, *relative* to s .
- (c) the leaf nodes of the tree represent the subspaces into which the data space is directly partitioned. The full key corresponding to any one of these subspaces is formed by concatenating the keys of all the subspaces which enclose it along the path from root to leaf of the tree.

If this tree can be made to have the properties of a 'grow and post' tree, then it will have further valuable properties:

- (d) the full key defining a leaf node (subspace) does not need to be any longer than is necessary to differentiate it from the keys of all other leaf nodes.
- (e) when a leaf node overflows and splits, as the tree grows, then the full key of at least one of the resulting nodes will grow in length.
- (f) as the tree grows, common key prefixes will be promoted upwards through all levels of the tree.

The choice of regular binary partitions of the domains of the data space results in a partitioning scheme very closely related to that of quad-trees [4] . A criticism which has often been leveled at quad-trees is that they are not 'grow and post' trees, and therefore do not have guaranteed logarithmic worst-case access characteristics when mapped to secondary storage. But a quad-tree can always be mapped to a binary tree, and a 'grow and post' tree is a binary tree mapped to an n -ary tree. So it is natural to ask why it is not possible to map a quad-tree to a 'grow and post' tree?

In fact, a mapping between a quad-tree and a B-tree has been proposed [10] . But the spatial proximity relationships of the components of the quad-tree are lost in the mapping from two dimensions to one. To preserve these relationships, a mapping is needed from a quad-tree to a two-dimensional structure with the properties of a 'grow and post' tree. The problem lies not with the mapping, but in devising the two-dimensional 'grow and post' tree. Once this problem is solved, it becomes possible to support large-scale, persistent quad-trees, and the very large body of computational techniques now associated with them, within a generic, multi-dimensional indexing framework.

The adoption of regular binary partitioning of the data space also brings a particularly attractive feature of the quad-tree to the whole supporting framework: the precise registration (coincidence) of data space partition boundaries in overlay and spatial join operations - notably in GIS (Geographic Information System) applications. This is extremely important for the efficient and accurate execution of such operations. And since this property is a feature of the whole framework, it applies equally to raster and vector representations of spatial data, and conversions between the two.

It has sometimes been claimed that regular binary partitioning leads to the inclusion of too much empty space in the representation of skewed data distributions, particularly in the direct representation of extended spatial objects. We have found this not to be the case [11]

The basic reason for this is that, using a recursive partitioning scheme in a dynamic environment, the positions of the partitions at any level are moved very rarely compared to those at the level below. They are therefore effectively fixed, and in a dynamic environment their position becomes almost arbitrary. What is much more important is that, as already emphasized, it is possible to contract the representation to a set of occupied

subspaces. It is also important that regular binary partitioning does not necessarily imply a strict cyclic order in the choice of dimension/domain for the partitions.

A consequence of properties (d) and (e) is that, in contrast to conventional indexing methods, the binary key for an indexed object can be generated *dynamically* during the descent of the tree. Its value and length will no longer depend solely on the indexed value of the object - it will also depend on its *degree of similarity* to other objects. When a leaf node overflows and splits, the keys of the two resulting nodes will be extended just as far as is necessary to differentiate them.

This incremental approach to index key generation opens the way to the indexing of data entities with different structures in the same index. In conventional indexing, it is assumed that all the elements in an indexed data set have the same structure. In a relational database, this structure is stored in the schema, and is used to interpret the contents of the data elements in the corresponding relation.

New data models, however - specifically nested relational, object-oriented and logic - all allow structural variation within the elements of a relation, class or predicate respectively. In order to apply conventional indexing techniques to nested relations or objects, their structures must be resolved into simpler, fixed structures connected by physical or logical links. There is as yet no mechanism for a *single* index on the whole nested tuple or complex object instance.

However, it is always possible to generate a unique key from a (unique) instance of any complex structure, by including the structural information as well as the value information in the key. Then property (d) makes it possible to index very complex structures efficiently without in general using correspondingly long keys.

Further, a consequence of property (f) is that, if a set of points or objects in the data space generates a set of keys which are all different within the first few bits, then only these very short keys need be stored in the index. If, on the other hand, all the keys share a long, identical prefix, then this prefix will migrate to a single instance in the root of the index tree. Either way, the index representation becomes extremely compact.

All these properties, however, rest on the assumption that a data structure can be devised which complies with the ten principles listed above - which themselves require a solution of the n -dimensional B-tree problem. This is why a solution of this problem is such a key result.

BANG indexing

The original motivation for this research was to provide symmetric indexing support for rules and facts in a deductive database system based on logic [12] [13] [14] }. This proved to be a very difficult problem so, following the classical precept, we began by trying to solve a simpler problem: the multi-dimensional indexing of fixed-structure tuples. The result was the BANG file [15] [16] - a **B**alanced **A**nd **N**ested **G**rid file. With hindsight, this was a misnomer: a BANG index is in fact a balanced tree structure like a B-tree. What makes it different from a B-tree, and what it shares with the Grid file [17] [18] , is that it partitions the data space itself, rather than the values within the space i.e. BANG index entries are not data values, but encoded representations of subspaces of the data space.

Apart from the B-tree, the other main source of inspiration for the BANG index design was linear hashing [19] . This led to the incorporation of two essential elements: variable length and bit-wise extensible index keys, and the dynamic generation of search keys.

The BANG file represents a set of n -tuples as points in an n -dimensional space. The index key associated with each point is a Peano code, generated by interleaving the bits of its n attribute values, from most to least significant, cycling through the n dimensions. However, the full key is not stored with the tuple. In fact it is not stored at all: it is generated dynamically during a tuple search, and during data space partitioning. Further, it is rare that the full key is generated: usually only a short prefix is necessary.

The data pages containing the n -tuples correspond to subspaces of the data space. These subspaces are also uniquely identified by Peano codes, generated by a sequence of regular binary partitions of the domains of the data space, cycling through the dimensions in the same sequence as for the index keys. The Peano code of any subspace is a prefix of the index key of every data point which it encloses.

The index itself is a balanced, multiply-branched tree structure in which every level corresponds to a recursive partitioning of the space represented at the level below. Each branch node represents a subspace which is defined by a Peano code in the index entry which points to it from the index level above (with the exception of the root node, which by definition represents the whole data space). Conversely, each index entry within a branch node represents a subspace which is enclosed by the subspace represented by the branch node itself.

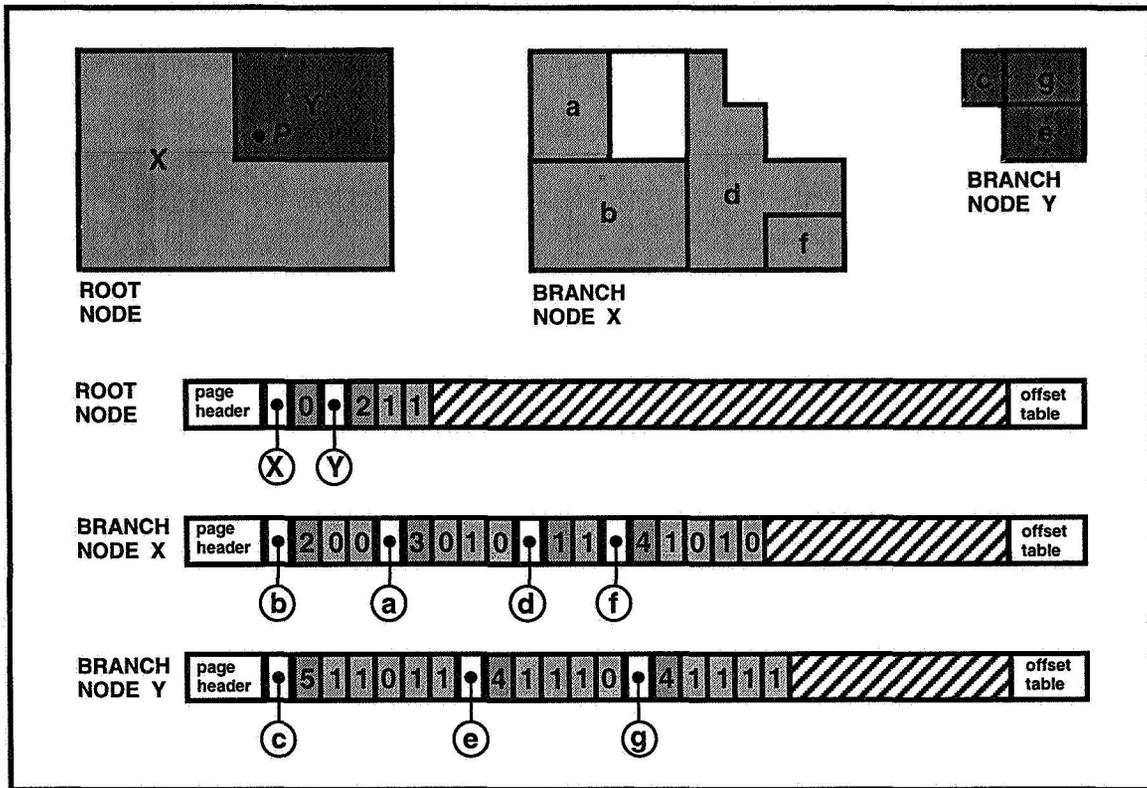


Figure 1: BANG indexing

Because of the regular binary partitioning sequence, the boundaries of partitions never intersect—either within the same index level, or between levels (although they may partially or fully coincide). However, there is nothing to prevent the creation of a subspace which fully encloses another at the same index level.

An example of a simple two-level partitioning of a data space, and the BANG index structure corresponding to it, is given in figure 1. Note that each index entry consists of three components: a pointer to the level below, an integer giving the length of the Peano code in bits, and the Peano code itself. Note also that the Peano codes defining subspaces at the lower index level are *relative* to the codes at the level above. In general, the full Peano code defining a subspace at index level l is the concatenation of all the Peano codes encountered in the direct traversal from the root to level l . This leads to an extremely compact index. (In practice, trailing code bits beyond the penultimate byte boundary are repeated at the next lower level, since it is much more efficient to perform code comparison and update operations if the alignment of bits in the byte is maintained).

Exact-match tuple search proceeds downwards from the root in the familiar B-tree manner. But within an index node, a bit-wise binary search is made for a Peano code which matches the corresponding bits of the search key. The key is dynamically extended during

this search: only as many key bits are generated as the length of the longest Peano code encountered. Note in the example in figure 1 how only very short keys need to be generated to locate the target data node, however long the actual tuple attributes may be.

Within a data node, tuples are stored in Peano code order. The target tuple can then be located by bit-wise binary search, dynamically generating both search and target keys. (Alternatively, each tuple can be stored with its full Peano code as an additional attribute).

It will be seen from figure 1 that there is a potential location ambiguity which arises when one subspace encloses another. This ambiguity is, however, simply resolved by always choosing the index entry with the longest Peano code which matches the search key: this must represent the innermost of any nested subspaces. Figure 1 also demonstrates a critically important feature of domain indexing: it is not in general necessary to represent the whole data space in the index, if large tracts of the space are empty. In the example of figure 1, some exact-match queries will fail at the root index level. In general, therefore, the *average path length* traversed to answer an exact-match query *will be less than the height of the index tree*. The more highly correlated the data, the greater the advantage of this feature compared to the conventional range-based indexing of the B-tree.

This advantage is reinforced by the extremely compact index - typically 1%-2% of the size of the data set for 1K index and data pages - due to the (on average) very short index entries. This leads to high fan-out ratios and hence a shorter index tree height and consequent faster access for a given data set.

But the greatest advantage of all is the multi-dimensional flexibility of this design. It can be used in conventional ways as a one-dimensional primary or secondary index, providing a smooth upgrade path from B-tree indexing, or it can be used multi-dimensionally when the query pattern is unpredictable. Figure 2 shows the results of two sets of two small experiments with A BANG index. The middle table shows the number of data pages accessed in response to all possible combinations of an exact-match/partial-match query on an employee relation of three attributes. Identity no distinguishes a tuple uniquely, the other two attributes do not.

Employee relation:		<i>10,000 tuples</i>		<i>20,000 tuples</i>	
1-d: <u>identity no</u> , surname, forename		<i>615 data pages</i>		<i>1246 data pages</i>	
3-d: <u>identity no</u> , <u>surname</u> , <u>forename</u>					
Form of query	1-d	3-d	1-d	3-d	
(100, smith, john)	1	1	1	1	
(100, ? , ?)	1	42	1	60	
(? , smith, ?)	615	84	1246	164	
(? , ? , john)	615	145	1246	230	
(100, smith, ?)	1	7	1	9	
(? , smith, john)	615	21	1246	28	
(100 , ? , john)	1	15	1	22	
Total:	1849	315	3742	514	
Av. per query:	264	45	535	73	
	Ratio 5.9 : 1		Ratio 7.3 : 1		

Figure 2: 1-d versus 3-d indexing

The 1-d column shows the number of accesses when the relation is indexed one-dimensionally on the *identity no.* attribute (which is a unique key). The 3-d column show the number of accesses when the relation is indexed multi-dimensionally on all three attributes. (the *surname* and *forename* attribute values are not unique). Assuming that each query is equally probable, the average number of data page accesses per query is shown at the bottom of the table. It will be seen that, *on average*, the single 3-d index is almost six times faster than a single 1-d index. The right-hand table shows the experiment repeated with a relation of twice the size. The 3-d index is now over seven times faster than the 1-d index. This is very good news: the larger the relation, the greater the relative performance improvement. In this example, the improvement is approaching an order of magnitude.

It must be said at once that this is a very crude test, and a number of factors - notably in the data distribution - could substantially change the outcome. Nevertheless, the performance advantage of the multi-dimensional approach in this example is so great that it is surely hard to dismiss. A better testimony is that BANG indexing has been used for several years now as the only index method supporting persistent data in the ECLIPSe Common Logic Programming System [20], which is now in use by over 300 sites around the world. So far, no request has ever been received for an alternative indexing method.

At this point the reader may justifiably ask: if multi-dimensional indexing is really so much better, why does the B-tree still exist? One answer is that, in order to take full advantage of multi-dimensional indexing, practically everything else in the database system must also be changed, from the setting up of indexes in the schema definition to the query optimizer and the fundamental relational operators - notably joins. There is therefore an enormous legacy problem. But the failure of all multi-dimensional indexing methods to date (including BANG indexing) to guarantee acceptable worst-case performance characteristics must also have been a major contributing factor. This weakness manifests itself in every multi-dimensional design in one of two ways: either there is no guaranteed minimum occupancy of data and index pages, or there is no guaranteed maximum path length for (single) search and update operations. Although this risk may be acceptable in many conventional business applications, it is certainly not so in the increasing number of real-time, mission-critical applications.

The BV-tree

In [6] we showed that this universal problem is a consequence of the basic topological properties of a data space, or rather of previous erroneous assumptions about the mapping of a recursively partitioned data space to a tree structure. The solution to the problem, surprisingly, is an *unbalanced* rather than a balanced tree.

Briefly, the problem arises when an index node overflows and splits along a boundary which does not coincide with any node boundary at the next lower index (or data) level. The choice is then either to choose a different splitting boundary - thereby losing any minimum occupancy guarantee for the two resulting nodes; or to force a split along the upper level boundary in one or more lower level nodes - which may propagate recursively to the bottom of the tree, so that no certain upper limit can be placed on the number of node splits resulting from a single update.

The nature of the problem is illustrated in figure 1. The root node of the two level index in figure 1 is created when the original single index node overflows into two nodes X and Y . The boundary between the subspaces represented by X and Y is chosen so that the occupancy of both nodes is as equal as possible. But it will be seen that this boundary cuts through region d at the index level below. In the figure, region d has been assigned to node X , with the result that a key search for a point P will fail, since there is no entry corresponding to this point in node Y .

In the original BANG index design [15], the problem was side-stepped by backtracking up the tree to the next longest entry key (i.e. the next smallest subspace - if such exists) matching the search key. However, in the worst case this could involve searching the entire index tree. In [16] forced recursive splitting was suggested as a better performance compromise, together with algorithms designed to minimize its worst effects. The maximum number of resulting split nodes in the worst case could then be exactly predicted, but the price for this was the loss of the minimum node occupancy guarantee. A

Careful examination will reveal the same trade-off in every other multi-dimensional index design.

The solution offered in [6] to this impasse is not in itself an index method, but a general result concerning the representation of a recursively partitioned data space by a tree structure. The basic idea is very simple: instead of forcing a node to split, promote it to the index level above. At first sight this may seem illogical, since it destroys the one-to-one correspondence between the levels of recursive data space partitioning and the levels of the index tree which represents this partitioning. However, all that it really destroys is the prejudice that such a static correspondence is necessary.

Instead, the correspondence is reconstructed *dynamically*, while traversing the tree structure. By promoting a node instead of splitting it, its index entry is moved to the root of the subtree which contains the pointers to the two nodes which would otherwise contain the split node entries. Thus any index tree search which visits this subtree root can then pick up the promoted entry - if the search key matches - and follow one of these pointers back down to the next lower index level. The entry is then included in the set of index entries to be searched at this level, exactly as if it had been split rather than promoted.

The result, as illustrated in figure 3, may seem paradoxical: an unbalanced tree (a BV-tree) with the operational properties of a balanced tree. But this is simply because all search and update operations are effectively carried out on a balanced tree reconstituted from the unbalanced tree. In figure 3 the subspace boundaries are deliberately chosen to be randomly shaped, to emphasize that this is a general technique, not a specific index method. A more detailed account is given in [6], but the main point is that the structure makes it possible to combine a minimum node occupancy guarantee with guaranteed maximum search and update path lengths.

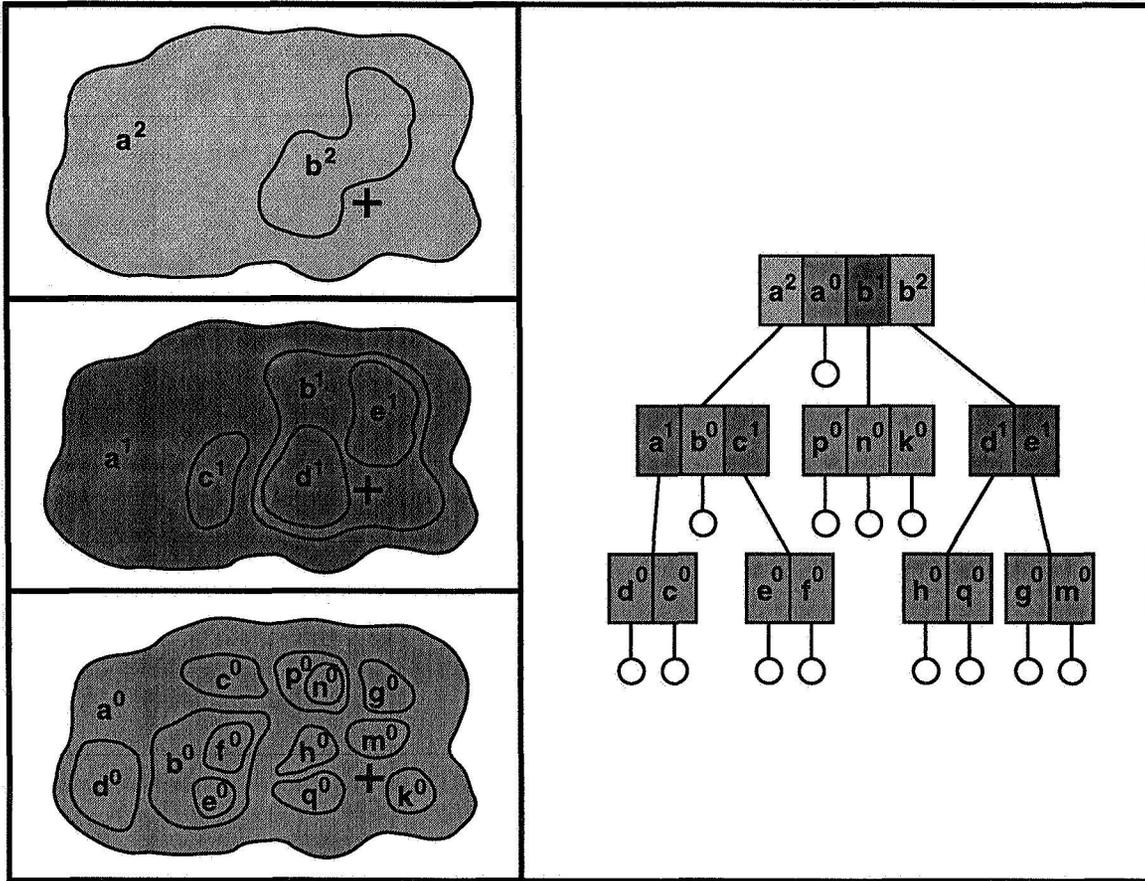


Figure 3: The BV-tree

The worst-case node occupancy is 33% for the BV-tree, compared to 50% for the B-tree, but this is the maximum which is topologically possible in more than one dimension. In practice, as with the B-tree, the worst case can always be improved upon by redistributing the contents of nodes whose occupancy levels fall below some arbitrary value. The average occupancy is the same as that of the B-tree i.e. around 69%.

The number of nodes visited in an exact-match search, and the maximum number of nodes *split* as the result of a single update, is logarithmic in the size of the data set, as in a B-tree. The number of nodes *visited* during a single update can however be greater than logarithmic: in the worst case it is $h(h+1)/2$, where h is the maximum height of the tree.

[N.B. This case was overlooked in [6], where it was claimed that updates can always be performed in logarithmic time. This is indeed true except in one case, which arises when a promoted node overflows, and when one of the resulting split nodes can be demoted. The demotion must be carried out immediately, in order to maintain the dynamic behavior of the tree. We believe - although have not proved - that this, like the 33% minimum page occupancy, is a topological inevitability. In practice, this will be such a rare occurrence that

the average update time will hardly deviate from logarithmic. More important, the worst-case update time remains entirely predictable].

It is also possible that, if a fixed page size is used, the height of the index tree may be greater than that of the equivalent balanced tree (see [6] for an analysis). This tendency may however be more than compensated by the fact that the average path length for an exact-match search in domain-based indexing is less than the height of the tree.

So if we are to judge the acceptability of multi-dimensional indexing by comparison with the worst-case characteristics of one-dimensional B-tree indexing, there is clearly now very little difference between them. The important point is that multi-dimensional indexing can now safely support mission-critical applications with absolutely predictable worst-case performance. Best of all, there is *no* price to pay for the extra flexibility and average performance which multi-dimensional indexing gives.

A BANG representation of a BV-tree

We have so far made the implicit assumption that the general properties of the BV-tree can be implemented in a specific index representation. And if we are to achieve our ultimate goal of an indexing technology which satisfies all the principles and properties listed above, we also have to combine this implementation with all the principles applied in BANG indexing. So the question remains: can such a design be implemented efficiently?

There is not space here to set out the full details, but we hope to show enough to demonstrate that it is possible to build an implementation which is both efficient and elegant, with relatively low software complexity. The appendix contains the pseudo-pascal code for a procedure to perform a key-search within an index node of a BANG index representation of a BV-tree. Figure 4 illustrates both the modified form of a BANG index node, and the key-search operation on it. The contents of the ten index entries are set out vertically rather than horizontally as in figure 1, but otherwise the only difference in their structure is the addition of the *partition level* of the entry. This is the index level at which the entry was originally created, counting the index leaf node level as level 0. The index node shown is located at level 2, since the highest partition level shown is 2, but the node also contains two entries promoted from level 1, and one entry promoted from level 0. (In figure 2, the equivalent would be the promotion of the entry for subspace *d* from node *X* at level 0 to the root node at level 1). We frequently refer to promoted entries as guards.

Before the tree search begins, an array is initialized, which we call the *guard set*, having one element for each partition level of the tree, such that array element *l* corresponds to partition level *l*. Each element consists of a record of two fields: the relative length of an index entry key (i.e. the length relative to the page in which it lies); and the pointer associated with that index entry. The array is used to hold this information on the longest entry key of each partition level so far encountered in the traversal of the tree.

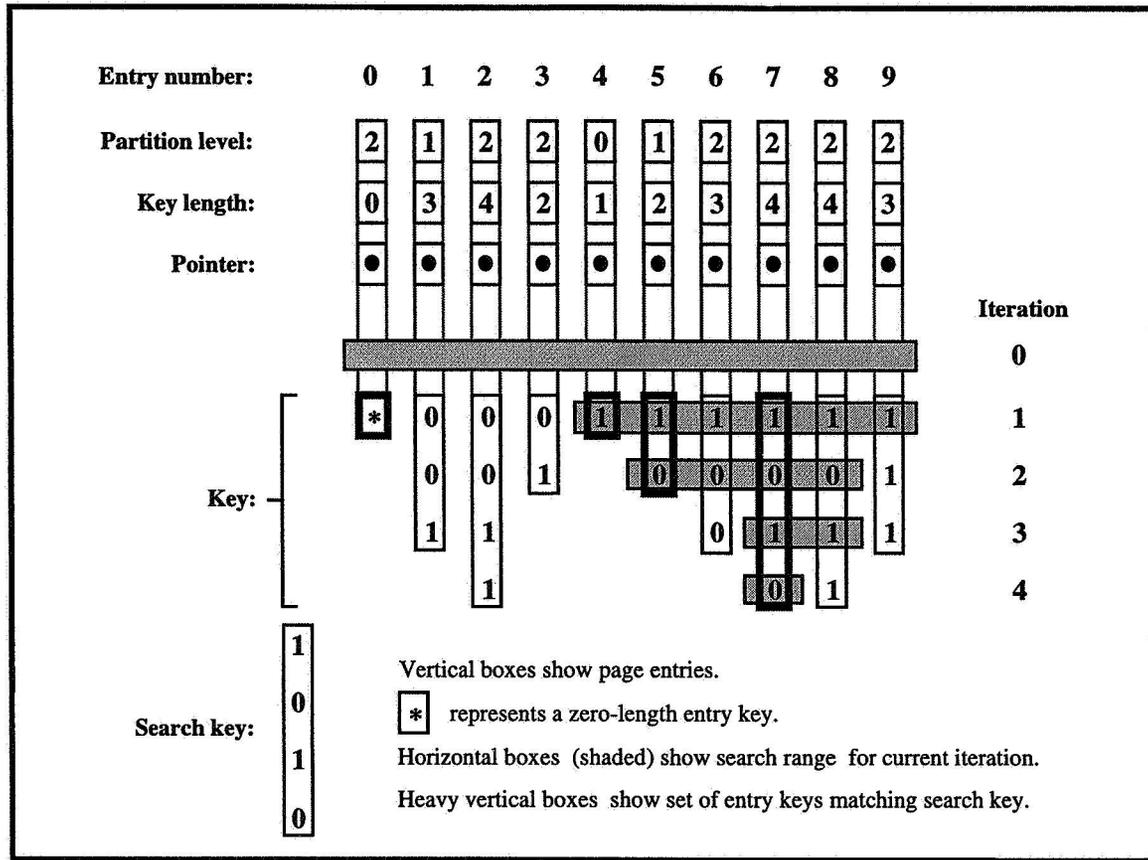


Figure 4: Key search in a BANG implementation of a BV-tree

Regardless of their promotion level, index entries are stored in lexical order of their entry keys, so that the longest key matching the search key can be found by bit-wise binary search of the index node (entry 7 in figure 4). The range of each iteration of this binary search is shown by the shaded horizontal bars in figure 4, and it will be seen that the next matching prefix of the search key is always the first entry in the range. At each iteration, the relative key length of this entry is compared with the length stored in the array element corresponding to the same partition level. If the new entry is the longer, its length and associated pointer are stored in this array element, overwriting its previous contents.

When the binary search within a node at index level l is complete, the next node to be searched is that pointed to by array element l . Note that array element l may at this stage simply hold the key length and pointer of the longest matching entry in the node just searched at level l , or it may hold values carried down from a promoted entry at level $(l+1)$ - if this entry has a longer key. And such an entry may also have been carried down from higher levels in the course of the tree search.

An example

The effect of this simple algorithm is to reconstitute a balanced tree by moving the relevant promoted entries back to the index level at which they were created. As an example, consider a search for the point marked + in figure 3. At the root index level (level 2) entries \mathbf{a}^2 , \mathbf{b}^1 and \mathbf{a}^0 represent the smallest subspaces enclosing the target point at each partition level, so the lengths of these entries and their associated pointers are stored in the guard set. We then follow the pointer in element 2 of the guard set down to the next index level. Here the only matching entry is \mathbf{a}^1 . But there already exists in array element 1 of the guard set another entry - \mathbf{b}^1 - carried down from the level above, and which plainly encloses the target point more closely than \mathbf{a}^1 i.e. in a BANG representation, it has a longer key.

So we now follow the pointer associated with \mathbf{b}^1 . Note that this causes the tree traversal to effectively backtrack up the tree and down another branch, although no previously visited nodes are revisited. Finally, we see that there are no matching entries at level 0 in the node pointed to by \mathbf{b}^1 , so we pick up the existing pointer in array element 0, which is the pointer from entry \mathbf{a}^0 carried down from the root. Thus a second 'virtual backtrack' occurs. Note however that the total number of nodes actually visited is equal to the maximum direct path length from root to leaf of the tree. This is true for all exact-match search paths in a BV-tree, thus guaranteeing its logarithmic search behavior.

Of course, the complete search and update code is considerably longer and more complex than that shown in the appendix, but the tree traversal and dynamic tree reconstruction techniques shown recur throughout.

Conclusion

We can now reasonably claim to have fulfilled all the principles and properties laid down in our initial list of design requirements. Multi-dimensional indexing really is now a more flexible alternative to B-tree indexing. But this is not the end of the matter - it is only the beginning. As we began by pointing out, our true objective has not been to find a replacement for the B-tree: that is just a spin-off. The main aim has been to develop a basis for a new *generic* indexing technology which is capable of supporting much more generalized data types than those of conventional business applications, and in a scalable way for large applications. We are confident that, with the technology which we have assembled above, we can now develop this generalized support. But it is clear that there is a lot of research still to be done, and whole new areas to be investigated.

References

1. Bentley, J. *Multidimensional Binary Search Trees in Database Applications*. IEEE Trans. on Soft. Eng., Vol.SE-5, No.4, July 1979.

2. Robinson, J.T. *The K-D-B-Tree: A Search Structure for Large Multidimensional Dynamic Indexes*. ACM SIGMOD Conf. 1981.
3. Bayer, R. and McCreight, E. *Organisation and maintenance of large ordered indexes*. Acta Informatica, Vol.1, No.3, 1972.
4. Samet, H. *The Design and Analysis of Spatial Data Structures*, Pub. Addison Wesley, 1989.
5. Kriegel, H.-P., Schiwietz, M., Schneider, R. and Seeger, B. *A Performance Comparison of Multidimensional Point and Spatial Access Methods*. 1st Symposium on the Design of Large Spatial Databases, Santa Barbara CA, 1989. [Lecture Notes in Computer Science No. 409, Springer-Verlag, 1990].
6. Freeston, M. *A General Solution of the n-dimensional B-tree Problem*. ACM SIGMOD Conf., San Jose, CA, May 1995.
7. Lomet, D. and Salzberg, B. *The hB-tree: a Robust Multi-Attribute Indexing Method*. ACM Trans. on Database Systems, Vol.15, No.4, 1989.
8. Lomet, D. Grow and Post trees.
9. Orenstein, J. *Spatial Query Processing in an Object-Oriented Database System*. ACM SIGMOD Conf. 1986.
10. de Jonge, W., Scheuermann, P. and Schijf, A. *Encoding and Manipulating Pictorial Data with S^+ -trees*. 2nd Symposium on Large Spatial Databases, Zurich, August 1991.
11. Freeston, M. *The Comparative Performance of BANG Indexing for Spatial Objects*. 5th International Symposium on Spatial Data Handling, Charleston SC, August 1992.
12. Bocca, J., Dahmen, M. and Freeston, M. *MegaLog—A Platform for Developing KnowledgeBase Management Systems*. 3rd Russian Conference on Logic Programming and Automated Reasoning, St. Petersburg, Russia, July 1992.
13. Bocca, J. *MegaLog—A Platform for Developing Knowledge Base Management Systems*. Int. Symposium on Database Systems for Advanced Applications, Tokyo, Japan, April 1991.

14. Vieille, L., Bayer, P., Kuechenhoff, V. and Lefebvre, A. *EKS-VI, A Short Overview*. Proc. AAAI-90 Workshop on Knowledge Base Management Systems, Boston, July 1991.
15. Freeston, M. *The BANG file: a new kind of grid file*. ACM SIGMOD Conf., San Francisco, May 1987.
16. Freeston, M. *Advances in the Design of the BANG File*. 3rd International Conference on Foundations of Data Organisation and Algorithms (FODO), Paris, June 1989.
17. Nievergelt, J., Hintenberger, H. and Sevcik, K. *The Grid File: An Adaptable, Symmetric Multikey File Structure*. ACM Trans. on Database Systems, Vol.9, No.1, 1984.
18. Hinrichs, K. *The Grid File System: Implementation and Case Studies of Applications*. Doctoral Thesis Nr. 7734, ETH Zurich, 1985.
19. Litwin, W. *Linear Hashing: a New Tool for File and Table Addressing*. Proc. 6th Int. Conf. on Very Large Databases, Montreal, Canada, 1980.
20. Eclipse 3.5 *User Manual*. European Computer-Industry Research Centre (ECRC), 1996. URL <http://www.ecrc.de/eclipse/html/umsroot/umsroot.html>.

Appendix

procedure search_index_page(page, bottom, top, search_key, guard_set, insertion_point)

{ Initialize the byte offset from the start of the (absolute) search key to the start byte of the relative search key i.e. to the byte corresponding to the first key byte of the page entries }

relative_search_key := search_key + page_level div BYTESIZE;

{ Initialize the bit offset of the start of each entry key in the page, relative to the start of the first key byte }

start_bit_offset := page_level mod BYTESIZE;

{ Initialize comparison bit number relative to the start of the first byte of each entry key. The most significant bit in a byte is numbered as bit 0.

Initially, the comparison_bit_number is set to one less than that of the actual start position, so that any initial entries of zero length will be correctly treated }

comparison_bit_number := start_bit_offset - 1;

while bottom <= top **do begin**

if length(key(page_entry(page, bottom)))=comparison_bit_number-start_bit_offset + 1

then begin

{ This is a matching entry i.e. the complete key of the entry matches a prefix of the insertion key. Note that there may be several matching entries with identical keys, provided that they differ in their index node level. }

Note also that the presence of a guard of node level l-x in a page of node level l does not imply that the page must also contain guards of all higher levels l-x+n, where 0 < n < x }

{ Record the matching entry in the guard set }

level := index_node_level(page_entry(page, bottom));

guard_set.entry[level] := bottom;

bottom := bottom + 1

end

else

if length(relative_search_key) = comparison_bit_number - start_bit_offset + 1

then *{ No further matching entries possible: the search key (region) encloses all remaining entries in the current search set. }*

Note that there must be at least one entry in this set, since this point in the code is only reached if there exists at least one entry in the current search set whose key is at least one bit longer than the current position of the comparison bit }

begin

insertion_point := bottom;

{ The insertion point lies immediately before the entry numbered bottom in the page entry table }

return

end

else begin

{ Establish range of entries matching search key as far as the next comparison bit }

{ Advance comparison bit }

comparison_bit_number := comparison_bit_number + 1;

if value(comparison_bit(key(page_entry(page, top)))) = 0

then begin

if value(comparison_bit(relative_search_key)) = 1

then bottom := top + 1 *{ No further matching entries possible }*

{ else all entries in the current search range must match as far as the current comparison bit: the range thus remains unchanged }

end

else if value(comparison_bit(key(page_entry(page, bottom)))) = 1

```

then begin
  if value(comparison_bit(relative_search_key)) = 0
  then top := bottom - 1 { No further matching entries possible }
  { else all entries in the current search range must match as far as the current
    comparison bit: the range thus remains unchanged }
end
else begin

  { Binary search for first entry with current bit set to 1. If this section of code is
    entered, there must be at least two entries in the current search range. This
    follows from the fact that the program execution must drop through the
    preceding code to arrive at this point. This can only occur if the value of the
    comparison bit for the top entry is 1, and that for the bottom entry is 0. Hence
    these two entries cannot be the same. It also follows that there must be at
    least one entry in the new search range }
  top_limit := top;
  bottom_limit := bottom;
  repeat
    middle := (top_limit + bottom_limit) div 2;
    if comparison_bit(key(page_entry(page, middle))) = 1
    then top_limit := middle - 1
    else bottom_limit := middle + 1
  until top_limit < bottom_limit; { end of binary search }

  { Reset lower or upper limit of search range: bottom_limit is first entry with
    comparison bit value 1; top_limit is last entry with comparison bit value 0 }

  if value(comparison_bit(relative_search_key)) = 1
  then bottom := bottom_limit
  else top := top_limit
  end { Binary search ... }
  end { Establish range of entries matching search key... }
end { while bottom <= top }

  insertion_point := bottom
  { The insertion point lies immediately before the entry numbered bottom in the page
    entry table }

end {procedure search_index_page}

```


59-82
83191

Advanced Optical Disk Storage Technology

Fred N. Haritatos

Rome Laboratory
32 Hangar Road
Rome, NY 13441-4114
haritatosf@rl.af.mil
Tel: 315-330-4581
Fax: 315-330-2728

Abstract

There is a growing need within the Air Force for more and better data storage solutions. Rome Laboratory, the Air Force's Center of Excellence for C3I technology, has sponsored the development of a number of operational prototypes to deal with this growing problem. This paper will briefly summarize the various prototype developments with examples of full mil-spec and best commercial practice. These prototypes have successfully operated under severe space, airborne and tactical field environments. From a technical perspective these prototypes have included rewritable optical media ranging from a 5.25-inch diameter format up to the 14-inch diameter disk format. Implementations include an airborne sensor recorder, a deployable optical jukebox and a parallel array of optical disk drives. They include stand-alone peripheral devices to centralized, hierarchical storage management systems for distributed data processing applications.

Introduction

Command, Control, Communications, Computing, and Intelligence (C4I) is essential to the Air Force and, for that matter, to industry as well. C4I systems must effectively store, retrieve, and manage massive amounts of digital data. 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. Over the years, Rome Laboratory has nurtured a comprehensive program, developing new storage techniques that meet the various demands for data storage and retrieval. This article traces the history of and presents an overview of Rome Laboratory's research in optical disk storage technology.

The Evolution of Optical Disk Technology

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 1 mm 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. 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.

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. Rome Lab's current development activities are concentrating in using this technology in various disk format sizes.

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.

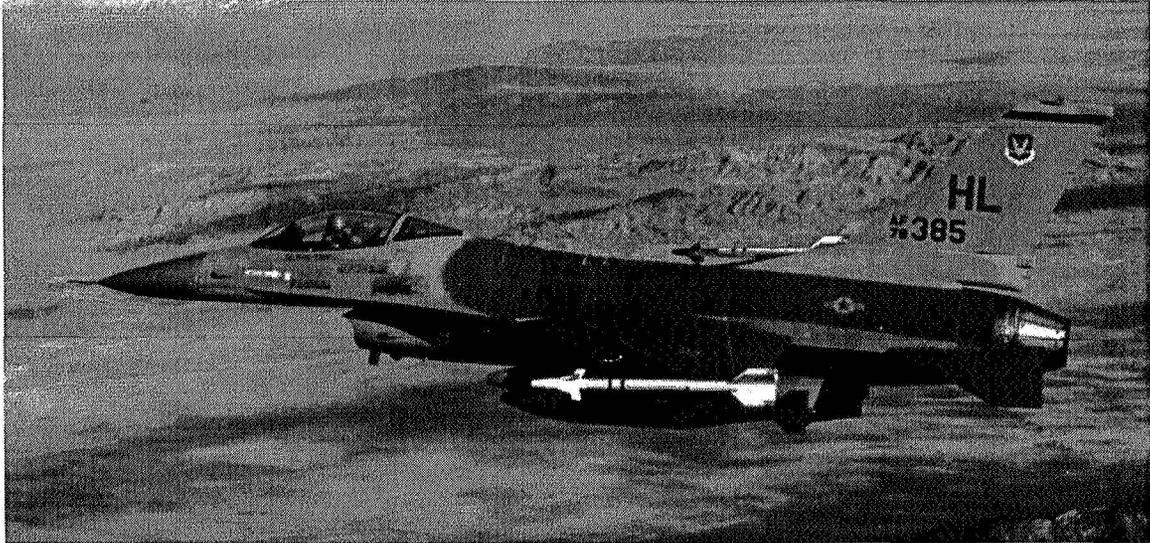
Rome Laboratory's Technology Development Program

Rome Laboratory has sponsored work since the early 1970s to exploit the benefits of optical disk technology. An early prototype used an argon laser to record and playback digital data from a 12.5-inch plastic-based optical disk. Further investment led to the delivery in 1982 of a large-capacity optical jukebox for satellite imagery storage and retrieval applications. The jukebox held 100 write-once, read-many (WORM) disks that provided a [1] one Terabyte storage capacity.

In the late 1980s, we transitioned our rewritable optical storage techniques from our laboratory environment to the "real world." This involved building and testing a family of high-performance prototypes for operation in working Air Force environments. Three advanced development models (ADMs) were built: 1) a 5.25-inch diameter, rewritable optical disk system; 2) a 14-inch rewritable optical disk system and 3) deployable optical jukebox.

5.25-inch Optical Disk System Advanced Development Model

The first Advanced Development Model [2] was designed to operate on board tactical fighter aircraft.



Some of the key environmental performance parameters included: high and low temperature, vibration, mechanical shock, acceleration, altitude, and humidity. The recording head, which moves across the disk surface to read and write, is susceptible to shock and vibration. It usually consists of a laser diode and various optics. By moving the laser diode and the majority of the optical components off the recording head assembly, we were able to reduce this mass. By reducing the recording head mass, we enabled faster access times and exceptional vibration and shock performance, ensuring reliable in-flight operation.

5.25-inch Optical Disk System Advanced Development Model:

Storage capacity:	300 megabytes, single-side
Data transfer rates:	5 megabits per second sustained 10 megabits per second burst
Max access time:	100 milliseconds (inner to outer radius)
Size:	5.0 inches x 6.5 inches x 10.5 inches
Weight:	16 pounds with disk cartridge

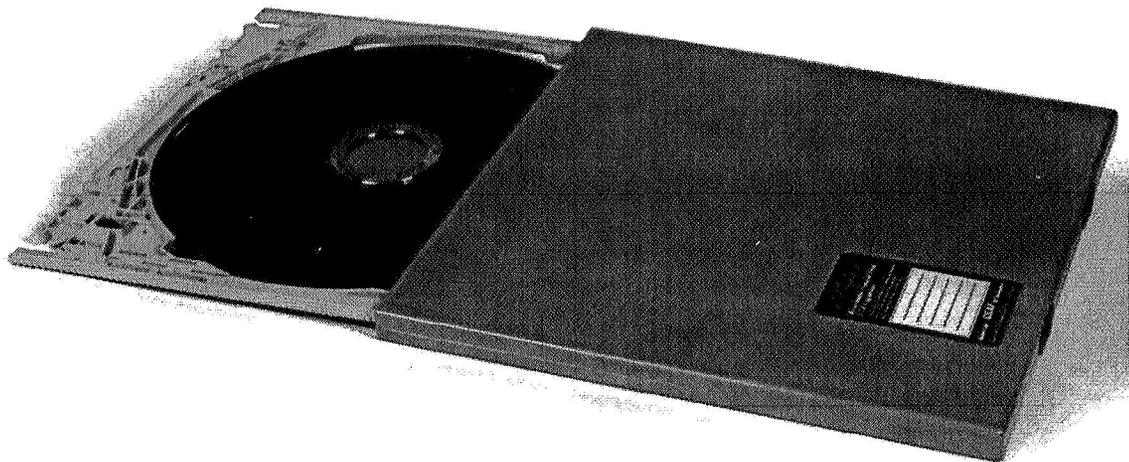
To fully evaluate the design robustness under realistic environmental conditions, we conducted an operational flight test on board an F-16 tactical fighter aircraft at Eglin AFB, Florida, from June - July 1989. An important Air Force milestone, it represents the first rewritable optical disk system to successfully fly on a high-performance aircraft using unconstrained flight maneuvers. A second device, provided to NASA-Goddard Space Flight Center and launched on the Space Shuttle Discovery, demonstrated advanced robotic concepts. Information describing the robot's operational environment was stored on the optical disk. The rewritable optical disk contributed greatly to the success of the NASA experiment.

14-inch Optical Disk System Advanced Development Model

The second ADM was [3] designed to provide larger data storage capacity and faster data transfer rates. The equipment operated in tactical environments found in larger aircraft and deployable, data processing facilities.



Digital data was recorded on a 14-inch, rewritable, optical disk. The larger media is a double-sided disk that uses a preformatted spiral pilot track to allow continuous tracking during write, read, and erase operations. As a result of evaluating several candidate approaches, this is the preferred disk construction for operation under severe environmental conditions.



A dual-channel laser diode is used to provide faster data recording and erasure. Each data channel is simultaneously recorded by using individually addressable elements on the laser diode array. The array is imaged through a single optical system and aligned to a common focal plane.

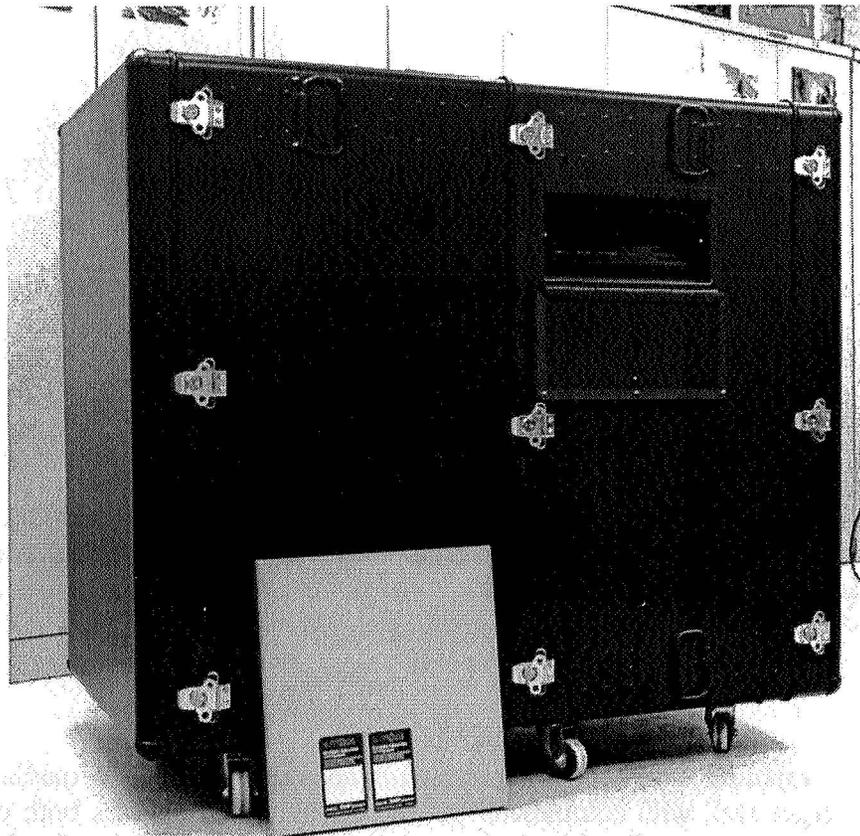
In September 1993, the equipment successfully completed a series of airborne tests on an RC-135 reconnaissance aircraft. This demonstrated its reliability under a wide range of airborne conditions. The equipment operated through tactical descents, mid-air refuelings, 60-degree bank turns and touch-and-go landings. No data were corrupted or lost during the test, and there were no hardware failures.

The 14-inch ADM possessed the following features:

- Storage capacity: 6 gigabytes per side (GB)
- Data transfer rates: 25 megabits/second sustained
50 megabits/second burst
- Max access time: 400 milliseconds
- Size: 17.5 inches x 23 inches x 24 inches
- Weight: 150 pounds

Deployable Optical Jukebox

Based on past successes with optical disk systems for space and airborne applications, we have developed a follow-on Advanced Development Model [4].



The Advanced Development Model optical jukebox provides mass data storage and retrieval capabilities for ground-based, large data base requirements. This electro-mechanical jukebox stores 10 double-sided rewritable optical disks of 12 gigabytes each. Under computer control, individual optical disks can be located, transported, and inserted into the optical disk drive within 10 seconds. A dual picker robotics mechanism enables fast disk access by reducing the time required to extract and insert a new disk within the drive.

The optical jukebox is designed to be modular for quick setup and tear-down times. Within 30 minutes, the equipment can be disassembled and packed into five deployment cases. Each case is two-man portable. Equipment assembly is completed without special tools, personnel, or training. The equipment was shipped on Jan 1996 to Air Force Special Operations Command for operational evaluation.

With the end of the Cold War and the shrinking DoD budget, there is a growing trend to rely more heavily on commercial-off-the-shelf (COTS) products to satisfy many of the Air Force's mass storage requirements.

Dual-Use Optical Disk Technology

The reliance on dual-use technology has motivated Rome Laboratory's initiative called "High Capacity Optical Jukebox." Many of the media and drive technologies demonstrated under the earlier militarized optical disk program will be transferred to commercial implementation. Under this effort, 50 rewritable optical disks and a rewritable optical drive will be developed and delivered to Rome Laboratory. Each optical disk can be rewritten almost endlessly, thus saving life-cycle media costs. The 14-inch diameter, double-sided optical disk can store 10 GB.

Because the optical disk drive is a modification of a commercial WORM product, we're able to leverage current WORM production lines. The design effort will concentrate on developing a multifunction optical drive that can access both WORM and rewritable media. The disk access times and data transfer rates will be comparable to the current commercial product. One of our objectives is to deliver an optical disk library system that is highly leveraged from an existing write-once product. This offers a path toward commercialization and lower acquisition and maintenance costs.

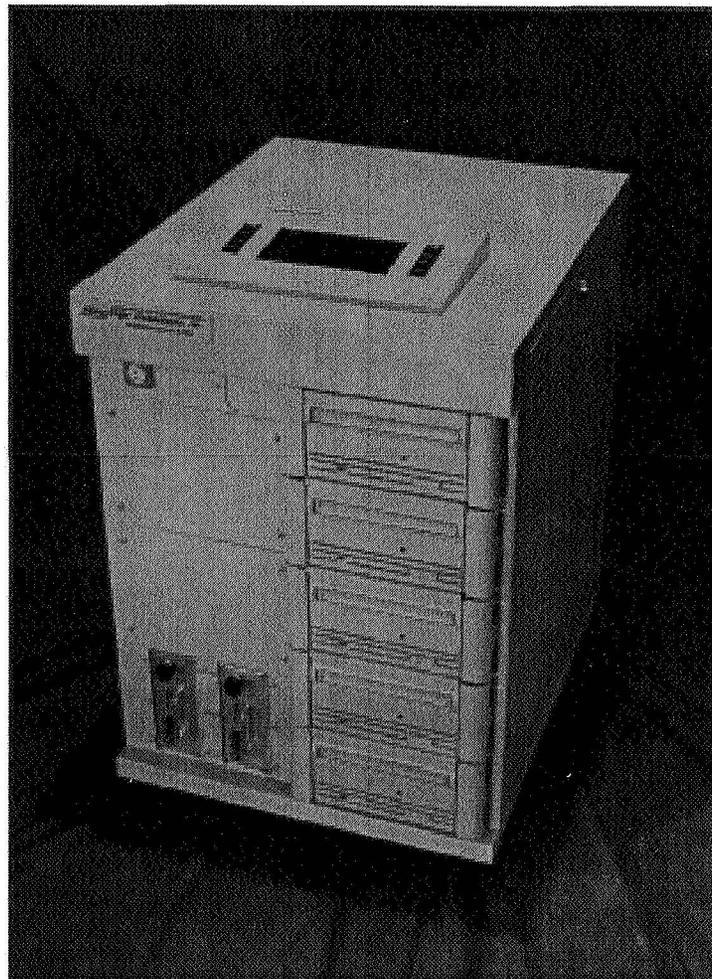
With today's emphasis on COTS solutions, this approach develops a media and drive design that merges well with commercial product plans. This enables both commercial and military interests to benefit while sharing costs and risks, shortening the development

cycle and saving research and development funds by leveraging an existing commercial product. An important goal is to enable the developer to incorporate the technology into a next-generation commercial product, thus enabling Air Force access to improved mass storage technology.

When the equipment is delivered in 1997, the rewritable library will be mated with a commercial WORM library to provide a 1-Terabyte storage capability. The hybrid (0.5-TB WORM and 0.5-TB erasable) optical disk library will become an integral part of the Hierarchical Storage Management (HSM) environment aimed at satisfying high-performance automated-intelligence data handling and image-exploitation requirements.

Parallel Optical Disk Operations

Another Advanced Development Model is our Optical disk-based Redundant Array of Inexpensive Disks (O-RAID).



The purpose is to determine the feasibility of developing a single, integrated, high-performance device capable of combining the benefits of optical disk technology within a RAID system. The design is partitioned into three basic elements: the optical disk drives, the rewritable media and the electronics controller. The basic O-RAID design consists of multiple optical disk drives operating as a single large optical disk to a host computer requesting the data. Optical disk drives are used because of their high reliability, data retention capabilities and the ability to remove media for storage. The RAID architecture supports increased reliability and accuracy of the data stored. Our initial prototype demonstrated a data transfer speed of 8.8 megabytes per second combined with a base storage capacity of 6.5 gigabytes provided by 5 parallel optical disk drives. The system design is modular and scalable to take advantage of improved optical disk technologies that may be available in the future.

Next-Generation Optical Disk Technology

While today's optical disk systems rely on laser diodes operating at 780 nm to record 1 μ m spots, using shorter wavelength lasers may increase disk storage. Throughout the world, researchers are investigating blue-green laser diodes operating at 460 nm. However, to fully exploit the new laser device advantages, new media and recording techniques must be developed. ARPA selected Rome Laboratory as Executive Agent for the "Short-Wavelength Optical Storage" Technology Reinvestment Project (TRP). The project's primary goal is to develop a 20 GB 5.25-inch optical disk by the year 2000, with access times and transfer rates that are at least equivalent to magnetic disk technology. To do this, we must resolve several key issues relating to substrate materials, servo, format, mastering and substrate processes, recording layers and processes, and read-channel electronics. As the Air Force makes greater use of digital images, video, and multimedia products for their military applications, we must develop cost-effective, high-density data storage. This effort's goal is to develop and demonstrate prototype high-density, rewritable optical disk technology (including media, heads, and drive) and viable manufacturing technology. The project has the potential to radically alter the way information is stored and retrieved in future military data storage systems.

Summary

Optical disk storage technology is playing an increasingly more important role in the Air Force's data storage and management requirements. Rome Laboratory's role has been to advance the state-of-the-art to satisfy the Air Force's unique operational needs through prototype development and operational testing. Our needs cover environments from space to airborne to tactical field conditions. Additionally, optical disk technology provides inherent advantages in distributed computing environments that require Terabyte and Petabyte storage capacities, medium access times and archivability.

However, optical disk, along with most secondary storage devices, suffers from storage capacity limits imposed by its 2-D planar format. Rome Laboratory is addressing these deficiencies by investigating alternative 3-D and 4-D optical memory technologies. These new approaches are being presented in a complementary paper.

References

- [1] Haritatos, Fred N., "Optical Recording," *Defense Electronics*, vol. 25, no. 10, Sept. 1993.
- [2] Haritatos, Fred N., "High Performance Optical Disk Systems for Tactical Applications," *IEEE*, May 1991.
- [3] Haritatos, Fred N., "Air Force Applications for Military Optical Disk Systems," *SPIE/IS&T Image Storage and Retrieval Systems*, Feb. 1992.
- [4] Haritatos, Fred N., "High Performance Optical Disk Systems," *SPIE/OSA Optical Data Storage Conference*, Feb. 1991.

Is The Bang Worth the Buck? A RAID Performance Study

510-82
83192

Susan E. Hauser, Lewis E. Berman, George R. Thoma
Lister Hill National Center for Biomedical Communications
National Library of Medicine
Bethesda, Maryland 20894
hauser@nlm.nih.gov
Tel: 301-435-3209
Fax: 301-402-0341

Abstract

Expecting a high data delivery rate as well as data protection, the Lister Hill National Center for Biomedical Communications procured a RAID system to house image files for image delivery applications. A study was undertaken to determine the configuration of the RAID system that would provide for the fastest retrieval of image files. Average retrieval times with single and with concurrent users were measured for several stripe widths and several numbers of disks for RAID levels 0, 0+1 and 5. These are compared to each other and to average retrieval times for non-RAID configurations of the same hardware. Although the study is ongoing, a few conclusions have emerged regarding the tradeoffs among the different configurations with respect to file retrieval speed and cost.

Rationale and goals

The Lister Hill National Center for Biomedical Communications, a research and development division of the National Library of Medicine, procured a Sun SPARCstorage Array (SSA), model 101, to house image files for prototype image delivery applications. The SSA model 101 is configured with eighteen Seagate ST31200W 1.05 GB disks connected to six internal fast wide SCSI busses. The SSA is connected to a Sun SPARCstation 20 via a Fiber Channel port. SPARCstorage Volume Manager software supports use of the SSA as independent volumes or as:

RAID 0: Data is split into equal sized blocks, or stripes, and distributed among the disks in the RAID volume.

RAID 1: All data in a volume are duplicated on the mirror volume.

RAID 0+1: Both the original volume and the mirror volume are striped.

RAID 5: In addition to data blocks, RAID Level 5 includes parity blocks, which are distributed among the disks in the RAID volume [1,2].

The specifications of the Seagate disks [3] in the SSA cite a data transfer rate of 3.3 to 5.9 MB/sec. The fast wide SCSI interface has a data transfer rate of 20 MB/sec, and the Fiber Channel connector has a data transfer rate of 25 to 50 MB/sec. Those specifications the following statements from a technical white paper led us to expect very high data retrieval rates in addition to the data security available from RAID.

“Each of the disks in a stripe are generally assumed to be on their own independent data channel, allowing the transfer rate of a RAID 0 implementation to approach the sum of the transfer rates of each of the drives.” [4]

“ Both SPARCstorage Array models ... are capable of over 2000 two-KB input-output operations per second, and sustained transfer rates exceeding 15 MB/second.” [4]

One goal of the study was to determine the optimum configuration and stripe width for fast retrieval of a variety of file sizes. Documentation from Sun [5] and other sources [1] mention the importance of “tuning” the RAID to the data and application through choices in RAID level and stripe width. Yet the guidelines for selecting these, especially for selecting stripe width, are general. One suggestion is to set the stripe width to be the length of a disk track. However, although the specifications of the Seagate drives in the SSA state that the average is 84 sectors per track, one can deduce from those specifications that the track length varies from about 72 sectors per track to about 127 sectors per track. Another suggestion is to select the stripe width such that the stripe width times the number of disks exactly matches the size of the I/O requests at the application layer. However, the SSA is intended for use with applications that read entire files of a variety of sizes into memory at once.

Another goal of the study was to determine the optimum configuration of the SSA for rapid retrieval of files by the Medical Information Retrieval System (MIRS) server

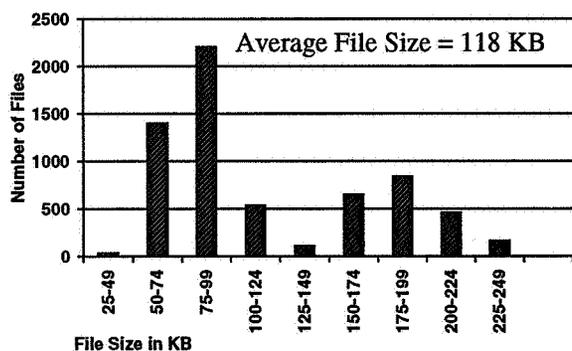


Figure 1. MIRS File Size Distribution

program. MIRS is a client/server application that provides Internet access to biomedical databases, including X-ray medical images [6]. The SSA stores lower resolution gif format versions of high resolution digitized X-ray images. One goal is to quickly display several of the lower resolution images that match a patron’s search criteria. The application reads appropriate images into server memory where they are concatenated and transmitted to the client as one file. The distribution of MIRS image file sizes is shown in Figure 1.

Study conditions

Six of the eighteen disks in the SSA, each attached to a separate SCSI bus, were used for the study. We measured performance of the RAID subsystem alone by removing such factors as reading from cache or swap space, and heavy system loads. The study measured the average time to read files from the SSA into system memory. The study concentrated

on measuring retrieval times for a single process reading files sequentially, and retrieval times for multiple processes reading files concurrently, varied by differing RAID configurations and stripe widths.

Preliminary study

In the first study we assumed that retrieval time for sequential reads was based on two performance components, Average Positioning Time and Data Transfer Rate [7], and attempted to determine these two performance indicators for the various configurations. This was done by measuring the average retrieval time for various file sizes and performing linear regression of average retrieval time as a function of file size. Two results of the linear regression are intercept, which translates to Average Positioning Time, and slope. The units of slope are seconds per byte, so the inverse of slope is the Data Transfer Rate in bytes per second.

To measure retrieval time for concurrent reads, the average retrieval time for a mix of file sizes was measured while none to several other processes were also retrieving files of the same mix of file sizes. In this case, linear regression was performed on average retrieval time as a function of the number of concurrent processes. The calculated slope of the linear relationship is the Average Additional retrieval Time per File per additional concurrent Process. Average Additional Time per File per Process is our performance indicator for concurrent processes.

The three performance indicators were measured for several configurations of the six disks in the SSA for two ranges of files sizes. The smaller range, from 50 KB to 275 KB was similar to the range of files used by MIRS. The larger range, from 1 MB to 12.5 MB, was used to determine if the optimum configuration depended on file size.

For most RAID configurations that were measured, narrower stripes yielded larger data transfer rates for sequential reads for both small and large files. Wider stripes resulted in lower data transfer rates for sequential reads, but also less additional retrieval time per file per concurrent process. The generalization holds for the case where the six drives are configured as independent non-RAID, or “simple”, volumes. A simple volume can be considered as a volume with one very wide stripe. As simple volumes, the six drives had the lowest data transfer rate and the lowest additional time per file per concurrent process. The results suggest there is a tradeoff between optimizing for sequential reads and optimizing for concurrent reads.

We also found that stripe widths less than 16 KB gave results similar to 16 KB, and stripe widths greater than 160 KB gave results similar to 160 KB. Between these two widths the changes in data transfer rate and average additional time per file per process appeared to be a monotone decreasing function of stripe width.

The maximum system throughput measured was 8.2 MB per second, which occurred with 8 processes concurrently retrieving unique files with an average size of 6.42 MB. When 6

processes retrieved files with an average size of 123 KB, the system throughput was 3.9 MB per second, the maximum measured for the smaller file sizes. These were both achieved by distributing files between two 3-disk RAID 0 volumes with a 160 KB stripe.

There were large differences in Average Positioning Time and Data Transfer Rate between the data from the small and large files sizes for a given RAID configuration. For a 6-disk RAID 0 volume and the larger files, the calculated Data Transfer Rate ranged from 5.6 MB per second to 8.3 MB per second. For the smaller files, the calculated Data Transfer Rate ranged from 3.1 MB per second to 4.8 MB per second. For both sizes, Data Transfer Rate decreased with increasing stripe width. Average Positioning Time also varied by several hundred percent, but did not appear to be a function of configuration or stripe width. We concluded that the combined effect of zone bit recording [8,9] and data striping disallowed a simple linear relationship between file size and retrieval time.

Procedures for successive studies

With knowledge gained from the initial study, we modified our performance indicators and proceeded to study the SSA performance for file sizes in the range of the MIRS data, knowing that conclusions would include a caveat about file size. The performance indicators became the Average Retrieval Time and, again, the Average Additional Time per File per Process. Average Retrieval Time is the average time to read a file into memory as measured from a single process sequentially reading files of all sizes. Average Additional Time per File per Process is the same as for the preliminary study.

A typical test set consisted of the following steps:

1. Create a volume or volumes in the configuration to be measured.
2. Fill the volume(s) with files in ten sizes from 50 KB to 275 KB. Use an equal number of files of each size, for an average file size of 162.5 KB. To minimize the effect of zone bit recording, distribute files of each size over all portions of the volume.
3. Create one randomized list of all files on the volume(s). Create twelve randomized lists, each containing approximately one twelfth of the files on the volume(s) and an equal number of each file size.
4. To determine Average Retrieval Time, a program sequentially reads every file in the one large randomized list into memory, measuring the time required to open the file and read in into memory. When all of the files are read, the program calculates the mean retrieval time, standard deviation, maximum and minimum. The sample size, calculated statistics and time of day are recorded in an output file. The program is run several times for a total sample size of several thousand.

5. To determine Average Additional Time per File per Process, a program reads all of the files from the first small randomized list onto memory, measuring the time to read each file. Then two programs run concurrently, each reading files from a different small randomized list. Then three programs run concurrently, each reading a different list of files, and so forth up to twelve programs. The same statistics described above are recorded by each program in an output file. The series is run several times for a total sample size of several hundred for each case.

Average Retrieval Time is the grand average of all the runs using the one large list of files. Average Additional Time per File per Process is determined by first calculating the grand average retrieval time for each case of concurrence, then performing a linear regression of average retrieval time as a function of the number of concurrent processes. The slope of the line returned by the regression is the Average Additional Time per File per Process.

Results

Using the procedures outlined above, we obtained the two performance indicators for the following configurations:

- Three simple volumes
- Six simple volumes
- RAID 0 volumes with 4, 5 and 6 disks
- Two 3-disk RAID 0 volumes
- RAID 5 volume with 6 drives
- RAID 0+1 volume with 6 drives (3 drives, mirrored)

Because of the information obtained in the preliminary study, we used only two stripe widths for the RAID configurations tested: 16 KB and 160 KB.

Figures 2 and 3 compare the results from RAID 0 volumes with 4, 5 or 6 disks. Average Retrieval Time is smaller for the narrow stripe width and also for fewer disks in the volume. Conversely, Average Additional Time per File per Process is smaller for the wider stripe width and for more disks in the volume. Again we see a potential tradeoff between optimizing for a single process and optimizing for concurrent processes.

Figures 4 and 5 compare the results from three configurations using six disks. Although two of these are RAID, none offer fault tolerance. The tradeoff between narrow and wide stripe width is still evident. Although either of the RAID configurations is faster for single processes, configuring the six disks as simple volumes is better for concurrent processes.

Figures 6 and 7 compare the results for the two configurations of six disks that offer fault tolerance to the results for six disks as simple volumes. The mirrored, striped volume (RAID 0+1) offers speed comparable to simple volumes plus the security of data redundancy, at the cost of requiring twice as much media.

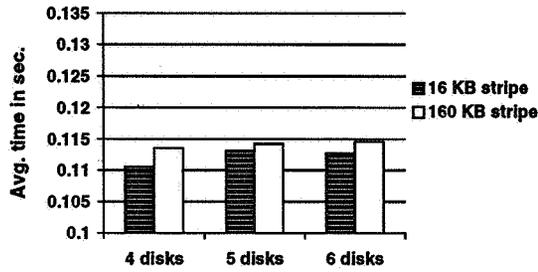


Figure 2. Average Retrieval Time, RAID 0, 3 volume sizes

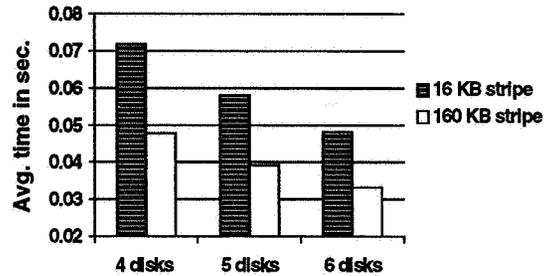


Figure 3. Additional Time per File per Process, RAID 0, 3 volume sizes

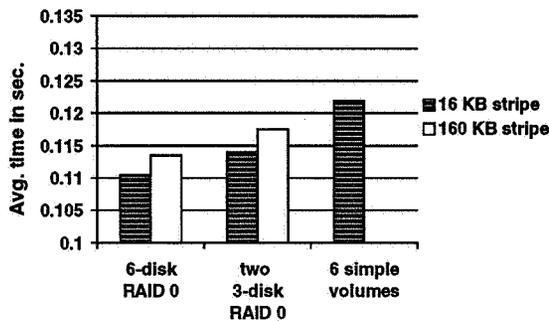


Figure 4. Average Retrieval Time, 6 disks in 3 configurations

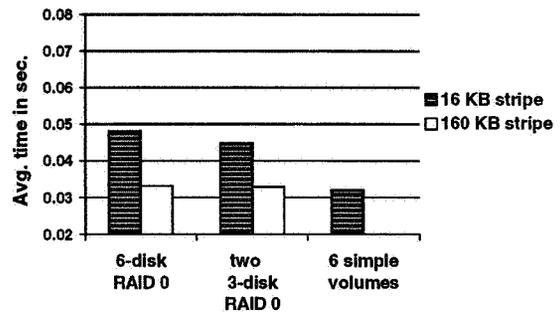


Figure 5. Additional Time per File per Process, 6 disks in 3 configurations

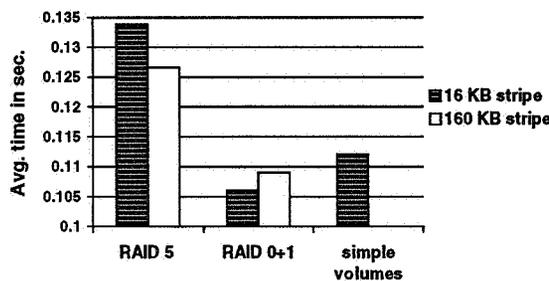


Figure 6. Average Retrieval Time, 6 disks in redundant configurations

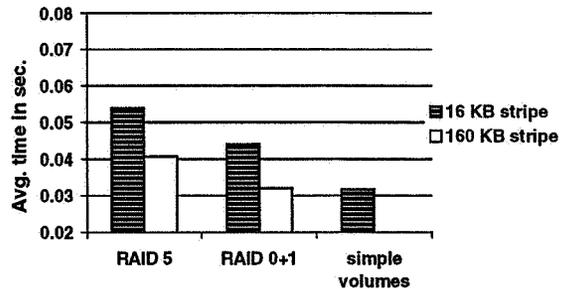


Figure 7. Additional Time per File per Process, 6 disks in redundant configurations

The maximum system throughput achieved during these tests was 4.3 MB per second, which is less than the specified maximum transfer rate of a single drive. That occurred for the RAID 0+1 configuration with 160 KB stripes, with 12 concurrent processes retrieving files with an average file size of 162.5 KB. Evidently, for files of this size the combined latencies of disk drive, SCSI and Fiber Channel interfaces and operating system overhead are great enough to counterbalance increased data transmission rates.

Configuration selection

Even if it is anticipated that access to the dataset will always be sequential reads by a single process, the choice of configuration may not be trivial. If fast retrieval is needed at

any cost, RAID 0+1 provides the fastest sequential retrieval times, excellent concurrent performance and the security of mirrored data. If cost is a consideration and some fault tolerance is required, RAID 5 is the only choice, even though it is not among the best performers for either sequential or concurrent retrieval. If cost is a consideration and fault tolerance is not, distributing the dataset across 4-disk RAID 0 volumes is a good choice.

It is more difficult to determine the optimum configuration if both sequential and concurrent access to the dataset is anticipated. In this case, cost, fault tolerance and system management issues may be more important than retrieval time, especially for small files.

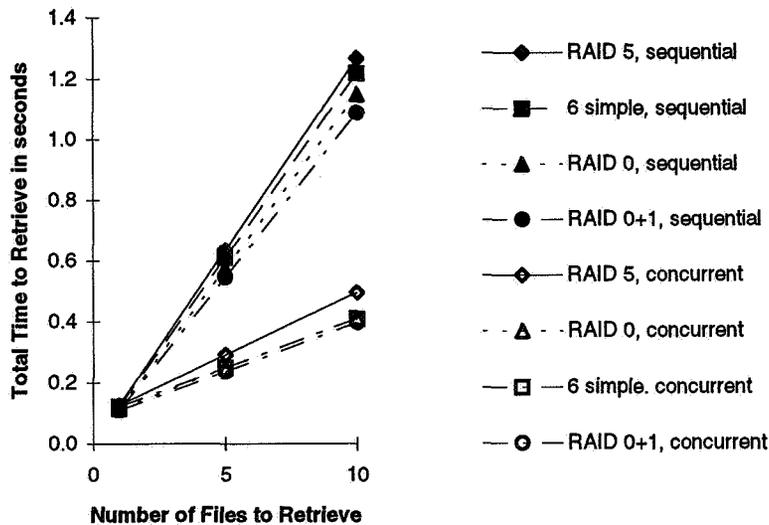


Figure 8. Total Retrieval Time of Average Size Files

Figure 8 plots total retrieval time as a function of the number of 162.5 KB files retrieved for four six-disk configurations. It shows the total time for sequential reads and the total time for concurrent reads. The lines on the graph are in the same order as the descriptions in the legend. For the occasional retrieval of a single file, any configuration of the

array yields about the same retrieval time. For applications that retrieve several files at a time, concurrent access improves total retrieval time more than any RAID configuration. This is illustrated by comparing the slowest concurrent access case, RAID 5, to the fastest sequential access case, RAID 0+1. However, the difference between the best and worst configuration for either kind of access is less than 1.2 seconds for up to ten files, which may be inconsequential for many applications.

What about “Bang” and “Bucks”?

We define:

$$\text{Bang} = \frac{\text{Average file size}}{(\text{Average Retrieval Time} + \text{Average Additional Time per File per Process})}$$

Bang increases with improvement in either of the performance indicators, and gives equal weight to each. The average file size in the numerator balances the larger performance indicators that would result from larger files. This definition of Bang is only useful for

quantifying file retrieval speed. It provides no information regarding write speed, or security or system reliability, which are less easily quantified. Table I shows Bang calculated for the configurations in this study, in order of decreasing value of Bang. The average file size for the study was 162.5 KB.

Table I: Bang for several configurations

Configuration	Stripe width	Average Retrieval Time (sec)	Additional Time per File per Process (sec)	BANG
3-disk RAID 0+1 (mirrored)	160 KB	.109	.032	1.152
6-disk RAID 0	160 KB	.115	.033	1.098
3-disk RAID 0+1 (mirrored)	16 KB	.106	.044	1.083
two 3-disk RAID 0s	160 KB	.118	.033	1.076
5-disk RAID 0	160 KB	.114	.039	1.062
6 simple volumes	NA	.122	.032	1.055
two 3-disk RAID 0s	16 KB	.114	.045	1.022
6-disk RAID 0	16 KB	.113	.048	1.009
4-disk RAID 0	160 KB	.114	.048	1.003
6-disk RAID 5	160 KB	.127	.041	0.967
5-disk RAID 0	16 KB	.113	.058	0.950
3 simple volumes	NA	.116	.057	0.939
4-disk RAID 0	16 KB	.110	.072	0.893
6-disk RAID 5	16 KB	.134	.054	0.864

The price for the SSA model 101 with 18 1.05 Gbyte disks, and SBUS to Fiber Channel host adapter was \$26,733. The same hardware capabilities without the RAID management features would have been approximately \$20,000. Each disk provides approximately 863 MB of space for user data, whether formatted as a simple volume or as part of a RAID volume. Thus 18 disks offer a total of 15.534 GBytes of data storage when configured as simple volumes or as RAID 0, 12.945 GBytes when configured in 6-disk RAID 5 volumes, or 7.767 GBytes when configured in 3-disk RAID 0+1 volumes. Table II shows the calculated Bang per Buck per Gbyte of data storage, for four configurations of the SSA or of the equivalent hardware.

Table II: Bang per Buck per Gigabyte

	Bang	Thousands of \$ per Gbyte of data storage	Bang / K\$ / GB
3-disk RAID 0+1	1.152	3.442	0.335
6-disk RAID 5	0.967	2.065	0.468
6-disk RAID 0	1.098	1.721	0.638
simple volumes	1.055	1.545	0.683

Summary and conclusions

For retrieval of files of a few hundred KB or less, Bang alone is not worth the Bucks. RAID offers many other attractive features, such as fault tolerance, ease of storage management, and, in many cases, a compact, well designed peripheral. If the subsystem is just one component of a large system, the extra cost of RAID may be worth these conveniences alone.

For the MIRS application, where a set of files between 50 KB to 275 KB must reside in fault tolerant storage that maintains the retrieval speeds that are available from hardware alone, there is no choice but RAID 0+1, even though it is expensive. RAID 0+1 is also the choice if fast retrieval is of primary importance and cost is not. Although narrow stripes produce slightly faster sequential retrieval times and wide stripes produce slightly faster concurrent retrieval times, the performance difference between wide and narrow stripes for this range of file sizes is so small that any choice would be acceptable. We recommend that the database of lower resolution MIRS images reside on a RAID 0+1 volume with a 16 KB stripe width. Because the MIRS application software reads images sequentially, the narrow stripe should give slightly faster retrieval times.

For applications where fault tolerance is required, and retrieval speeds can be slower than those available from hardware alone, RAID 5 is the best choice. For RAID 5, wider stripes appear to improve both sequential access speed and concurrent access speed for files in this range.

If either RAID 0+1 or RAID 5 is selected, retrieval times may be faster for more or fewer than six disks per volume. We plan to measure the performance several configurations in the next phase of the study.

For applications where fault tolerance is not important and funds are limited, balancing the load across several volumes without the benefit of RAID management can yield fast retrieval speeds at significant cost savings.

We find no reasons for choosing RAID 0 for applications involving small files. The slight performance advantage for sequential file retrieval is offset by the cost of the RAID management capabilities and the reliability risk incurred by distributing each file across several disks.

References

1. National Peripherals, Inc., "An Introduction to Disk Arrays and RAID Levels."
2. Procom Technology, Inc., http://www.procom.com/homepage/raid_def.html, RAID definitions, 1996.
3. Seagate Technology, Inc., <http://www.seagate.com/tech/techttop.shtml>, Technical specifications for drive model ST31200W, 1995.
4. Sun Microsystems, Inc., "The SPARCstorage Array Architecture, Technical White Paper," February, 1995.
5. Sun Microsystems, Inc., "SPARCstorage Array Performance Brief, Technical White Paper," Revision 2, July, 1994.
6. Long LR, et.al., "A Prototype Client/Server Application for Biomedical Text/Image Retrieval on the Internet," Proceedings of SPIE Storage and Retrieval for Still Image and Video Databases IV, San Jose, CA, February 1-2, 1996, pp.362-372.
7. Chen PM and Lee KL, "Striping in a RAID Level 5 Disk Array," Technical Report CSE-TR-181-93, University of Michigan, November 1993.
8. Sun Microsystems, Inc., Technical Product Marketing, "Configuration Planning for Sun Servers," Third Edition, January, 1994.
9. Van Meter R, et.al., comp.arch.storage FAQ, Subject [6.1], June, 1996.

511-82

83193

The Medium is NOT the Message
OR
Indefinitely Long-Term File Storage at Leeds University

David Holdsworth
Computing Service
Leeds University
LEEDS LS2 9JT
United Kingdom
dholdsworth@leeds.ac.uk
+44-113-233-5402
+44-113-233-5411

Abstract

Approximately 3 years ago we implemented an archive file storage system which embodies experiences gained over more than 25 years of using and writing file storage systems. It is the third in-house system that we have written, and all three systems have been adopted by other institutions.

This paper discusses the requirements for long-term data storage in a university environment, and describes how our present system is designed to meet these requirements indefinitely. Particular emphasis is laid on experiences from past systems, and their influence on current system design. We also look at the influence of the IEEE-MSS standard.

We currently have the system operating in 5 UK universities. The system operates in a multi-server environment, and is currently operational with UNIX (SunOS4, Solaris2, SGI-IRIX, HP-UX), NetWare3 and NetWare4. PCs logged on to NetWare can also archive and recover files that live on their hard disks.

Background

The earlier part of our experiences has a rather UK-specific flavor. Our 1968-1972 system had a large in-house element and ran on an English Electric KDF9 system[1]. From 1972-1980 we used ICL's George3 system[2], where the two-level filestorage was part of the system. In addition to on-line files, off-line files on half-inch tape were still part of the file system.

Our procurement of a system for the 80s brought us into contact with some of the harsh realities of the file systems of the time - particularly so as the decision was to go for the then rather new VM/CMS. This led to a second in-house system - known rather unimaginatively as the Leeds Filestore, and used at the universities of Reading and Warwick in the UK, and also University College, Dublin, and the Technical University in Braunschweig in West Germany.

Other UK universities had similar experiences, and so in 1990 a self-appointed working group formulated a set of requirements, but failed to find a product that met them.

The major points were:

- Files can be sent to the archive from any of the participating file systems on the campus.
- Recovery of a file can be onto any system, not necessarily the originating system.
- The retention of indexing information is done by the system.
- It should be easy for an end-user to rename files.
- The overheads per file must be very small, as many of the files are themselves small.
- The system should be able to exploit new storage technology seamlessly.
- The system should cope with data for a shifting population of many thousands of users.
- Data should be safe.
- There should be no reliance on operating system modifications.

We at Leeds implemented a system to meet the most important of the requirements, one of which was that we wished never again to need a new system. The resulting system and the experience that led to it form the subject of this paper. Like its predecessor this system has never really been given a name, so for now we shall merely call it the LEEDS system, claiming the acronym *Leeds Ever-lasting Extensible Data Store*.

The requirement is for something rather less than direct access to MSS volumes from applications programs. We need an MSS-type system into which users can consign their files for safe keeping. The actual processing of data by end-users will be in the form of files in “standard” manufacturers’ operating systems. Most of the helpful concepts from IEEE-MSS are actually from version 4. The drift in version 5 seems to be more towards end-user or applications programs being aware of the existence of MSS media.

Lessons from the past

Identification of users

Organisations change on timescales which are short compared to the lifetime of data. Departments get restructured, and user-names sometimes change, either as a result or because of name changes. In the past we have had a hierarchy of user naming based on departments (George3), and we have also labelled data on off-line media (tape) with user-names. We do neither of these things now, although Novell’s NDS is pushing things in the direction of user hierarchy, and eroding the importance of the internal object ID. The LEEDS system uses an internal ID for each individual, and there is evidence from BrainShare[3] that Novell are also moving back in that direction.

Staging of files

With George3 we had a system in which the user need not be aware whether a file was on-line; a request to open a file that was not on-line brought it on-line transparently. This transparency is actually very visible in the time domain. Users need to be able to stage requests for their files ahead of needing them. Armstead and Prahst[4] report the same lesson. In our systems since 1980 we have gone to the point of treating staging as the norm. Files are not automatically migrated just by referring to them. Where users have interactive access to the file system this has proven not to be a problem.

Naming Systems

Not only do organisations change, but the IT equipment changes. Our previous file archives were primed with the data from their predecessors. They were of course associated with the main frame systems on which they ran. We have now got a system in which the archive is a separately identified name-space, and it records from which system each file came. Any file can be recovered onto any currently existing system. This neatly deals with access to data from systems which no longer exist.

Keeping Data Forever

When we abandoned the KDF9 for George3, we transferred only material which seemed to be useful - mainly source text of programs and cosmic ray data belonging to our physics department. Material such as assembly code (including the system itself) was discarded as useless. Some years later, the Science Museum in London looked at the possibility of recreating computing of the 60s by emulation. We were asked if we had the capability to provide the system software in machine readable form.

The total disk storage on the KDF9 was 48 Mbytes, and the total file store was about 10 times that size. The cost of keeping it for ever would now be all but zero. Of course, its value would also be zero if it were not indexed in some way. Such is the advance of storage technology that it is not cost effective to discard old data if it is held on modern media. One copy of all the data from our now discarded VM/CMS system occupies 14 volumes in our EXB-120, compared to the 1102 half-inch tapes that previously held the same data.

Our philosophy has always been to preserve the data and not the medium onto which it is written. Our ambition is for an environment where users need not feel the need to delete data just to recover disk space.

Our present system is already designed to drive multiple robots of potentially different technologies, in such a way that data migrates automatically and routinely onto new media (see *Robotics* below).

Integration with back-up

We have learnt that there are advantages in system integrity when the archive is integrated with the back-up system. This has been the case with all systems up to (but not including) the present one. We have also learnt that there is wide-spread and vehement disagreement on this issue.

Size of index information

We had experience (with George3) of a system in which the amount of index information associated with data increased as the data became older. This only became a problem after the system had been running for about 6 years. We have thus always been careful to avoid indexing by use of structures which have the capability to grow faster than linearly with the amount of data in the system.

Format of off-line media

Our transfer of data from the VM/CMS archive relied heavily on our knowledge of the data format on the tapes. (We wrote and owned the software.) Some UK universities

used a bought-in system for which they were unable to obtain specifications of tape format, leading to a hiatus at the end of the lives of the CMS systems. The moral is that when buying in archive systems, the knowledge of the format of the data on the media should be part of the contract.

Overview of the current LEEDS system

End-users choose to move their files into the archive, or to recover them from it. The archive needs to be made aware (by the system management) of the existence of the domain in which the user has an authorised user-ID. It is this authorisation which controls access to the system. There is no need for a separate end-user registration for the archive (although it could be managed in this way by an installation that had that policy).

We thus think in terms of an archive server which is aware of a number of client file domains. Each of the client file domains is itself a name-space with a number of servers. UNIX file systems are accessed using NFS, and NetWare clients are accessed using FTP. There are no modifications to the operating systems of the client file systems.

Notification of a request is by placing a small file in a key directory. There is one such directory for each client domain. The archiver machine (a SPARC-20) polls these directories at regular intervals. Although this mechanism was initially thought to be an interim, awaiting a more elegant solution, it has stood the test of time, and we now have no plans to change it. Figure 1 gives a simplified schematic diagram of the archiver's position in the Leeds University installation. There are actually many more servers and domains. The section on *System Integration* below gives more detail.

Naming and indexing

The bitfile concept of Mass Storage System Reference Model Version 4 (MSSv4)[5] introduces an abstraction which, for us, neatly highlights the media independence of data objects (such as users' files). It is at the heart of our current approach that an ordered sequence of contiguous bytes is the basic object of data retention. The bitfile-ID associated with it provides the handle by which it can be located on an appropriate storage volume, whereas other indexing activity maps a user's view of the object's name to its bitfile-ID.

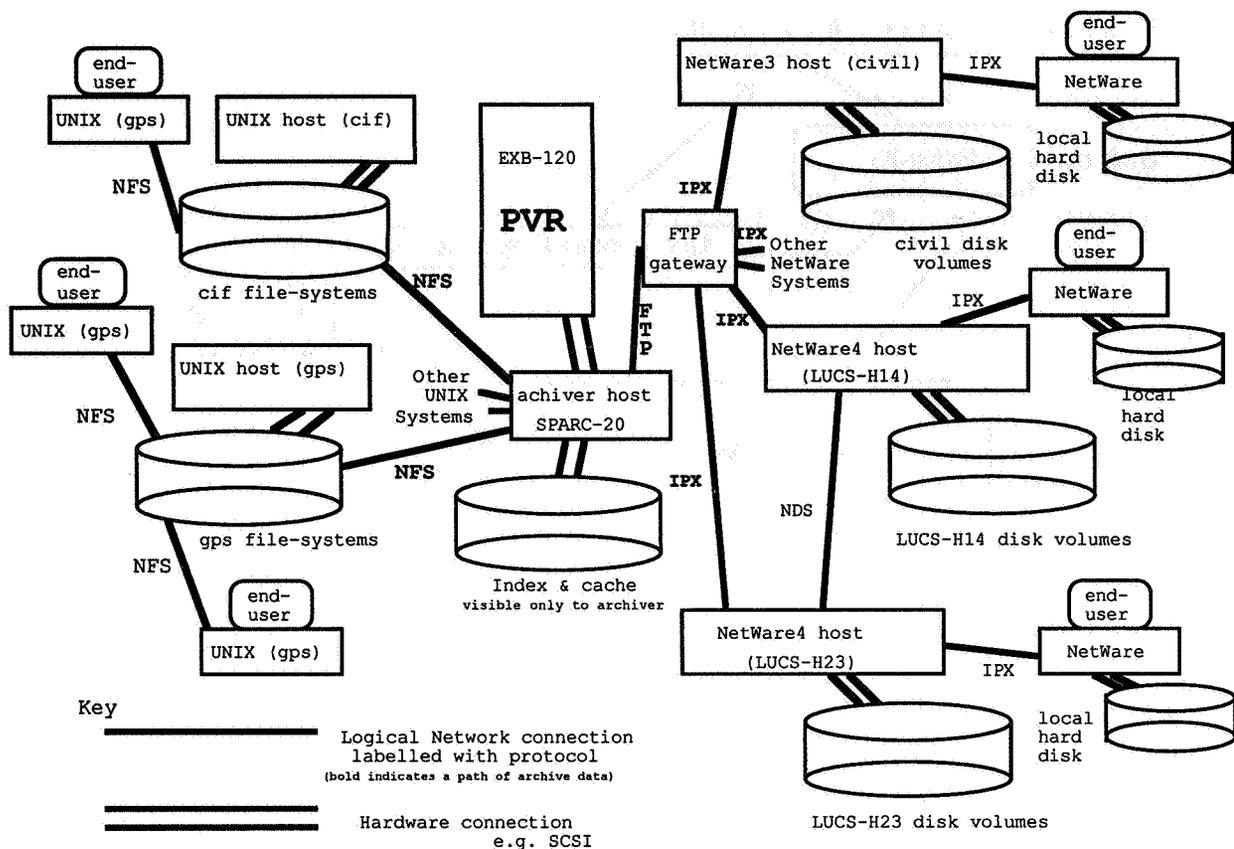


Figure 1: Schematic diagram of the archiver's position in the Leeds University installation

The naming convention of Mass Storage System Reference Model Version 5 (MSSv5)[6] is based around the all-embracing Storage Object ID (SOID). The SOID concept covers the naming of all storage objects, both physical and abstract. The SOIDs are typed, and thus the bitfile-ID concept has evolved into one particular type of SOID. The design of the LEEDS system is centred on the retention of virtual storage objects, and so continues to use the term bitfile-ID for the SOID of a Virtual Storage Object. Figure 2 shows the mapping process from end-user's name to data on a storage volume.

It seems that our preference for centrality of the bitfile concept is shared at CERN[7].

User name space

We next look at the user name space, and the mapping of file names as understood by end-users into bitfile-IDs.

The end-user sees the world in terms of user-names, each of which exists in a particular domain. Each user-name on a particular system is seen as having a filestore tree. We find that this abstraction fits well with UNIX and with NetWare, and it is clear that some other systems can also fit this model. The indexing in the LEEDS archive operates in two places. There is a mapping between bitfile-IDs and their corresponding file names as perceived by the end-user, i.e. a system-name, path-name pair. This mapping can be extracted by the end-user as a browsable file.

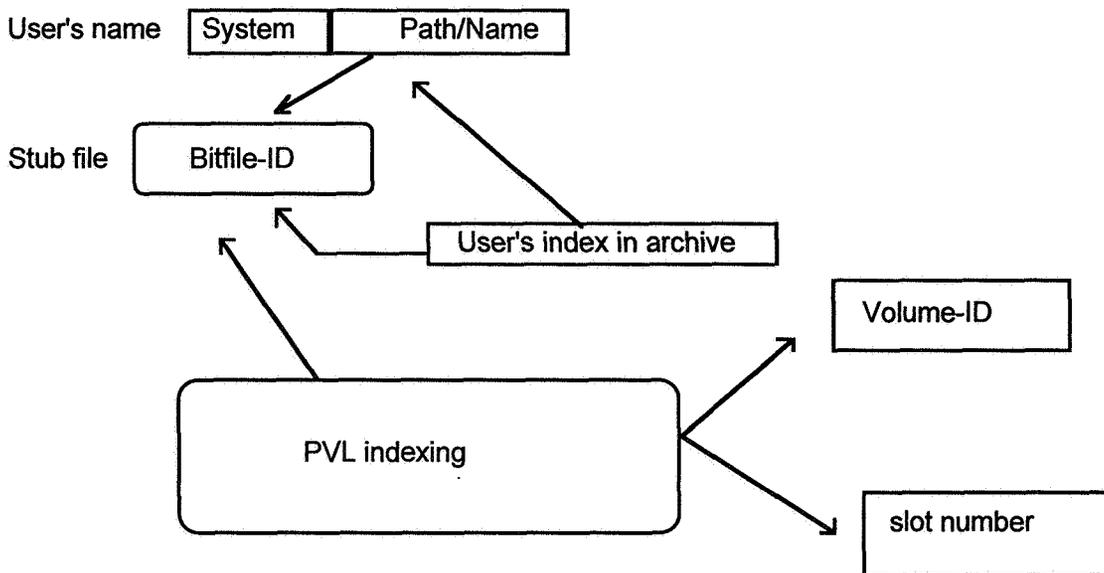


Figure 2: Name mappings

When a file has been migrated to the archive, it is replaced by a stub (of 120 bytes) which contains the bitfile-ID. This stub provides an alternative mapping between the user's view of the filename, and the real file. The stub can be copied or moved around - even between systems. It is a normal text file. It can be recreated from the indexing data held in the archive. This stub is then used as the argument to the recovery operation in order to reverse the migration process (see *End-User Operation* below).

System name space

There are 6 vital flat name-spaces. In terms of the MSSv5 model, each would appear to correspond to a particular type of SOID.

The *bitfile-ID*s are never re-used. The format is 14 letters - although the magic number 14 is in a single `#define`. This gives 26^{14} ($= 6.45 \times 10^{19}$) names with the current format.

A *volume-ID* of 8 characters is assigned to each near-line or off-line volume (currently 8mm helical scan tape). Again the length is in a single `#define`.

A *media access point* (8mm tape drive in the present implementations) is named by a single letter of the alphabet. At present the limit to 26 such devices does not seem to constrain our ambitions. It is also the same as the "mount point" of MSSv5, as we have not covered the distinction between a cartridge and a volume.

A *near-line volume location* is an integer, and corresponds somewhat with the notion of slot in MSSv5. Each of these integers addresses a single "virtual slot" in the Physical Volume Library (PVL). Although we currently have only a single media domain, the design of the API for driving the robotics (= PVL) provides an abstraction which allows for addition of extra domains, of possibly different media types (see *Robotics* below). A volume will have no near-line volume location when it has been removed, but the system retains knowledge of its existence, and its contents.

Internal user-IDs are integers. In practice large institutions like universities already assign unique numbers to the individuals with whom they are involved. Each of the five user sites has been able to integrate this aspect into its existing system for registration of end-users. These integer user-IDs provide the basis for management of ownership and access permissions.

Each *client system* has a *name* which is a character string -- often its IP host name, or yp-domain name. For each client system we maintain a simple file which maps the end-users of that system to the unique user number. Thus an individual may have identities on several systems, but have them map to one common archive owner.

Robotics (Physical Volume Repository/Library)

The robot driving component of the system is driven through an abstract API in which the key system call is for the mounting of the contents of a volume location (i.e. slot) into a particular drive (mount point/media access point). The reply to a call of this API routine has four possible outcomes.

1. OK - volume is mounted.
2. Cannot do it now - should be possible later
3. This operation is not possible
4. Hardware malfunction

A vital part of this API is the possibility of the reply that the requested mount can never succeed. This allows our simple naming scheme to work with multiple robots of mixed technologies, by building an implementation of the API in which each separate robot has associated with it a subset of the drive letters, and a subset of the (PVL) slots. It also caters for a robot such as the multiple Panasonic MARC machine in which full traverse capability is not available to each robot arm.

The actual robotic hardware in use in the 5 currently operational installations is from Exabyte:

- 2 systems with EXB-120 and 4 x EXB8500c drives,
- 2 systems with EXB-480 and 3 x EXB8500c drives.
- 1 system with EXB-480 and 4 x EXB8500c drives.

For test purposes, we also have an implementation of the PVR for use with a human robot who just loads tapes from a shelf of numbered slots into tape drives following instructions displayed on a screen..

System Integration

Each client file domain contains software (the `rkv` command) which writes queue entries into a directory reserved for the purpose, and each UNIX file system needs to contain a special directory which then links to files which are awaiting access by the archiver.

The main archive system runs on a dedicated SPARC-20 machine, with 8 Gbyte of disk space. The bulk of the disk space is used for cacheing of files in transit. Files newly migrated to the archive reside in the cache, and a periodic dump operation writes such files to tape. A periodic cache purging operation removes files on an LRU basis.

A request to recover a file leads to its being recovered from tape into the cache, and it is transferred from there to the end-user system. In the event that the bitfile is already in the cache, the first stage is omitted.

The UNIX client file domains export their file systems to the archive machine, and so the data is manipulated directly using NFS. The NetWare clients permit read and write to their volumes. The FTP access is via a small gateway server which has a dedicated UTP ether connection to the archiver. This is the only subnet on which the archive NetWare password appears in clear.

End-User Operation

A file (or more usually several/many files) is (are) transferred to the archive by an explicit user command (called `rkv` - chosen so as not to clash with any system's built-in commands, and yet still sounding a bit like the word "archive"). There are three possible operations on a file:

migrate - transfer the file to the archive

recover - get the file back from the archive

back-up - copy the file to the archive

The `rkv` command is available for both DOS and UNIX environments. There is also an add-on to the Windows file manager.

In addition to the three major operations, there are facilities for recreating stub files, and for obtaining directory information.

System Management

The system is designed to run with minimal attention. This is normally the case. If the network is behaving well, the main operational task is the feeding of blank tapes, and the removal of tapes to secure remote storage as a disaster precaution

There needs to be a regime for updating the maps between user identities on client domains and the internal user-ID. The detail of this depends on the site policies.

Data Integrity

The tape handling regime ensures at least 2 copies of each bitfile, with the added requirement that there must be at least 3 copies if there are no disk copies.

The bitfiles held on a tape are in order of bitfile-ID, and contain all the bitfiles in the range. This means that in order to index the total contents of the volume all we need is to

know the starting and ending bitfile-IDs. A separate index for each volume gives the block position of each bitfile. The complete set of bitfiles then resides on a set of tapes, forming a stream of bitfiles. The 3 copies of each bitfile are obtained by having three streams. We replicate the bitfiles themselves, not the volumes on which they reside.

When a file is migrated into the archive, it is written in duplicate onto different disk volumes. The periodic dump to tape extends the shortest of the 3 streams by writing onto a fresh tape, or by a complete overwrite of an obsolete tape. When this is complete one of the disk copies is deleted. Each bitfile is thus copied to tape 3 times, and only after the third time is the remaining disk copy a candidate for cache clearance. The time interval between periodic dumps is chosen so as to match the likely usage level. In the event of the caches becoming unusually full, an extra dump is run. The data is safe against filling of cache residences.

Obviously the dumping process generates a number of part full volumes. There is a copy process (for the most part initiated automatically) which copies multiple input tapes from the same stream onto a single output tape. When the output tape is full, it leaves an overlap in the stream between the output tape and the incompletely read input tape. A subsequent copy operation will then carry on from this point. The live listing of volumes in the Leeds University system can be inspected on the Web site.

An important property of the copying operation is the ability to substitute data from another stream in the event that the tape volume that would naturally be used is not available in near-line storage. One of the streams is routinely held remotely from the main machine room, but its contents can still be copied. Also, when a tape volume fails the procedure is merely to remove the offending volume and then instruct the system to copy it.

Operational Experience

There are currently 26939 user-IDs registered on 18 client domains in the system at Leeds University, which is the most mature of the 5 sites. It holds about 2.7 million files, with a total size of about 0.5 Tbyte. The system is coded to allow access to files which are not held in the robot, and offers a recovery time of about 2 minutes when the queue is empty. There is of course automatic batching of multiple requests from the same volume.

How safe is the data, given that computing hardware malfunctions from time to time? So far we have successfully recovered from all mishaps of this nature. This includes:

- accidental loss of the index partition,
- removal of a cache area,
- accidental corruption of the master table of tape locations,
- a very few tape failures - including two which actually snapped.

Assessment against design goals

Files can be sent to the archive from any of the participating file systems on the campus.

YES and it is open to add a separately managed departmental system, with its namespace distinct from the main campus facility. This is to ensure that management errors on the client system cannot compromise other people's data.

Recovery of a file can be onto any system, not necessarily the originating system.

YES but not between departmentally managed systems and centrally managed systems, for the above reasons of data security.

The retention of indexing information is done by the system.

YES

It should be easy for an end-user to rename files.

YES but this is true only for stub files. The name recorded in the archive is always the name that the file had when it was archived.

The overheads per file must be very small, as many of the files are themselves small.

YES - approx. 60 bytes + pathname of the file

The system should be able to exploit new storage technology seamlessly.

YES but not well tested. The implementation of human robot and of EXB-480 went smoothly. There is a present assumption of reading and writing of blocks on the storage medium via the UNIX driver, and also of FSR (forward skip).

The system should cope with data for a shifting population of many thousands of users.

YES - there are currently 26939 user-IDs registered on 18 client systems.

Data should be safe.

YES - replication and recovery techniques have been used in live situations

There should be no reliance on operating system modifications.

YES - The archiver machine in vanilla Solaris2, and all the client systems are also unmodified.

Assessment against IEEE-MSS standard

Our two major omissions are in relation to virtual storage objects, and PVR cartridges.

Firstly, our bitfiles are constrained to be a contiguous sequence of bytes, not the multi-segmented virtual storage objects of MSSv5. However, in defence we would argue that any structure can be mapped onto a contiguous sequence of bytes, and so why stop there. One could offer the whole panoply of indexed sequential access - but it would be a mistake.

Secondly, our volumes are not housed in cartridges. Each volume is only one mountable object. This will make the driving of some types of device rather contorted, but not impossible.

Because the design of the API for access to near-line volumes was designed before the release of MSSv5, the correspondence to the PVL/PVR structure is not quite total. There is an identifiable part of our system which is the *locator* and identifies the volume-ID and its slot number. This slot number is then presented to a mount request which is best thought of as a request to the PVL. If the drive and slot number in this request are in the same PVR, this request will succeed (hardware malfunction permitting). If the drive and slot number are not in the same PVR the reply is such as to cause the system to try other drives until the operation succeeds.

Conclusion

We do not have peta-bytes of data, but we have quite a lot, and it goes back in time. We do have a large floating population of users, and the lapse of time means that systems come and go. Our techniques enable meaningful long-term data storage, and we have a thoroughly operational system running on 5 sites.

Web Site

The WWW site gives a potted system description, more historical information, and some access to live information on the Leeds University installation. The URL is:

<http://www.leeds.ac.uk/ucs/systems/archive>

References

- [1] M Wells, D Holdsworth, AP McCann, "The Eldon2 Operating System for KDF9", *Computer Journal* vol14 no1 (1971)
- [2] G.B.Newell, "George 3, The Compleat Operating System", *J.Inst. Comput. Sci.* vol3 no3 (1972) *also* International Computers Ltd, "Operating Systems George 3 and 4", Technical Publication 4267, 4th edition (1971)
- [3] Dave Eckert, "NetWare Directory Services (NDS) Futures", DSS201T *BrainShare '96*, Novell Inc 1996
- [4] Betty Jo Armstead, Stephen Prahst, "Implementation of a Campus-wide Distributed Mass Storage Service: The Dream vs. Reality" *Fourteenth IEEE Symposium on Mass Storage Systems* IEEE 1995
- [5] Sam Coleman, Steve Miller, Eds., "Mass Storage System Reference Model: Version 4" IEEE draft 1991
- [6] Lester Buck, Sam Coleman, Rich Garrison, Dave Isaac, Eds., "Reference Model for Open Storage Systems Interconnection" Project 1244 - IEEE 1994
- [7] Jamie D Shiers, "Data Management at CERN: Current Status and Future Trends" *Fourteenth IEEE Symposium on Mass Storage Systems* IEEE 1995

Analysis of the Access Patterns at GSFC Distributed Active Archive Center

512-82
83194

Theodore Johnson¹
Dept. of CISE, University of Florida
Gainesville, FL 32611
AT&T Research
Murray Hill, NJ 07974
ted@cis.ufl.edu

Jean-Jacques Bedet
Hughes STX Corporation
7701 Greenbelt Rd., Suite 400
Greenbelt, MD 20770
bedet@daac.gsfc.nasa.gov
phone: 301-441-4285

Abstract

The Goddard Space Flight Center (GSFC) Distributed Active Archive Center (DAAC) has been operational for more than two years. Its mission is 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. Over 550,000 files and documents have been archived, and more than six Terabytes have been distributed to the scientific community.

Information about user request and file access patterns, and their impact on system loading, is needed to optimize current operations and to plan for future archives (i.e., EOS-AM1). To facilitate the management of daily activities, the GSFC DAAC has developed a data base system to track correspondence, requests, ingestion and distribution. In addition, several log files which record transactions on Unitree are maintained and periodically examined.

This study identifies some of the users' requests and file access patterns at the GSFC DAAC during 1995. The analysis is limited to the subset of orders for which the data files are under the control of the Hierarchical Storage Management (HSM) Unitree. For example, orders on pre-mastered CD-ROMs, which account for a substantial proportion of the total volume of data distributed, were excluded because they are not managed by Unitree. The results show that most of the data volume ordered was for two data products. The volume was also mostly made up of level 3 and 4 data and most of the volume was distributed on 8mm and 4 mm tapes. In addition, most of the volume ordered was for deliveries in North America although there was a significant world-wide use. There was a wide range of request sizes in terms of volume and number of files ordered. On an average 78.6 files were ordered per request. Using the data managed by Unitree, several caching algorithms have been evaluated for both hit rate and the overhead ("cost") associated with the movement of data from near-line devices to disks. The algorithm called LRU/2 bin was found to be the best for this workload, but the STbin algorithm also worked well.

¹ Theodore Johnson is supported by a grant from NASA, #10-77556.

Introduction

On-line scientific archives are playing an increasingly important role in data-intensive research. These archives hide the internal physical organization of the data and automatically migrate files between near-line and on-line (disk) devices, making data more easily accessible. However, building such a large-scale archive can be an expensive proposition and system resources need to be carefully managed. To date, there has been little published research that studies the performance of on-line scientific archives.

The EOSDIS archive is expected to have an ingest rate of Terabytes per day when fully operational. This data will be available on-line for browse, order, and distribution. Careful planning is needed to handle the very large volume of data, the very large number of files, and the expected high user demands. Many studies have been made to predict archive use, based on surveys of the expected users (for example, see the studies at [ESDIS]). However, little empirical evidence has been collected.

In this paper, we first study some of the user-request patterns and their impact on the overall system loading. Rather than examining all orders submitted at GSFC DAAC in 1995, a subset has been selected that has direct impact on the archive controlled by Unitree and the robotic devices. Not all data are stored under Unitree. For example, some data was received on 8-mm tape and never ingested into Unitree because of the substantial effort required. Orders for these tapes are usually simple tape copies and are conducted "off-line" and do not affect Unitree. To satisfy some of the GSFC DAAC users, a large farm of disks has been installed where data can be retrieved via anonymous ftp. These anonymous ftp orders, off-line orders, as well as CD-ROM requests are not used in the analysis.

There will also be presented an analysis of the GSFC DAAC Oracle data base that contains information on the orders and the files requested, as well as the Unitree log files that provides some insight on the mounts and stages operations. Based on the statistics gathered in the analyses, we discuss issues related to the user request and file access patterns, caching, clustering, migration, and system loading. Because the user access pattern is related in part to the data set accessed and because of rapidly changing technology, we do not claim that all future archives will have experiences similar to that of the GSFC DAAC. However, we feel that this study will provide insight into the nature of user access to on-line archives. We make comparisons to a previous study of the NSSDC NDADS archive (see [Jo95]) to point out similarities and differences.

Previous Work

Several studies on the reference patterns to mass storage systems have been published. Smith [Sm81d] analyzes file migration patterns in hierarchical storage management system. This analysis was used to design several HSM caching algorithms [Sm81c]. Lawrie, Randal, and Burton [LRB82] compare the performance of several file caching algorithms. Miller and Katz have made two studies on the I/O pattern of supercomputer applications. In [MK91], they find that much of the I/O activity in a supercomputer system is due to checkpointing, and thus is very bursty. They make the observation that

much of the data that is written is never subsequently read, or is only read once. In [MK93], they analyze file migration activity. They find a bursty reference pattern, both in system load and in references to a file. Additional studies have been made by Jensen and Reed [JR91], Strange [Str92], Arnold and Nelson [AN88], Ewing and Peskin [EP82], Henderson and Poston [HP89], Tarshish and Salmon [TS93], and by Thanhardt and Harano [TH88]. However, all of these studies apply to supercomputer environments, which can be expected to have access patterns different from those of a scientific archive.

The access patterns to the NASA National Space Science Data Center's on-line archive, NDADS, is studied in [Jo95]. However, there are many qualitative and quantitative differences between the NDADS archive and the GSFC Version 0 DAAC. We make comparisons whenever possible between results in this report and the results of [Jo95].

Access and Distribution Methods

The GSFC DAAC receives data from science projects such as Sea-viewing Wide Field-of-view Sensor (SeaWiFS), Coastal Zone Color Scanner (CZCS), Total Ozone Mapping Spectrometer (TOMS), Pathfinder AVHRR (Advanced Very High Resolution Radiometer) Land (PAL), Tiros Operational Spectrometer (TOVS), DAO (Data Assimilation Office), and Upper Atmospheric Research Satellite (UARS). These data are stored in a Mountain Gate Automated tape library system (RSS-600) using VHS tapes and an 1803 Cygnet jukebox using 12" WORM optical media. By submitting requests to the HSM (Unitree), the data can be retrieved to disks and made available to users.

In 1995, there were several ways by which a user could order data. A Graphic User Interface (GUI) based on X-windows, allowed a user to browse, select and order products. A Character User Interface (ChUI) was also available for users with VT100 terminals, but this interface has more limited capabilities. Orders could also be submitted by calling the GSFC DAAC, or by sending a fax, letter or email.

Orders can be filled by sending the data copied to tape (8-mm, 4-mm, 9 track) to the users or by transferring the requested data to a distribution staging area where the user has several days to ftp the files to her own machine. A size limit has been placed on the volume of data that can be transferred via ftp requests because of limited resources (disk space and network). Because of the high overhead associated with the retrieval of small files from the tertiary storage system, some specific datasets are also kept on a large farm of disks (200 GB) and are accessible via anonymous ftp. Some data sets of high demand have also been pre-mastered on CD-ROM for easy and quick distribution. As this study is intended to illuminate the nature of on-line access to tertiary-storage based archives, we limit the set of requests that we analyze to only those that access Unitree. In particular, we exclude requests for pre-mastered CD-ROMS, and accesses to anonymous ftp data, and internally generated requests. Internally generated requests includes testing, and do not reflect the nature of on-line user access.

The GSFC Version 0 DAAC uses several avenues to distribute data. There are two types of distribution orders: random orders and standing orders. The standing orders are requests by users for some or all of the data as it is being received at the DAAC. The

random orders are interactive requests for data that has been previously archived and is available for order.

Log Files and Databases

To identify some of the characteristics of the requests, we examine the GSFC DAAC Oracle data base, which contains information on the files and orders. The requests studied are only for external users of the GSFC DAAC and therefore do not include the test requests processed in 1995. The time of the requests used in this study corresponds to the time when the orders were submitted to the GSFC DAAC and may not be correlated with the time for processing and shipping. To analyze some of the system load, we examine the Unitree log files.

Aggregated User Analysis

In this section, we analyze user requests by aggregating requests according to an interesting classification. In particular we are trying to identify some patterns in the orders in terms of volume, number of files per request, products, data level, interfaces selected, and geographical locations of users.

In Figure 1, we aggregate user requests by month and data product. A *data product* is one of the 7 categories: ACRIM, DAO, PAL, CZCS, TOMS, TOVS, and UARS. A given data product can have multiple *data sets* (e.g. one per data level). The analysis shows that most data volume is for one or two data products (DAO and Pathfinder AVHRR Land (PAL) data). Figure 1 also shows that the volume ordered varies greatly between months. The result is consistent with observations of NDADS. Different data products have different average file sizes, so reporting volume alone tends to bias our results towards favoring data products with large files. For a comparison, we plot the number of files ordered each month by data product, in Figure 2.

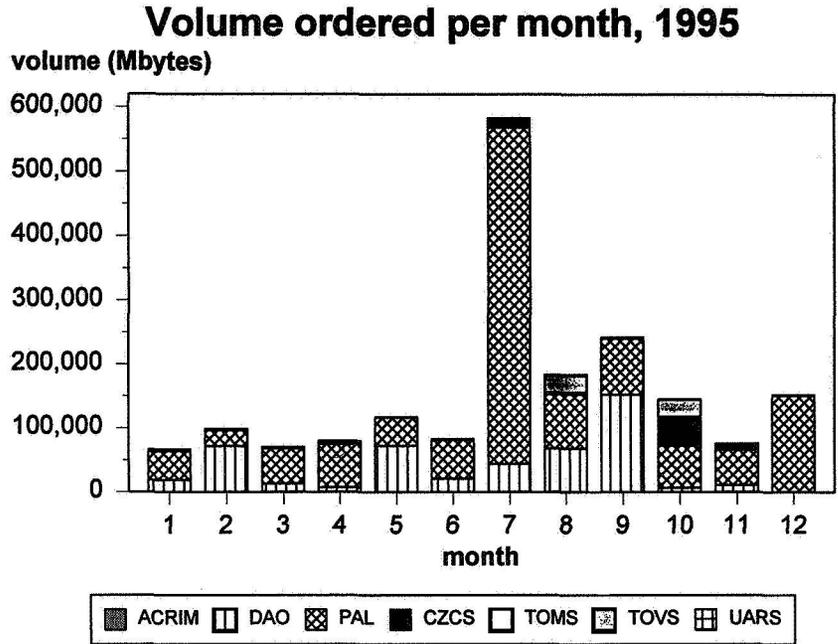


Figure 1. User volume ordered aggregated by data product.

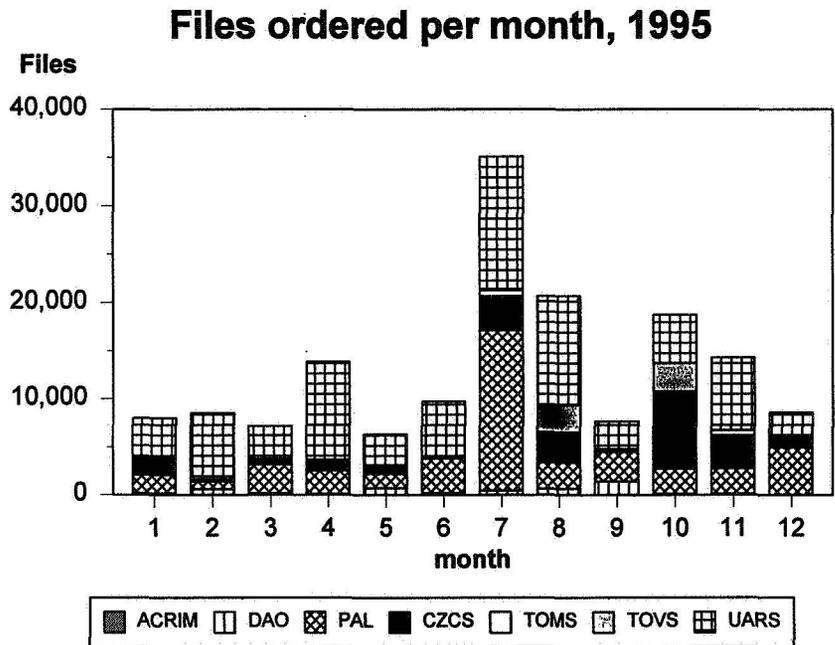


Figure 2. Files ordered aggregated by data product.

Next, we analyze requests according to data level (e.g. Level 1-4). The data that is ingested into the archive is in a variety of higher level data sets. Level 1 data is satellite data with corrections, while higher level data is binned and aggregated. Because of the processing, there is more volume in the lower level products (L1-2) than in the higher level products (L3-4). Figure 3 aggregates user volume by month and data level. As

expected most of the volume of requested data is for Level 3 data, and there is a substantial interest in Level 4 data.

Figure 4 aggregates volume ordered by month and by interface method. A substantial percentage of the total volume requested was from the Character User Interface (ChUI). The volume of standing orders is also important. Files that belong to standing orders are transferred to a distribution staging area soon after being ingested. This method of distribution reduces the load imposed on Unitree because the standing orders files are processed before the data is migrated to the near-line devices.

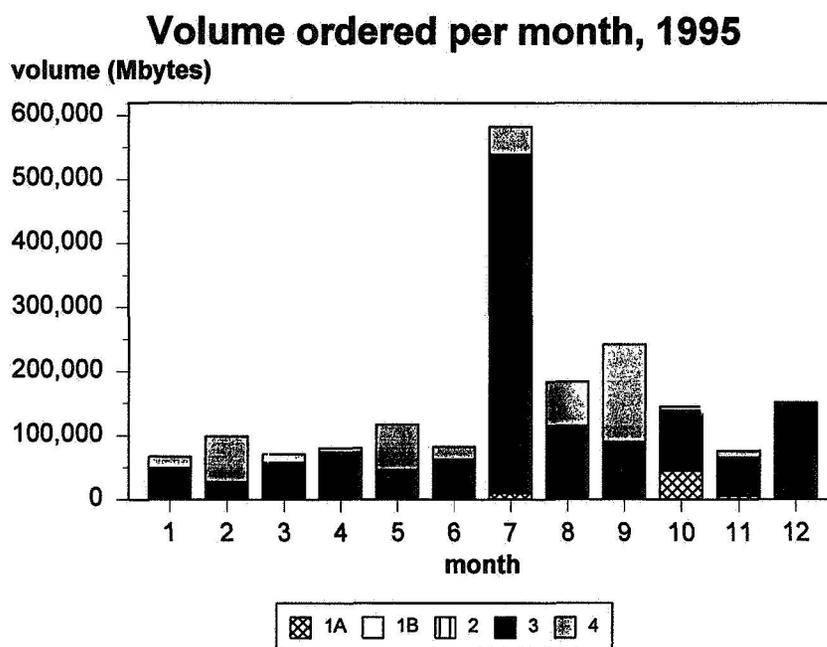


Figure 3. User volume ordered aggregated by data level.

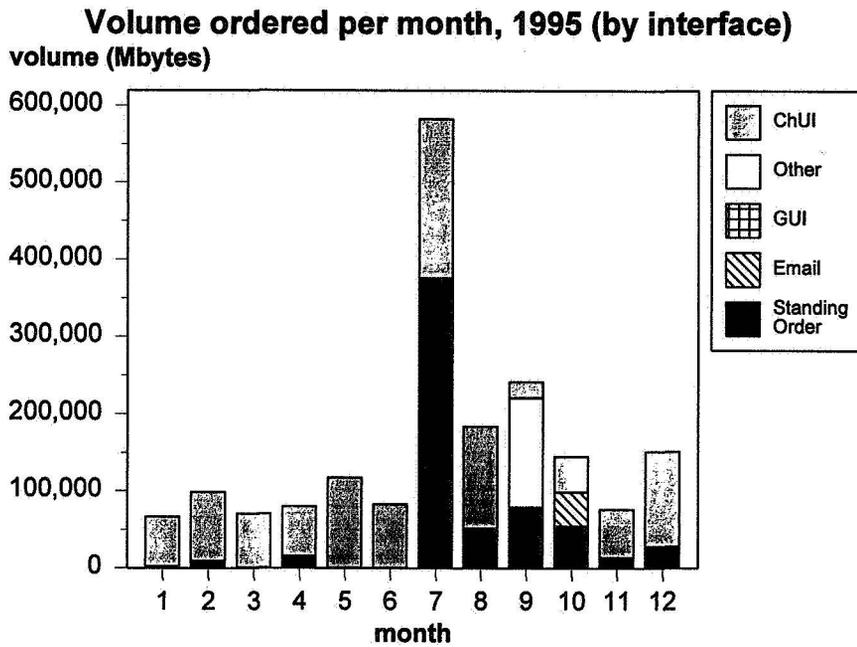


Figure 4. User volume ordered aggregated by interface and month.

In Figure 5 and Figure 6, we aggregate the volume ordered by interface and by hour of the day and day of the week, respectively. The volume ordered shows the typical pattern -- most requests are made during normal working hours. These results are consistent with observations of NDADS, although more requests are made to NDADS during weekends, and few requests are made to NDADS during early morning hours.

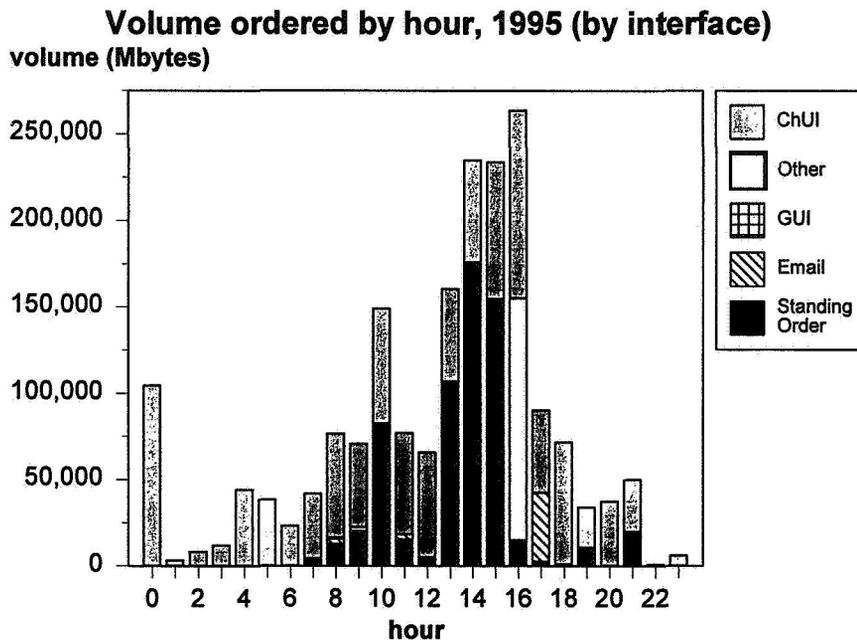


Figure 5. User volume ordered aggregated by interface and hour.

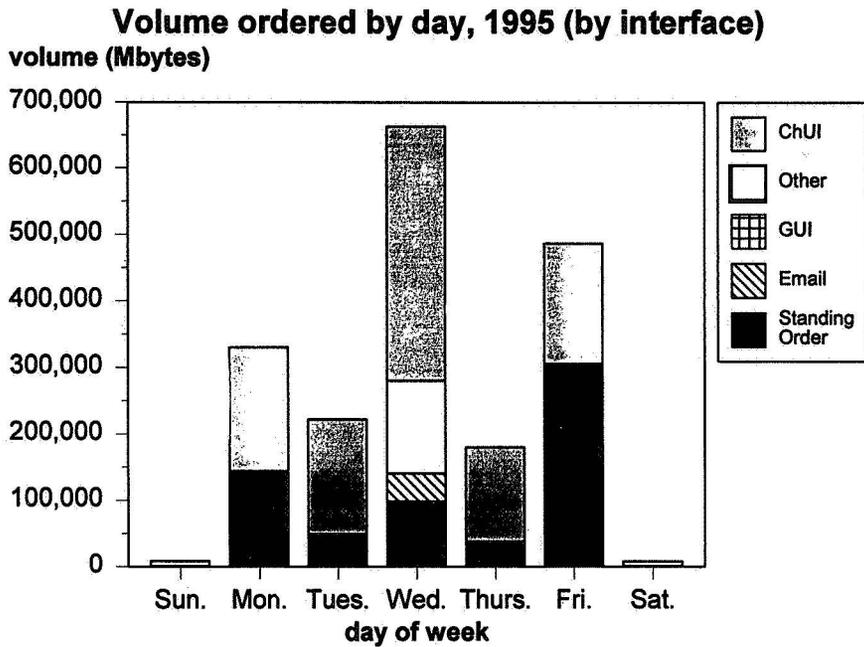


Figure 6. User volume ordered aggregated by interface and hour of the day.

The requested data can be distributed either electronically (via ftp) or on one of several different tape media. Figure 7 aggregates user requests by distribution media and by week of the year. Most data is distributed by creating 4-MM or 8-MM tapes. There is almost no request for 9-track tapes. The volume of ftp requests accounts for only a small portion of the data distribution. We should point out that, due to resource constraints (network and disk space), a limit has been placed on the volume of data that can be distributed via ftp for a given request.

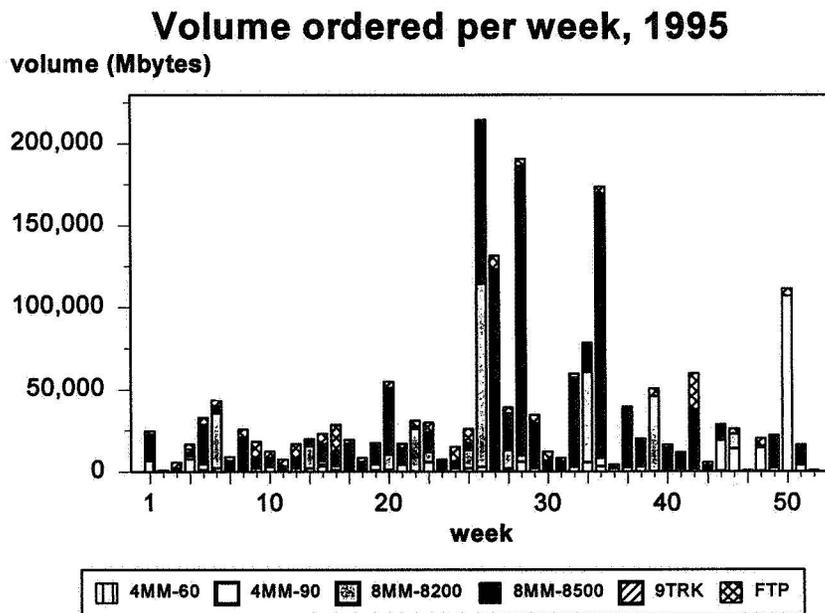


Figure 7. User volume ordered aggregated by media of distribution.

Finally, we plot the volume of data requested by different regions of the world in Figure 8. While most of the request volume came from North America, there is significant world-wide use of the DAAC.

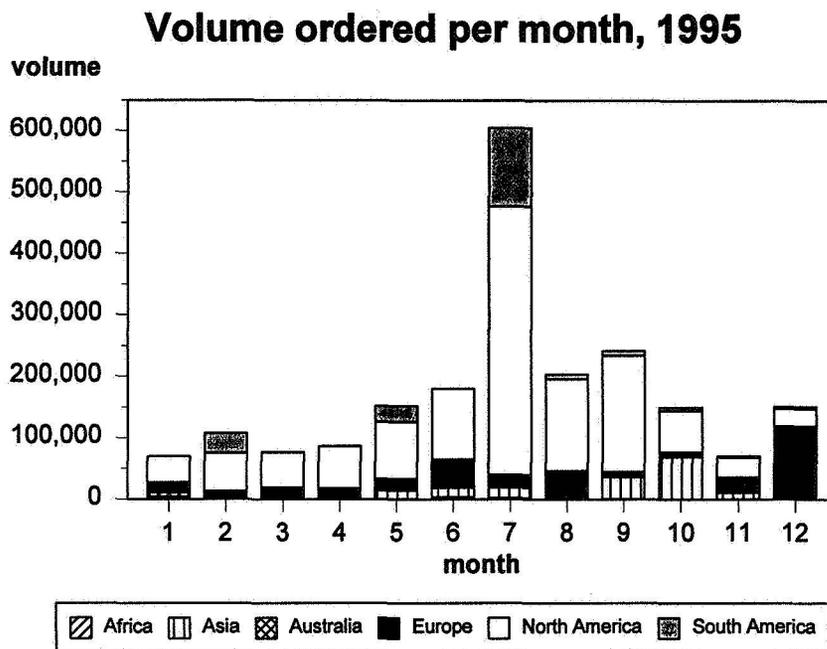


Figure 8. User volume ordered aggregated by region.

User Analysis

In this section, we analyze request size and user activity. The database records for every file accessed; the file size, the data set that the file is a member of (a refinement of data product), and a unique request ID. From this information, we can reconstruct the size of a request. In Figure 9, we plot the volume per request, in Figure 10, we plot the files per request. Figure 9 and Figure 10 are plotted on a log scale because of the very large range of request sizes. The wide range of request sizes suggests that request servicing should be aware of the size of the request and handle it accordingly.

Data in a data product can be divided into *data sets*, based on the type of the data (i.e., different levels, different sensors, etc.). In Figure 11, we plot the number of data sets requested per order. The number of files per request and the number of data sets per request are not correlated, as is shown in Figure 12. The average request in 1995 accessed 78.6 files and 1.6 data sets. These results show that most requests are clustered by the data set, with the implication that archive media should store files of a single data set. These results are consistent with our study of the NDADS archive.

Volume per unique request id, 1995 (sorted)
volume (Mbytes)

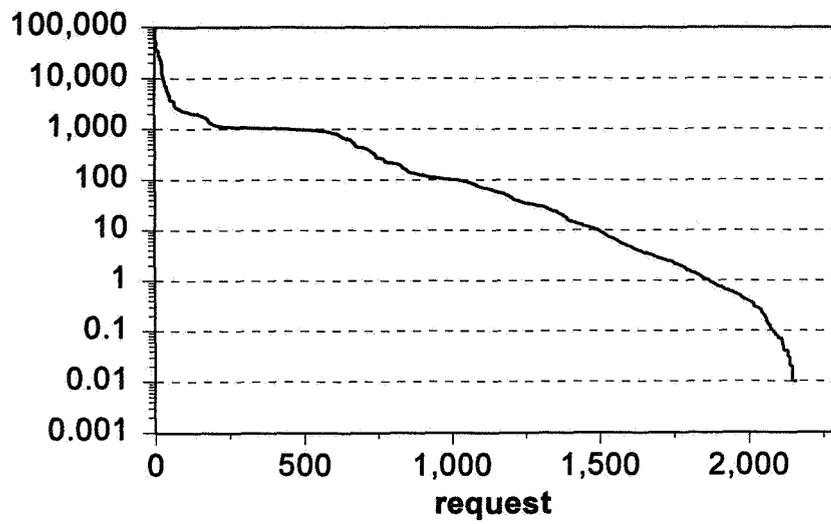


Figure 9. Volume per unique request id.

Files per unique request id, 1995 (sorted)
files

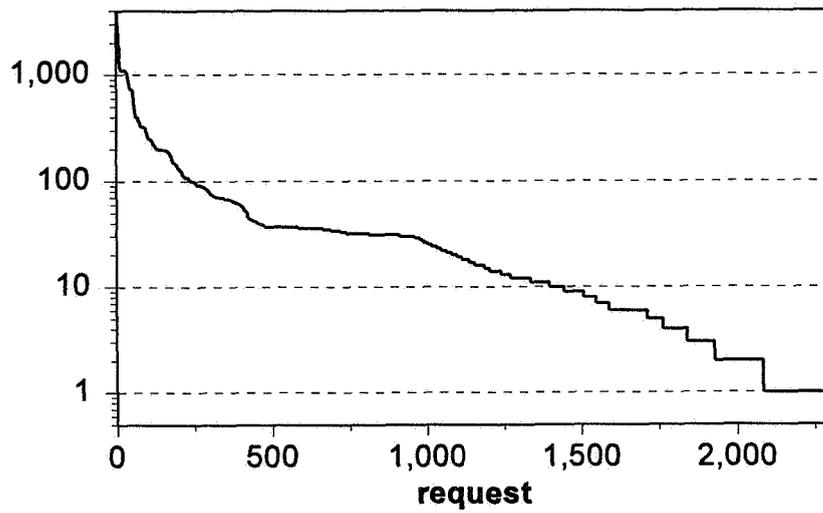


Figure 10. Files per unique request id.

Data sets per unique request id, 1995 (sorted)
data sets

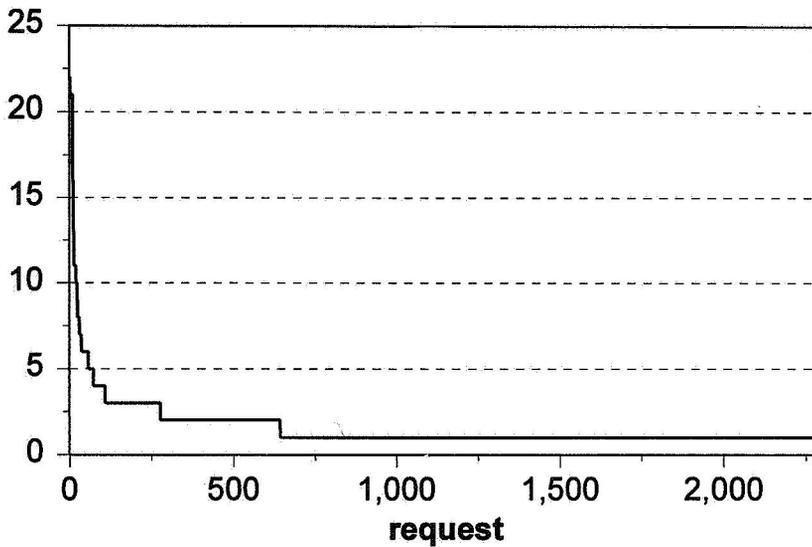


Figure 11. Data sets ordered per unique request id.

Files vs. data sets per request (1995)

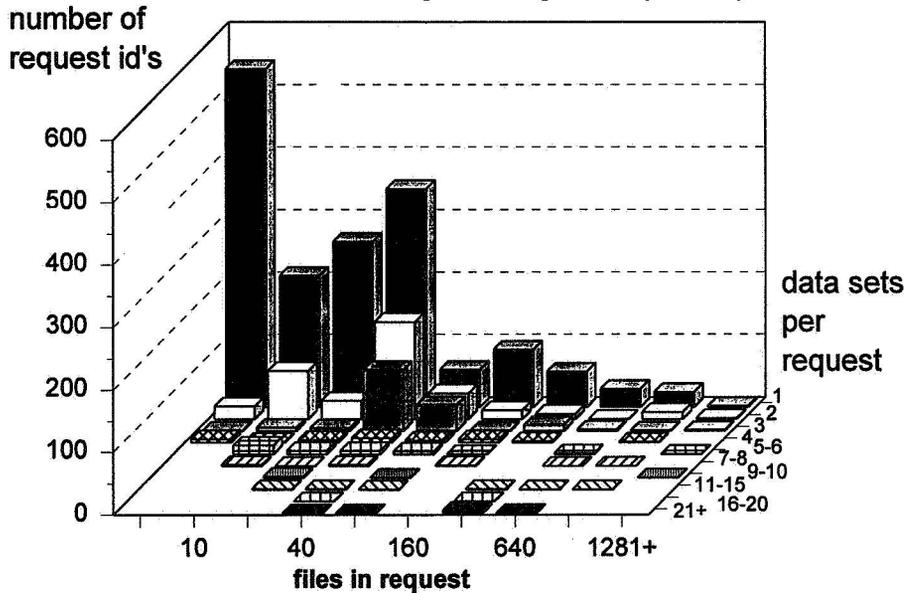


Figure 12. Files vs. data sets in a request.

We can also aggregate requested files based on the user ID. In Figure 13, we plot the number of files requested per unique user, in sorted order, and in Figure 14, we plot the volume requested per unique user in 1995. The curve is non-linear even when plotted on a logarithmic chart. We found that the top 20 users (of 442) requested 47% of all files and 70% of the data volume. We note that top 20 users, rated by files requested, is not the same as the top 20 users rated by volume requested. However, the correlation between

files requested and volume requested is strong (the intersection between the two top-20 sets contains 12 members). A scatter plot of volume requested vs. files requested is shown in Figure 15. These results are consistent with our study of the NDADS archive.

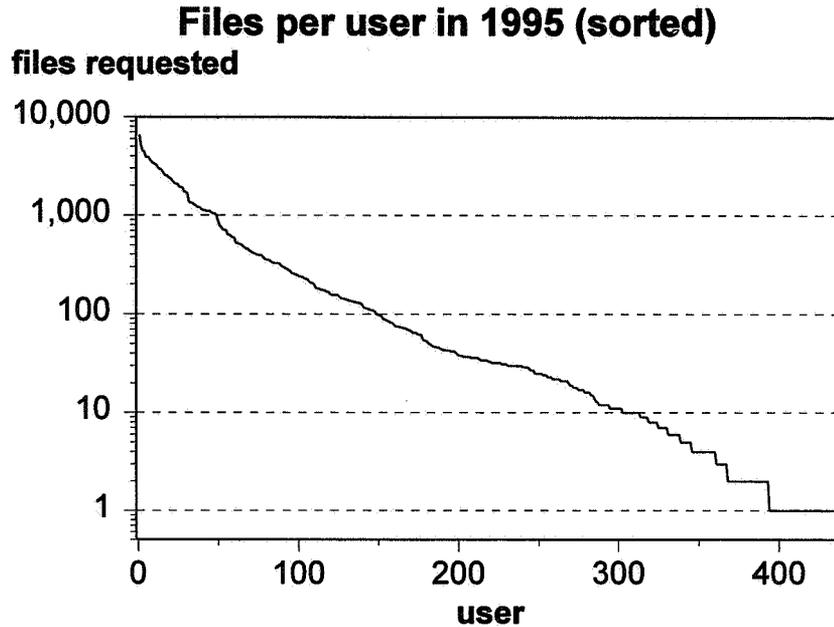


Figure 13. Files per unique user.

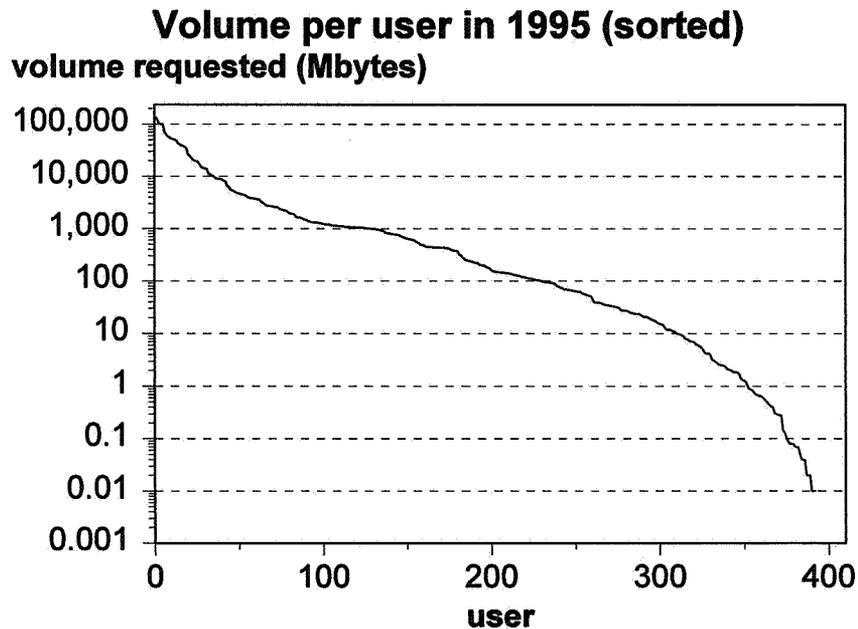


Figure 14. Volume per unique user.

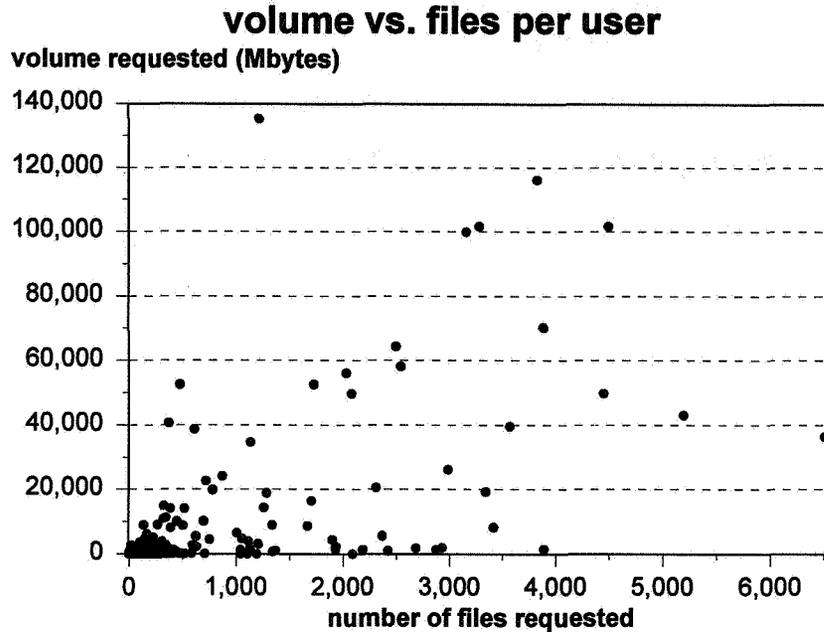


Figure 15. Scatter plot of volume ordered and files ordered per user.

Caching

When a user requests a file, the file is first searched in the on-line disk space. If the file is not located on the cache it is then fetched from tertiary storage into secondary storage and made available to the requester. The file typically has a minimum residency requirement to give the requester time to access the file. While the file is disk-resident, a second request for the file can be satisfied without fetching the file from tertiary storage. These cache hits can reduce the load on the tertiary storage system, and also improve response times. Fetching a file from the tertiary storage requires a tape to be picked by a robotic device, mounted, the file searched on the tape and then read. All this can take minutes before the file is ready to be read.

The archive systems should have enough disk storage to satisfy the minimum residency requirement. However, files referenced within the minimum residency may be deleted if the cache runs out of space. In this case, using a FIFO algorithm the oldest files are deleted first. The buffer might run out of disk space necessary to satisfy minimum residency due to a high request load, or due to a high *ingest* load (i.e., the ingested files must be stored on-line until they can be migrated to tertiary storage). Although the ingest load can interfere with the cache, we do not consider it in this study. However, the relative performance of the algorithms will be the same with or without the ingest load.

If the cached files are large (an average size of 12.8 Mbytes in this study), then the time to transfer referenced files not in the cache can be very long. We compute the *cost* of servicing a reference string to be the weighted sum of the number of cache misses and the number of bytes transferred. In these studies, we assumed that transferring 10 Mbytes is equal to the cost of a cache miss (The time to load a media is much larger than the transfer time, but this cost is amortized over all files loaded from the media). Let $cost_f$ be

the cost of transferring file f from the archive to on-line storage, and let S_f be the size of file f . Then, the (normalized) value of $cost_f$ is:

$$cost_f = 1 + S_f / 10 \text{ Mbytes}$$

We can evaluate the benefit of using a cache by looking at the *hit rate* of the cache, or by looking at the *cost reduction* of the cache. The cost reduction is the reduction of load on tertiary storage caused by the cache. More formally, let the *reference string* (i.e., the sequence of requests that pass through the cache) be $r=(f_1, f_2, \dots, f_m)$. Let $Miss(f_i)$ have the value 1 if f_i was not in the cache when it was referenced, and 0 if it was in the cache. Then, the cost of processing the reference string when using a cache, $cost(cache)$ is:

$$cost(cache) = \sum_{f_i \in r} cost_{f_i} * Miss(f_i)$$

Let $cost(tot)$ be the cost of servicing r when no cache is used (i.e., $Miss(f_i)=1$ for every f_i). Then the *fraction of cost saved* by using the cache is:

$$\text{fraction of cost saved} = 1 - cost(cache)/cost(tot)$$

A large body of caching literature exists when all cached objects are of the same size. The Least Recently Used (LRU) replacement algorithm is widely recognized as having good performance in practice. Caching objects of widely varying sizes is somewhat more complicated. If one wants to minimize the number of cache misses, then it is much better to choose large files than small files for replacement, because removing large files frees up more space. Let the set of files in the cache be F . The general scheme is to assign to each file f in F a *weight*, $weight_f$, and choose for replacement the file with the largest weight. Note that many files might need to be replaced on a cache miss.

The optimal replacement algorithm for variable size objects, with respect to cache misses, is the GOPT algorithm [DS78]: For file $f \in F$, let N_f be the time until the next reference to f and let S_f be the size of f . Set $weight_f = N_f * S_f$, and choose for replacement the file f in F such that $weight_f$ is the largest.

The GOPT algorithm cannot be implemented (because it requires knowledge of future events), but it can be approximated. The Space-Time Working Set (STWS) algorithm [Sm81c] approximates GOPT by substituting P_f the time since the last reference to f , for N_f .

While STWS can be implemented, it also requires a great deal of computation. For this reason, STWS is often approximated by what we call the STbin algorithm [Mi94]: A file is put into a bin based on its size. The files in a bin are sorted in a list using LRU. To choose a file for replacement, look at the file at the tail of each bin and compute its weight to be $P_f * S_f$. Choose for replacement the file with the largest weight.

The STbin algorithm does not account for the cost of transferring files, and may discriminate too strongly against large files. We examined two algorithms that modify the weight function to account for transfer costs. Let $cost_f$ be cost incurred if file f must be loaded. The *Costbin* algorithm computes the weight of file f to be $P_f * S_f / cost_f$, where $cost_f$ is defined above. Alternatively, we can use a non-linear function. The *Alphabin*

algorithm computes the weight of file f to be $P_f * S_f^\alpha$, where α is a real number. Note that setting $\alpha=0$ gives LRU and setting $\alpha=1$ gives STbin.

Another method for incorporating size and last-reference time into victim selection is to take a weighted sum. Let K_s be the *size factor* and let K_t be the *time factor*. Then, choose for replacement the file with the smallest weight $K_s * S_f + K_t * P_f$. Since the file weight is computed by a sum, we call the algorithm the *SUM* algorithm. It is used by several HSM products.

Recent work in caching algorithms has produced *statistical caching* algorithms [OOW93]. The LRU/2 algorithm chooses for replacement the object whose penultimate reference (instead of most recent reference) is the furthest in the past. We adapt LRU/2 to file caching by maintaining the bins in the ST-bin algorithm by LRU/2 instead of LRU. We call the new algorithm *LRU/2-bin*.

HSM systems typically use a "watermark" technique to manage their staging disk. When the staging disk space utilization exceeds a high watermark, files in the staging area are migrated into tertiary storage until the staging disk utilization reaches a low watermark. The motivation for the watermark technique is to write back dirty files in a single burst, thus improving efficiency by exploiting write locality. The archive that we study contains read-only files, so the watermarks should be set as high as possible for maximum efficiency.

The minimum residence period is implemented by partitioning the cache into the regular cache and the *minimum residence* cache. When a file is referenced, it is placed in the minimum residency cache, where it remains until the minimum residence period has passed. After the minimum residence period, the file is placed in the regular cache. Normally, files in the minimum residence cache are not selected for replacement. However, if the minimum residence cache size exceeds the total cache size, the oldest files in the minimum period cache are chosen for replacement.

In our caching analysis, we assume a disk block size of 1024 bytes, and set a limit on the number of disk blocks that are available for caching. We trigger replacement when fetching a new file will cause the space limit to be exceeded, and we remove files until the space limit will not be exceeded. For the STbin and LRU/2-bin algorithms, bin i holds files that use between 2^i and $2^{i+1}-1$ blocks. We set the minimum residency period to 1 day. We retrieved from the database a listing of all files requested in 1995², and sorted the list of time of reference to create the reference string for our cache simulators. We report both the hit rate and the reduction in cost due to running a caching algorithm with a particular cache size.

We first test the STbin variants. In Figure 16, we plot the cost reduction as we vary α for different cache sizes. The best setting of α is approximately 1/2. However, the improvement over STbin is not large. Next, we compare Costbin against STbin in Figure 17. While Costbin has better performance than STbin, the difference is not large.

² Subject to the restrictions listed in Section 1.2

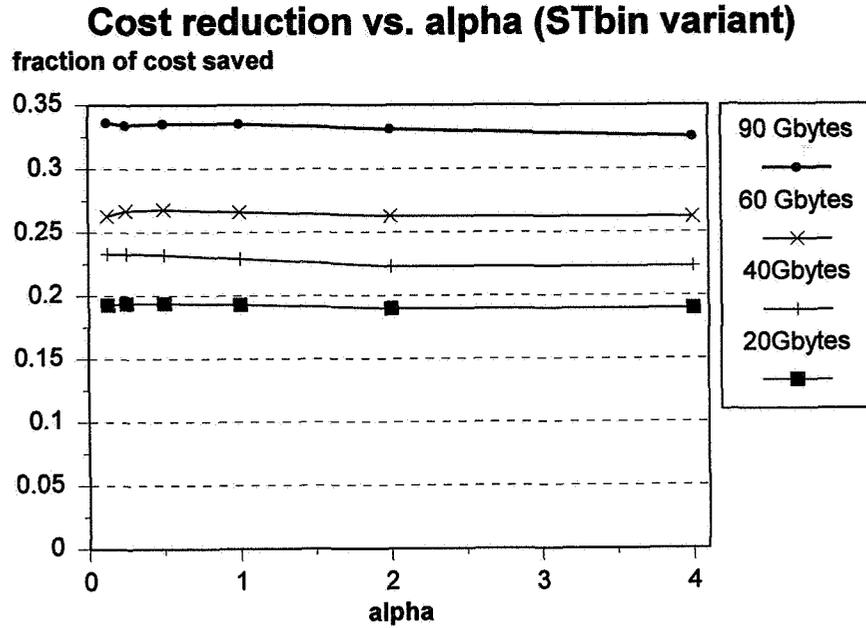


Figure 16. Finding the best value of α for alphabin.

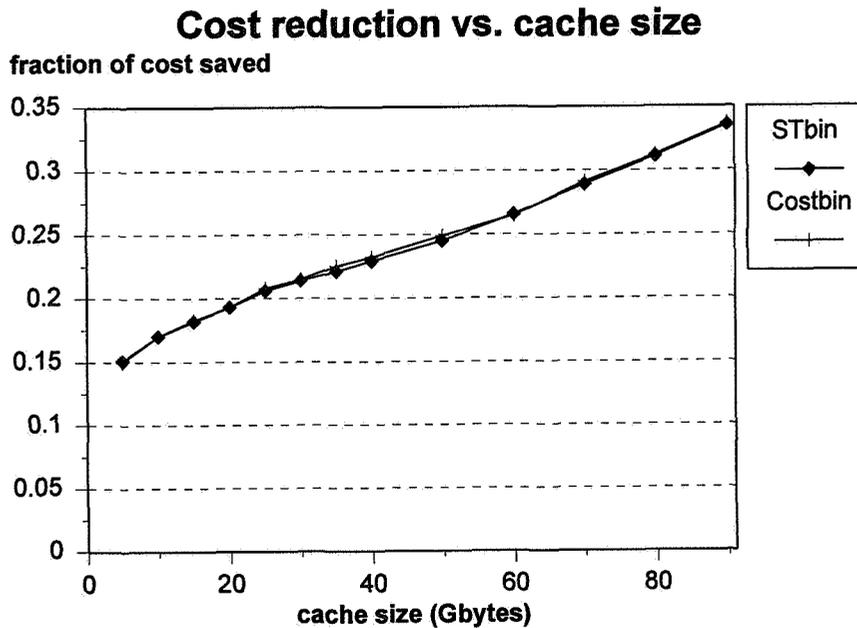


Figure 17. Comparison of Costbin to STbin.

In Figure 18 we plot the fraction of cost saved for the LRU, LRU/2, SUM, and STbin algorithms as we increase the cache size from 5 Gbytes to 60 Gbytes, and in Figure 19 we plot the hit rate. The results show that the STbin and the LRU/2-bin algorithms are significantly better than LRU, and somewhat better than the SUM algorithm. The LRU/2-bin algorithm had somewhat better performance than the STbin

algorithm. We note that STbin requires less CPU time for execution than either the LRU/2 or the SUM algorithm, and the SUM algorithm requires careful tuning.

The results show that caching can be effective in reducing the load on the tertiary storage device, in spite of the highly random nature of requests to an on-line archive. The GSFC Version 0 DAAC currently uses a 60 Gbyte distribution cache. Simulation results indicate that this size cache can provide a hit rate and cost savings of about 25%. The Unitree logs indicate that the internal Unitree cache (32 Gbyte) had an additional hit rate of 15.6%.

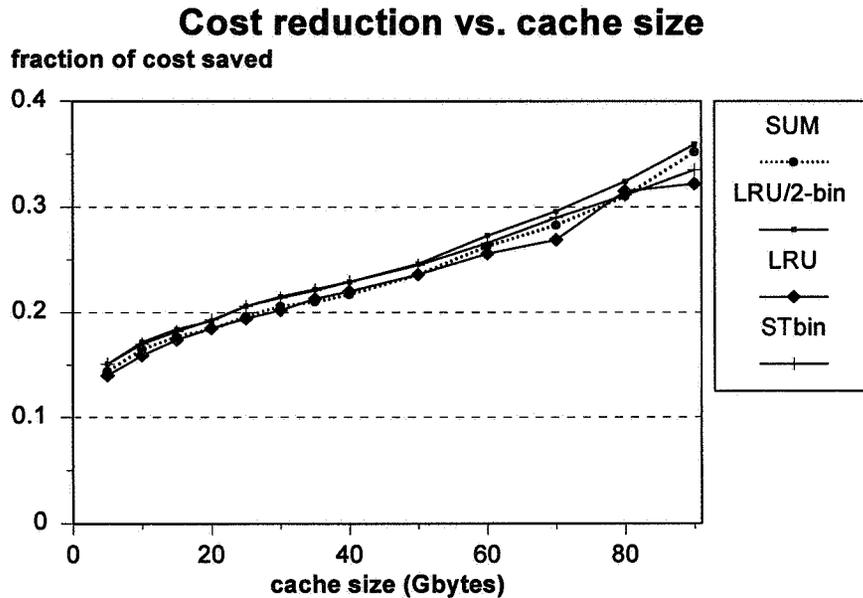


Figure 18. Cache algorithm comparison (cost reduction).

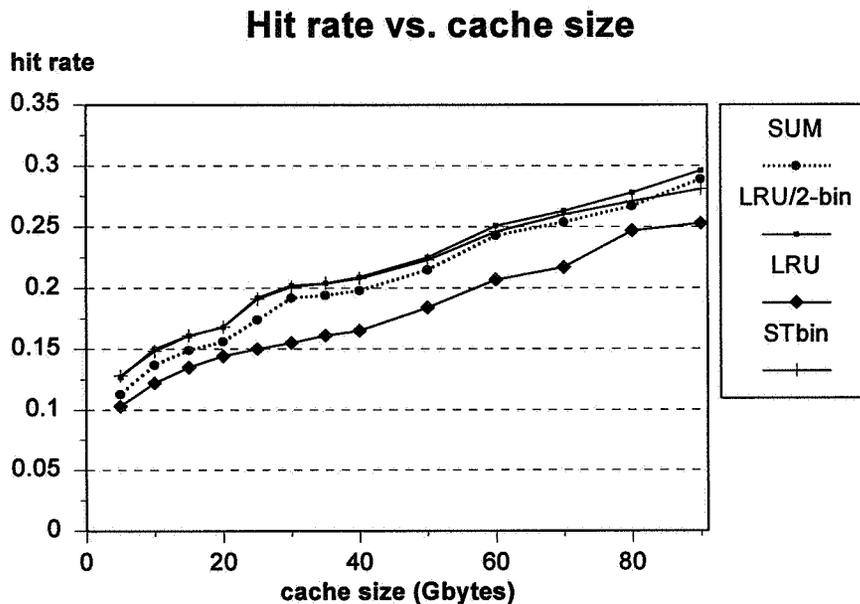


Figure 19. Cache algorithm comparison (hit rate).

We ran another experiment to determine the effect of changing the minimum residence period. Figure 20 shows the cost reduction of the STbin algorithm as the minimum residence period is varied from 10 minutes to 2 days. We varied the cache size between 10 and 60 Gbytes. Increasing the minimum residence time decreases the cost reduction, but the effect is small.

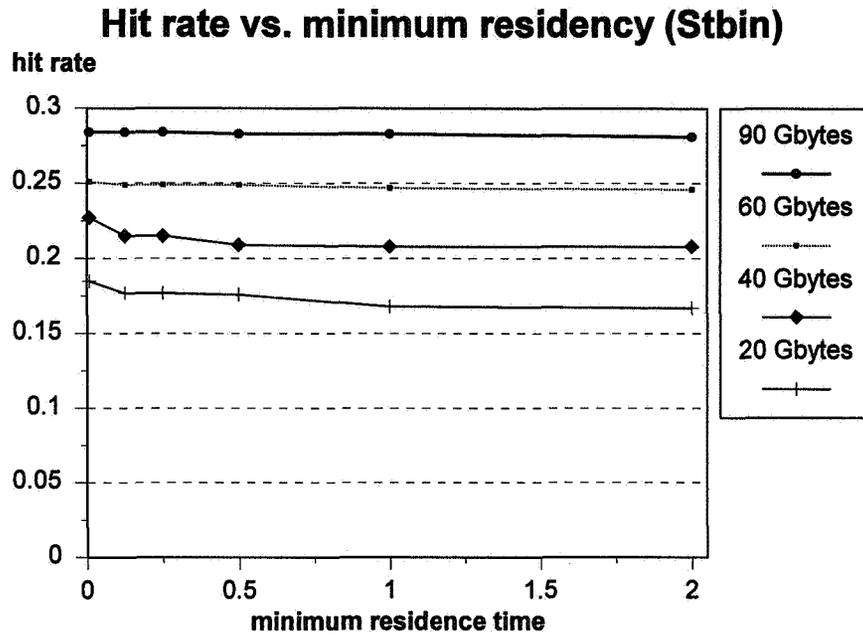


Figure 20. Effect of changing the minimum residence time.

File Access Pattern Analysis

The success of caching depends upon the file access patterns. In this section we examine some aspects of the access patterns. These results also have implications for archive design.

Most files are accessed only a few times, limiting the maximum cache hit rate. Figure 21 plots distribution of the number of times a file was referenced in 1995. The fact that so most files are referenced once limits the performance of statistical caching algorithms, such as LRU/2-bin. We note further that only 12% of the 550,000 files in the archive were requested during 1995. This result is consistent with observations of the NDADS archive.

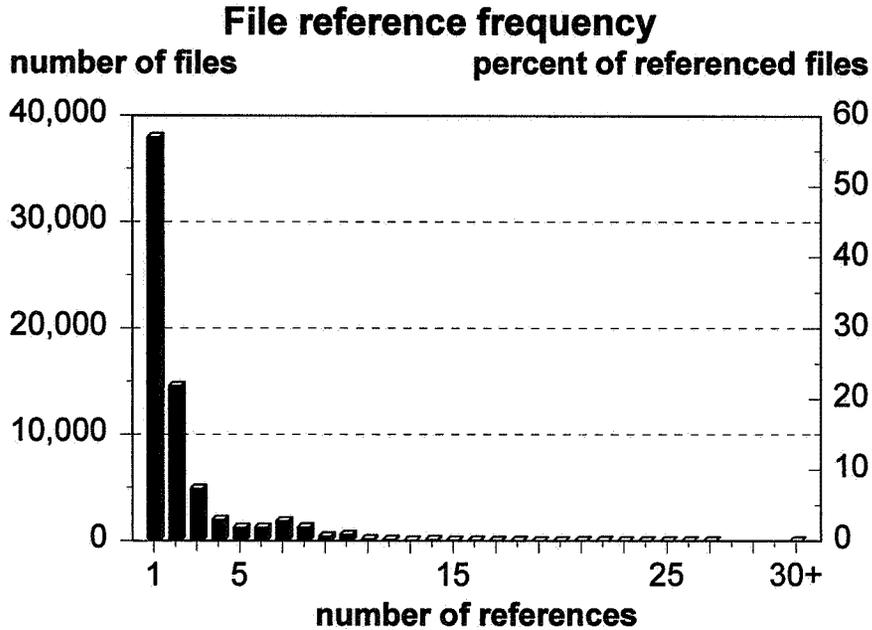


Figure 21. Distribution of the number of references to a file in 1995.

The effectiveness of caching also depends upon the average time between references to a file (the *inter-reference* time). In Figure 22 we plot the distribution of inter-reference times during 1995. To generate this plot, we scanned through all file accesses and searched for repeat accesses. Whenever a repeated reference was found, we incremented a histogram based on the number of days since the last reference. The plot shows that most repeat references occur shortly after an initial access, but that the inter-reference time distribution has a long tail. The rise at the end of the tail represents all repeat references with an inter-reference time of 186 days or larger. The average number of days between an access to a file, given that the file is accessed at least twice in 1995 is 46.1 days. This result is essentially consistent with observations of the NDADS archive, which has an average interreference time of 27.6 days. Both archives show a peak in the inter-reference time near 0 days, and at 1 and 2 months after the previous reference. However, these characteristics are stronger in the NDADS references. For both archives, the inter-reference time distribution has a long tail (i.e., represented by the point at ``185+").

We found that many of the repeat references are due to the same user requesting a file for a second time. This is shown in Figure 23, which plots the fraction of repeat requests that are due to the same user, by time since last request. In total, 15.4% of the repeat references in 1995 are due to the same user as had submitted the previous reference. By contrast, 57% of the repeat references to NDADS are due to the same user. One explanation for this difference is that most requests to the GSFC Version 0 DAAC are submitted interactively, while most requests to NDADS are submitted by email. Network and mailer delays compound archive delays to cause the user to suspect that the request has been lost.

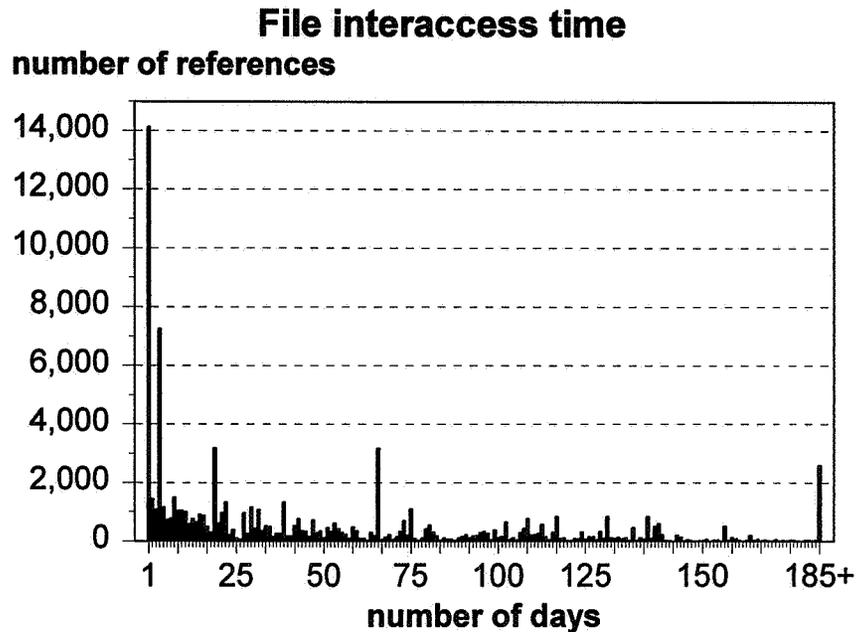


Figure 22. Distribution of file inter-reference times. The point at ``185+'' represents the tail of the distribution.

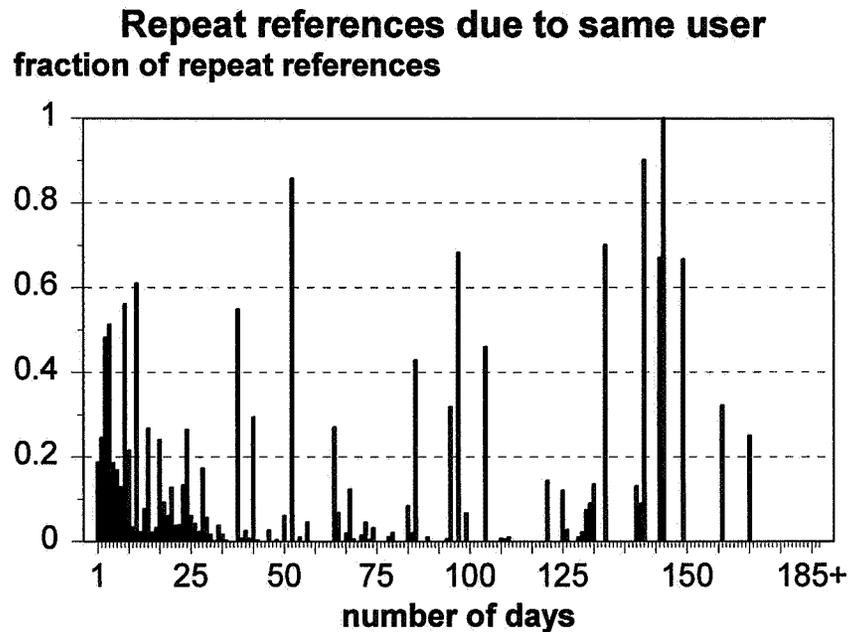


Figure 23. Fraction of repeat references in which both the current and the previous reference are submitted by the same user (binned on inter-reference time).

The performance of the STbin algorithm and its variants (Alphabin and Costbin) depends on the distribution of file sizes. In Figure 24, we plot the file references binned on the file sizes. Large files account for a large fraction of the accesses (the average size of a requested file is 12.8 Mbytes). Caching large files can be effective if caching large files is likely to result in a cache hit. In Figure 25, we plot the proportion of file references that

are to files previously referenced, binned by file size. We also plot the average interaccess time. Large files have high reaccess rates, and for this reason the STbin variants improve hit rates as well as cost reduction.

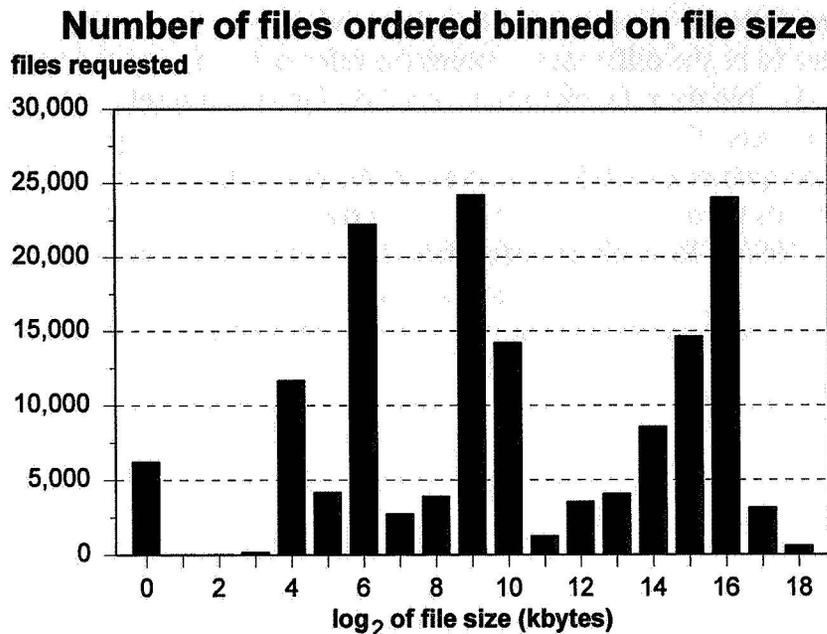


Figure 24. Ordered files binned on file size.

Repeat reference and interaccess time, by file si

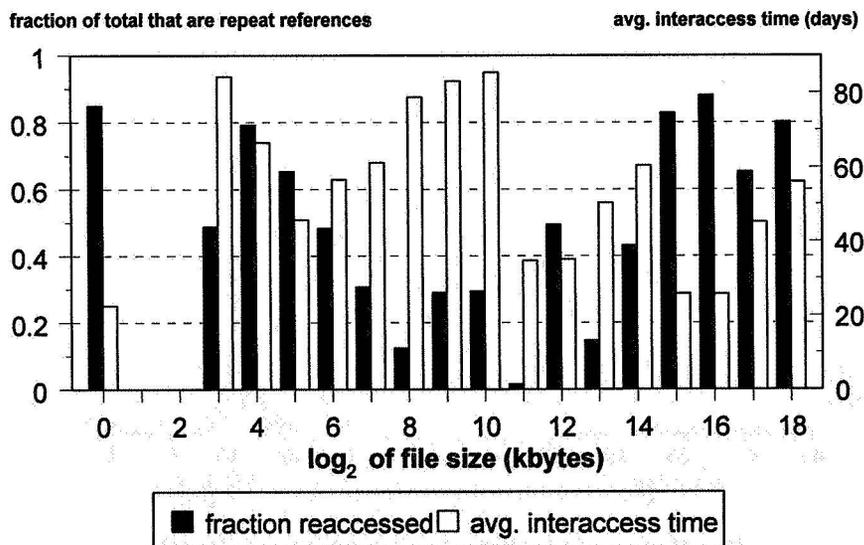


Figure 25. Repeat requests and interaccess times binned on file size.

The result that the average size of a requested file is 12.8 Mbytes was unexpected, because the average size of a file in the archive is 1 Mbyte. The bias towards ordering large files is due in part to the anonymous ftp archive, which serves small files. Users prefer small files, but there is a high volume of data ordered for the DAO and PAL data

products, both of which are stored in large files (an average of 16.5 Mbytes and 55.0 Mbytes, respectively).

In Figure 26, we plot the "age" of the files that are referenced (requested). We compute this distribution as follows. For every file referenced in the observation period, we compute the *file age* to be the difference between the reference time and the time that the file was archived. We bin the referenced files based on file age in weeks. Because there is a dependence between the time of reference and the file age, we plot the file age distribution for each quarter of 1995. The peak in the age of the referenced files in all four charts corresponds to roughly the same archiving dates (we note that many files were rearchived in early 1995). Recently ingested data does not show an unusually high user interest. One explanation for this result is that new data is not immediately known to most users, and it is only after some advertisement (newsletter, conference, word of mouth) that the data may be more frequently requested. This result is consistent with observations of the NDADS archive.

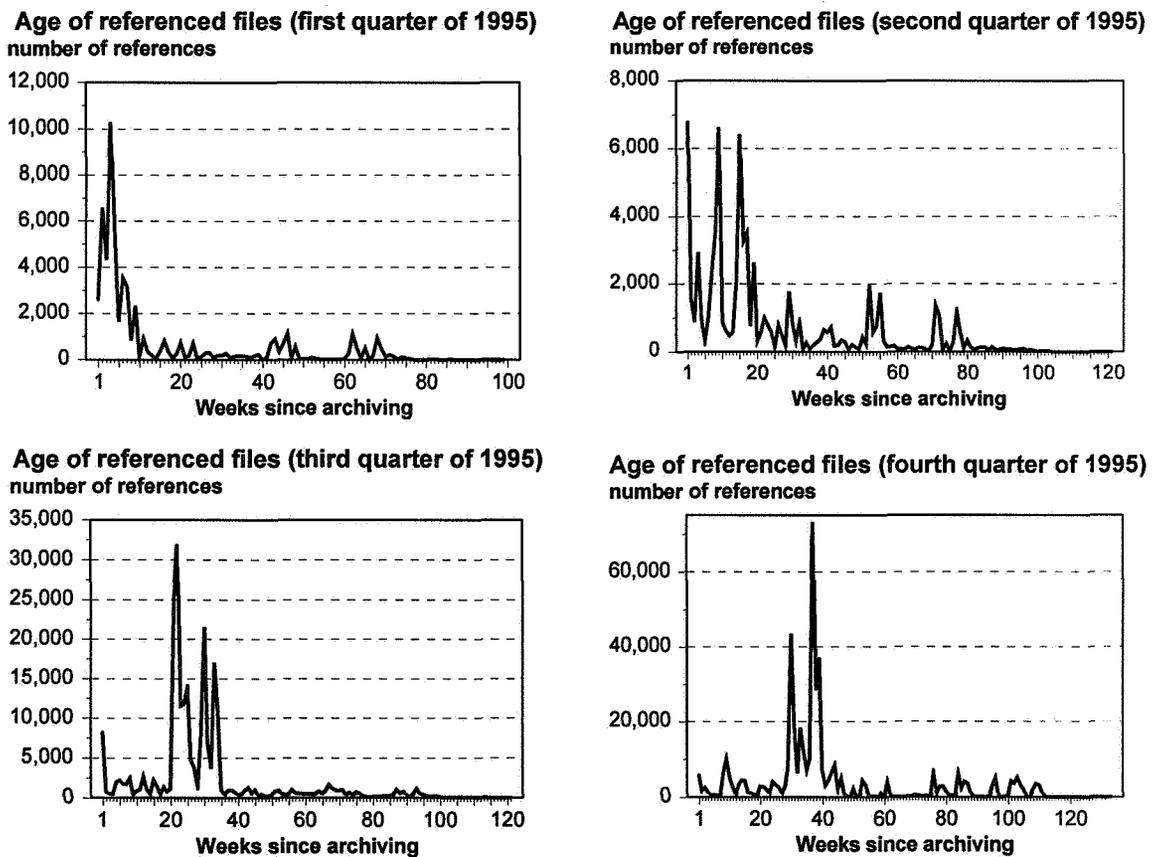


Figure 26. Distribution of time between file archive and file reference.

Internal Activity

We conclude with some observations of the work performed by the archive. In Figure 27, we plot the number of Unitree media mounts per week, and the average number of files transferred per mount (we obtained this data from the Unitree logs). The average

number of files transferred per mount for all of 1995 is 8.6. Unitree does not distinguish between mounts to read or write data. Consequently, the average number of files transferred includes both read and write operations. Migrations are executed periodically (e.g., one hour) to increase the chance of writing multiple files to the same media. Stage operations are performed as soon as the resources are available (e.g., tape drives). It would have been interesting to derive the average number of files retrieved for stage operations only, and to correlate the stages with the requests. This could have provided some insight on how well the stage operations are scheduled and how clustered the files requested are on tapes.

The reader might note that this chart does not correlate well with the results presented in Figure 1 through Figure 8. There are two reasons for this discrepancy. First, the data in Figure 1 through Figure 8 is based on time an order was requested, not time the order was processed or distributed. Request processing might be delayed due to heavy loads, or to handle very large requests (e.g., see Figure 9).

As described in the introduction, we have limited this study to only those orders that are filled by "pulling" data from the mass storage system Unitree. However, the entire activity for the DAAC is significantly higher (see Figure 28) because of the other distribution methods used at GSFC DAAC that are not included in this study (e.g. CD-ROM, anonymous ftp, off-line requests). It is interesting to note that the distribution volume is much greater than the ingest volume.

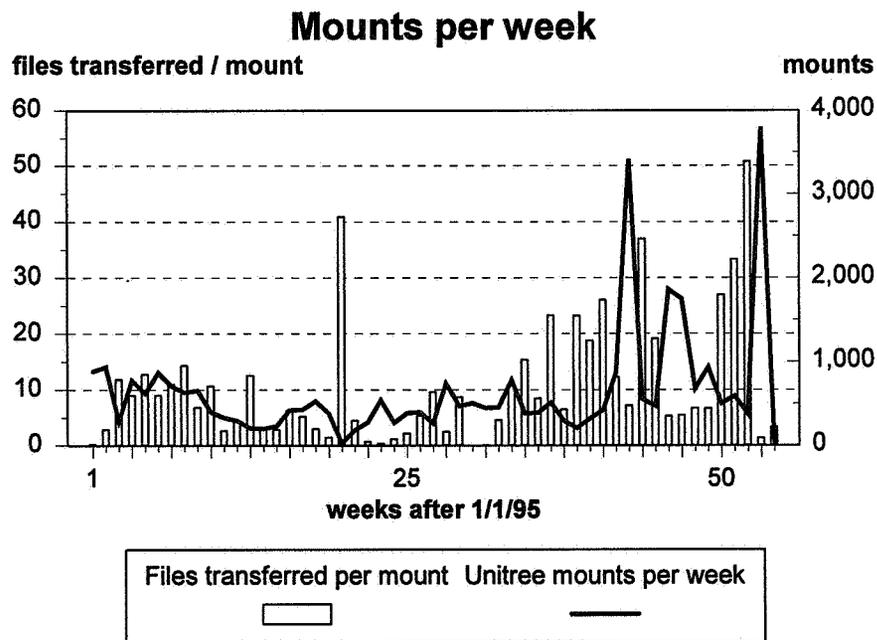


Figure 27. Unitree mounts per week and files per mount.

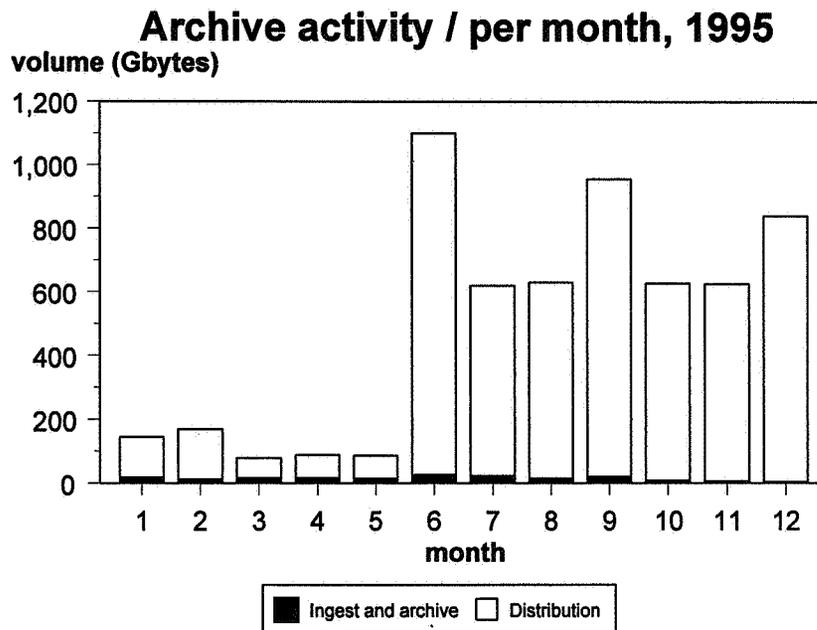


Figure 28. Total archive activity, per month.

Conclusions

We have presented a study of the external requests made to the GSFC Version 0 Distributed Active Archive Center. The analysis examined only a subset of all requests submitted. In particular orders for CD-ROMs, off-line requests and anonymous ftp are excluded because they did not affect the performance of Unitree and the near-line devices. A summary of the results are:

- Most of the volume of the data ordered is concentrated on two of the seven data products, and on higher level data.
- Most of user requests (by volume) were submitted via the Character-based User Interface (ChUI).
- Most of the volume of data is distributed via tape.
- The requested volume varies greatly between months. Most of that volume is submitted during normal working hours.
- There is a wide range of request sizes, and some requests are very large (100+ Gbytes).
- Most requests require service from a small number of data sets.
- A small set of hot users account for most of files and volume requested.
- LRU/2-bin is the best file caching algorithm on this workload, STbin also works well.
- The file interreference distribution has a peak at < 1 day, and a long tail.

- Interest in data is not correlated with the time of archiving.

Acknowledgments

We would like to thank Dr. Blanche Meeson for reviewing this paper, and we would like to thank Andy Griffin, Gary Gregorich, Frances Bergmann, Robert Swafford, and Hughes STX for their help with collecting data.

Bibliography

[AN88] E.R. Arnold and M.E. Nelson. Automatic Unix backup in a mass storage environment. In *Usenix - Winter 1988*, pages 131-136, 1988.

[DS78] P.J. Denning and D.R. Sluts. Generalized working sets for segment reference strings. *Communications of the ACM*, 21(9):750-759, 1978.

[EP82] C.W. Ewing and A.M. Peskin. The masstor mass storage product at Brookhaven national laboratory. *Computer*, pages 57-66, 1982.

[ESDIS] Estdis document catalog. http://spsosun.gsfc.nasa.gov/ESDIS_Docs.html.

[HP89] R.L. Henderson and A. Poston. MSS II and RASH: A mainframe unix based mass storage system with a rapid access storage hierarchical file management system. In *USENIX - Winter 1989*, pages 65-84, 1989.

[Jo95] 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, NASA Conference Publication 3295

[JR91] D.W. Jensen and D.A. Reed. File archive activity in a supercomputing environment. Technical Report UIUCDCS-R-91-1672, University of Illinois at Urbana-Champaign, 1991.

[LRB82] D.H. Lawrie, J.M. Randal, and R.R. Barton. Experiments with automatic file migration. *Computer*, pages 45-55, 1982.

[Mi94] E. Miller, 1994. Private communication. Thanks also to comp.arch.storage.

[MK91] E.L. Miller and R.H. Katz. Analyzing the I/O behavior of supercomputing applications. In *Supercomputing '91*, pages 557-577, 1991.

[MK93] E.L. Miller and R.H. Katz. An analysis of file migration in a unix supercomputing environment. In *USENIX - Winter 1988*, 1993.

[OOW93] E.J. O'Neil, P.E. O'Neil, and G. Weikum. The LRU-K page replacement algorithm for database disk buffering. In *Proceedings of the 1993 ACM Sigmod International Conference on Management of Data*, pages 297-306, 1993.

[Sm81d] A.J. Smith. Analysis of long-term reference patterns for application to file migration algorithms. *IEEE Trans. on Software Engineering*, SE-7(4):403-417, 1981.

[Sm81c] A.J. Smith. Long term file migration: Development and evaluation of algorithms. *Communications of the ACM*, 24(8):521-532, 1981.

[Str92] S. Strange. Analysis of long-term unix file access patterns for application to automatic file migration strategies. Technical Report UCB/CSD 92/700, University of California, Berkeley, 1992.

[TS93] A. Tarshish and E. Salmon. The growth of the UniTree mass storage system at the NASA Center for the Computational Sciences. *In Third NASA Goddard Conf. on Mass Storage Systems and Technologies*, pages 179-185, 1993, NASA Conference Publication 3262

[TH88] E. Thanhardt and G. Harano. File migration in the NCAR mass storage system. *In Mass Storage Systems Symposium*, pages 114-121, 1988.

513-82
83195

A Media Maniac's Guide to Removable Mass Storage Media

Linda S. Kempster

IIT Research Institute

4600 Forbes Blvd.

Lanham MD 20706

lkempster@mtc.iitri.com

Phone: 301-918-1037

Fax: 301-731-0253

Background

The electronic imaging world has been changing. For over a dozen years, users and customers have been the target of presentations providing the incentive to get them to buy into the concept of electronic document handling. Initially, these systems incorporated the revolutionary 12-inch WORM (write once, read many) optical disks that were supposed to replace file cabinets and microfilm repositories. In 1984, it was new, it was scary and it was threatening to data processing managers who had long been the keepers of the corporate data. Once the technology behind the optical disk industry became more familiar, systems hit the marketplace with a variety of shapes, sizes, capacities, and characteristics. Choices became confusing and the presentation to management changed from "should we buy" to "what should we buy?"

Just when laws began to back optical for its non-erasability, that very feature was seen as an impediment to progress. Buyers were looking for the erasable features to allow them to get more long-term usage out of their investment in the systems. Instead of replacing file cabinets, they wanted to replace expensive hard drives — and WORM media was not a suitable candidate. To meet a growing market demand, erasable disks were introduced in 1988-89.

Early systems, dating back to 1983-84, cost over \$1,000,000. Potential users were faced with a sticker shock that was tough to argue. There were no components of an imaging systems that did not shock a procurement official. The drives, media, autoloaders, scanners, monitors, printers — everything! Monitors to display images were different from those to display normal data, so high resolution image viewers had to be purchased. Expensive scanners were used to digitize the documents because all previous image capture functions had been performed with cameras in film-based systems. Printers that had been turning out 2-Kilobyte (KB) ASCII reports or word processing files, were now being asked to generate high resolution 50-KB document images. Speed and user contention were serious matters for these departmental workhorses. The first media cost close to \$1,000 for 1,000 megabytes (MB) of capacity. This industry was a tough one to launch!

Meanwhile, the scientific community was developing tapes for customers such as NASA, NOAA, and CIA to record digital data. Some of the first high capacity media to offer a reasonable price to the digital storage industry was the digital VHS tape made by Honeywell. In 1986, when the 12 inch optical boasted a capacity of 2 gigabytes (GBs), the VHS tape could hold 5.2 GBs. The \$35 price for tape compared to the disk's price of \$1000 gave Honeywell an impetus to enter the document imaging market. They had to create a buffer strategy to change from capturing streaming instrumentation data to handling packets of data called images, but once that was established, their breakthrough had an extensive impact on other vendors offering tape solutions. These vendors included Exabyte, Ampex, Sony, Storage Technology (STK) and IBM. The movement of the tape industry to support the document image industry, began in late 1989 when groups such as the Tape Head Interface Committee (THIC) were exposed to new business opportunities. These groups became instrumental in providing a pathway for vendors with systems that offered a higher transfer rate, lower per megabyte cost and higher per unit density, to address those areas of the electronic document industry where they were best suited. To complete the offering from the tape industry, Creo manufactured a drive to record data on a non-erasable reel of optical tape from ICI ImageData. A single 12.5 inch reel holds 1,000 GBs or 1 terabyte (TB).

The compact disc (CD), was beginning to enter the market place carrying not only published music, but published data. As early as 1986, vendors could demonstrate interactive Compact Disc-Read Only Memory discs (CD-ROMs) on \$10,000 systems. These systems were not ready for the massive consumer market, but they certainly could point toward the future. Published reference materials or marketable information databases became some of the first to be available on this new media. Entire industry consortiums sprung up in support of the 650 MB disc that could be duplicated from a master tape for under \$5 and sold at a tremendous profit. The next logical step in the evolution of the CD was to sell recorders to the public so end users could create their own masters. These recording systems were introduced at \$15,000 and are predicted to soon fall under \$700.

The first 12 inch optical platter in 1984, held 1 GB of data. A dozen years later, the commercial capacity is 12 GB. Tape formats have also increased. VHS started out at 5.2 GB and now can store up to 50 GB. The recordable 650 MB CD will not improve in data storage capacity until 1998-99 when it is projected to reach 2.6 GB.

Introduction

This paper addresses at a high level, the many individual technologies available today in the removable storage arena including removable magnetic tapes, magnetic floppies, optical disks and optical tape. Tape recorders represented below discuss longitudinal, serpentine, longitudinal serpentine, and helical scan technologies. The magnetic floppies discussed will be used for personal electronic in-box applications.

Optical disks still fill the role for dense long-term storage. The media capacities quoted are for native data. In some cases, 2 KB ASCII pages or 50 KB document images will be referenced.

Longitudinal Recorders

The first "industrial strength" tape recorders used longitudinal recorders that moved tape in a single pass across stationary heads to read or write the length of the tape. The closer the heads are to each other, the denser the recording. Futurists predict reaching a 1,000-head recorder. Early longitudinal recorders stored 180 MB on open reel 9-track tapes. These were replaced in 1986-87 by 200 MB cartridges which could be managed in automated libraries. Over 90% of the world's data centers use the half-inch 3480-type tape cartridges. STK has developed and sold over 7,500 circular library units, commonly called silos, which can house up to 6,000 cartridges. Using the newest 800 MB cartridges provides 4.8 TB of robotically-addressable storage. The silos have a footprint of 121 square feet and are large enough to allow a technician to walk inside to provide necessary service. The drives can transfer data at 52 Megabits per second (Mbps). Up to 16 libraries can be linked to offer 76.8 TB of data storage. The current generation of the silo, PowderHorn, provides two robotic arms to retrieve and load cartridges up to 350 times per hour.

Serpentine Recorders

Serpentine recorders write a single track from end to end on the tape and then reverses directions to write the second track in the opposite direction. The back and forth recording continues until all data is recorded. The quarter inch cartridge (QIC) tape format cartridge is 4 inches by 6 inches and stores 13 GB as a result of a cooperative effort between Tandberg, IBM and 3M. The data rate has reached 12 Mbps. Vendors expect the storage capacity to go to 25 GB by 1997 with the adoption of thin film media, 50 GB in 1998-99 using barium ferrite technology and 180-200 GB per tape by 2000 using multi-channel drives on thin film media. By then, the data rates should be close to 56 Mbps. There are currently 8 million QIC drives in use today. In the spring of 1996, Tandberg introduced three autoloaders to accommodate 10, 20 or 30 tape cartridges.

The QIC mini-cartridge offers 4 GB in a cartridge which is 2.5 by 3.5 inches. Following the same improvement path as the full sized QIC and using multi-channel tape and thin film technology, the capacity for this unit should reach 30 GB by 2000 and the data transfer rate is expected to reach 56 Mbps. There are no autoloaders for this media.

The newest small tape format to enter the market is 3M's Travan. It was introduced in April of 1995 with a capacity of 400 MB, and by June the capacity had reached 800 MB. By December 1995, Travan could store 1.3 GB and the current capacity is 4 GB. Vendors predict that by 1997, the capacity should be 15 GB. Drives are made by HP, Conner, Iomega and AIWA. The tape is manufactured by Sony or 3M and the autoloaders are scheduled to come out from Exabyte or Conner. The Travan drive will accept the mini-cartridge as well as all previous generations of Travan tapes.

Quantum bought the storage products division of Digital Equipment Corp. and now offers their internal data cartridge as a removable storage media. In early 1996, the native capacity was increased from 20 GB to 35 GB on the half inch tape. With a data transfer rate of 24 Mbps and Bit Error Rate (BER) of E-17, this tape format should provide an interesting addition to the storage hierarchy. EMC is currently offering this media as the disaster recovery solution to support their warehousing projects. The future introduction of thin film media should provide storage of 100 GB before 2000. The data transfer rates are due to increase to 80 Mbps. The available multiple-cartridge units include a stackloader of 7 cartridges on the low end, and a broad selection of autoloaders supporting 28, 48, 60, 264, 360, or 900 tapes on the high end. The 900 tape unit comes from MountainGate and offers 31.5 TB in 18 square feet (1,750 GB per square foot). The maximum of 20 drives provides access to 700 GB of mounted data. The AML/2 from EMASS supports 32,720 DLT tapes for a potential capacity of 1,145.2 TB in 1300 square feet (881 GB per square foot).

Longitudinal Serpentine Recorders

These recorders combine the two previous technologies to record one set of multiple tracks down the length of the tape, then reverse direction to record another set of tracks back toward the beginning of the tape. The new longitudinal serpentine head assembly designed by IBM can read or write 16 tracks at a time. The media is formatted to hold eight sets of the 16 tracks resulting in data on a total of 128 tracks. IBM's 10 GB subsystem is called Magstar and it can support a 72 Mbps data transfer rate. The largest IBM library (model 3495) is 92 feet long and uses a robot on a rail to move tapes from the slots to the drives. The library can store up to 18,920 cartridges for a potential storage of 189.2 TB. The smaller model 3494 library can store between 210 and 3,040 cartridges and support a maximum of 30.4 TB using the Magstar, or 2.4 TB using the 800 MB cartridges.

Helical Scan Recorders

Helical scan technology uses multiple read/write heads to record data on tracks at a slant across the tape. This track, known as a swipe, is slanted such that the angle between the track and bottom of the tape is between 4 and 7 degrees. Some helical drives yield 15 MB per square inch or 16 KB per swipe. Major helical scan drive

components were brought back after WWII by Bing Crosby and Ampex was the first company to demonstrate this technology in 1955.

Four-mm formats

Several companies offer the Digital Audio Tape (DAT) tape format that holds 4 GB of uncompressed data on 120 meters of 4-mm tape housed in a standard cassette measuring 2.9 inches by 2.1 inches. In the fall of 1995, HP announced the DDS3 standard format which can store 12 GB by recording on longer tape with smaller tracks. The data transfer rate is 6.2 Mbps and the seek time is 40 seconds. In 1997, the DDS4 standard should be introduced and provide a native capacity of 24 GB using metal evaporated tape and a data transfer rate of 8 Mbps. Due in part to the Forward Error Correction (FEC) technology which promises a BER of E-15, this tape format has been used for CD-ROM mastering, COM (computer output to microfilm) replacement, and image storage. For automated applications, autoloaders are available to manage a variety of cassettes in several formats including carousel, vertical and horizontal configurations. The largest automated handler for this size media is available from Exabyte and holds 218 units for a potential capacity of 2.6 TB.

Eight-mm formats

The 8-mm format is primarily available from Exabyte. March 1996 brought the long awaited release of the Mammoth drive. Data can be recorded at a transfer rate of 24 Mbps. The 8-mm cassette of advanced metal evaporated tape holds 20 GB of uncompressed data. To put this capacity into perspective, the first 12-inch optical disks held a ground-breaking 1 GB. Numerous vendors are supporting this technology. Mass storage libraries have come from the back-up storage and video broadcast industry and hold 10, 40, or 80 cassettes. The EMASS model AML/2 library sets the record for highest number of tapes in a single autoloader. The maximum load would be 58,880 tapes providing 1,177.6 TB. The BER has improved from E-15 to E-17. In 1997-98, the capacity should double to 40 GB and the rate should reach 32 Mbps. By 2000, the capacity will double again to 80 GB and the transfer rate should reach 48 Mbps.

A recent entrant to the 8 mm market is coming out from Sony. This product was originally promised in June 1996 to come out in late summer at 25 GB. The drive fits into a 3.5-inch slot, smaller than the 5.25-inch Exabyte slot. A design advantage that contributes to performance, is the location of the index on a chip on the cartridge. This intelligence will allow the drive to reject the cassette without having to spend precious moments rewinding the tape. When the dust settles, the tape could be released with a native capacity of 35 GB. Time and the market will tell.

Half inch formats

The most recent tape format to enter this market is the digital tape format (DTF) manufactured by Sony. The two sizes available are the 12 GB and the 42 GB capacities. Several libraries are available from Sony to support this media. The smaller one holds 35 large tapes (1.5 TB) or 70 small tapes (840 GB) with one drive. The BER is E-17 and the transfer rate is 96 Mbps. As demand grows for capacity, other autoloaders may become available and the capacity could extend into the Petabyte range.

A long-awaited helical scan recorder was shipped by STK in 1995. The new Redwood drives store 50 GB on a single 4 inch square cartridge and raise the capacity of their silo from 4.8 TB using 800 MB cartridges, to 300 TB and 4.8 Petabytes per 16-unit cluster. The transfer rate is 96 Mbps.

MountainGate offers the VHS drive that was originally modified from broadcast industry-developed technologies by Honeywell, then Metrum Information Storage. The RSP-2150i drive can record 21 GB using T-180 tape format in a VHS cassette. For data processing purposes, 21 GB could replace 117 reels of 6250 tape. An autochanger with a footprint of 18 square feet, can accommodate 600 cassettes and provide an automated storage capacity of 12.6 TB. Legacy Storage Systems used the PRML (partial response, maximum likelihood) recording technology to store 50 GB on a T-180 VHS cassette. The origin of this system is the radio astrology industry. Currently, the company supports ganging drives together to meet customer needs rather than putting tapes into robotic handlers. With the right software, the system appears to the host as a mounted disk drive. The current transfer rate is 16 Mbps, but they have a short term goal of reaching 32 Mbps by the end of 1996. By then, they plan for the capacity to be 100 GB per cassette.

Nineteen-mm formats

The DD (Digital Data)-1 or DD-2 tape formats are available in the small, medium or large 19 mm cassettes. These cassettes are used by the broadcast industry to store TV data, by the scientific community to store instrumentation data, and by the computing community to store computer data. The drives used to record data from scientific instruments are called ID-1 (Instrumentation Data 1) and the tapes can be recorded at rates up to 400 Mbps per drive. Four drives can operate simultaneously to provide a data capture rate of 1600 Mbps. Loral, Lockheed Martin, DATATAPE, Penny & Giles, and Sony, are building these drives. The standard for the ID-1 tape was established by the ANSI X3B.6 committee and other committees are currently meeting to develop standards for the DD-1 and DD-2 formats as well. The suppliers of 19 mm tapes include Hitachi, BASF, Sony, Fuji, 3M, TDK, DATATAPE, Penny & Giles, Maxell, and Quantegy. The DD-1 tape format offers a BER of E-13 while the DD-2

format has recently been improved to E-17. The storage capacities of the six tape formats are shown below.

•DD-1S	16 GB	•DD-2S	50 GB*
•DD-1M	40 GB	•DD-2M	150 GB*
•DD-1L	96 GB	•DD-2L	330 GB*

* These doubled capacities are currently in Beta test.

Ampex currently offers a 7 square foot library that holds 7 DD-2L for a capacity of 2.3 TB. Another autochanger is available that will hold 256 DD-2S cassettes for a capacity of 12.8 TB. The DD-2 drives can reach a transfer rate of 120 Mbps. EMASS offers three sizes of autoloaders, the AML/J, AML/E and AML/2. The AML/2 has the largest capacity and could support either 16,896 DD-2S tapes for a total of 845 TB, or 10,752 DD-2M for a total of 1,613 TB in 1300 square feet. Using the DD-2 M tape would result in 1,240 GB per square foot. To relate this to a paper volume, 1,613 TB represents 32.3 billion 50 KB document images which could fill 3,226,000 file cabinets covering 753 football fields.

The Sony DD-1 drives have a recording speed of 128 Mbps. Sony offers a cassette tower which can hold all three sizes of DD-1 cassettes and either one or two drives providing a storage capacity ranging from 512 GB to 2.2 TB in less than 8 square feet. There are two larger libraries which hold 320 DD-1M cassettes for 12.8 TB capacity, or 736 DD-1M cassettes for a total capacity of 29.4 TB.

Optical Cards

Requirements for personal storage that need to be met by high density storage systems may incorporate the low-cost option offered by numerous vendors supporting the 4.6 MB optical card invented and patented by J. Drexler. This credit-card sized optical storage media can store 2,300 ASCII text documents or 92 document images using WORM technology on a media carrying a 10-year life expectancy. These cards are being used for individual medical record storage, personal registration data, and financial transactions. A test is being conducted at the border between Canada and the U.S. Soon, frequent travelers may be able to use their Canon optical registration card in addition to a fingerprint scanner to speed processing through customs.

Compact Disc Read Only Memory

One of the least expensive media for mass distribution of reference-type database information is the Compact Disc-Read Only Memory (CD-ROM). The first compact discs were introduced in 1981 to replace 8 mm audio tape cassettes with high quality digital sound discs. By 1987-88, published data became available on these systems and an alternative printing industry was born. The 12 cm, 650 MB disc can hold

325,000 2-KB pages of ASCII text. To print this many pages would require 1.9 tons of paper. There are numerous library units to house CD-ROMs. One of the interesting jukeboxes on the market can hold 240 disks in a spiral configuration. Four of these "slinky-looking" units can be connected to provide access to 960 discs. A different type of library unit holds eight rows of eight drives providing simultaneous on-line access to 64 discs. Other libraries offer larger capacities ranging from 500 discs in a Pioneer jukebox to 1478 discs in a DISC jukebox.

New CD applications include multi-media which support full motion entertainment or instructional video storage. By early 1996, the price of a drive to record a CD-R (recordable) dropped from an opening price in 1991 of \$15,000 to less than \$1,000. Affordability and availability allows the media to be used for small departmental or desktop storage applications. It has also opened new doors for small publishers to create and distribute documentation. Drives to record and erase data on the compact disc are due out before the end of 1996.

There is considerable controversy in the industry today about the forthcoming Digital Video Disc (DVD). At a minimum, the DVD should:

- 1) Record a full length feature movie
- 2) Offer picture quality better than video discs
- 3) Provide audio in three to five languages
- 4) Build in a copy-protection system
- 5) Support choices between subtitles or dubs
- 6) Insure parental-lockout features.

The proposed DVD format exceeds all of these requirements.

Agreements that are in place meet these specifications:

1. Backward compatibility with current CD media. This will protect the software vendors who have in hundreds of programs on this media. It will also protect consumers who own disc collections.
2. Two substrates - each 0.6 mm thick, with a data layer capable of holding up to 4.7 GB - bonded together to form a 1.2 mm thick disc.
3. EFM Plus signal modulation. This scheme is simpler to implement and yields a more robust and stable product, a desirable feature for the entertainment industry.
4. Reed Solomon Product Code (RS-PC) error correction. This error correction code is similar to that on the magnetic and optical media, rather than that used in CD-R. This is important because CD-Rs are not ideal for computer applications. Data cannot be placed arbitrarily on the disc, it has to chain each recording session to the end of the last session. Moving to RS-PC, allows a random, block-oriented kind of recording anywhere on the disc surface. The DVD format that will be available for recording data, will not be available to consumers until 1998-99 and the capacity will be 2.6 GB.

Rewritable Magnetic Disks

In 1995, the new Zip drive was introduced by Iomega. The price is under \$200 and the removable magnetic media is available in 25 MB (\$10) or 100 MB (\$20) capacities. The drive has since taken the personal data storage industry by storm and led to the development of an entire market for personal storage media. This demand has been encouraged by the electronic in-box revolution taking place in reaction to the ease of downloading documentation from the Internet. Other available disks offer capacities of 120 MB (3M's drive that can also read the 3.5 inch 1.44 MB floppies), 135 MB (SyQuest EZ135), 170 MB (Avatar APS 170 PB), or 1 GB (Iomega Jaz).

SyQuest offers removable magnetic drives with associated removable floppies. These magnetic drives cost between \$500 and \$700. A 2.5 inch floppy is available with a 42.8 MB capacity. A 1.8 inch floppy is also available which holds 80 MB. These media are commonly used in the printing industry to exchange color prints because a digitized color image requires roughly 20 MB of capacity.

The Bernoulli disk from Iomega, is based on barium ferrite media. The current storage capacity of 230 MB is spread over two platters that can be accessed simultaneously. These two disks are back-to-back in the cartridge and as they spin, the air between them forces them away from each other and toward the recording heads. They are often used in rugged environments because if there is an interruption in power, the media falls away from the heads to avoid any head damage to the disk or loss of data. This media is also a favorite of some security-oriented organizations because the media can be shredded.

Rewritable Optical Disks

Magneto-optic (MO) technology uses optical properties and some principles of magnetic storage to store data on 2.5-, 3.5-, 5.25-, 12- and 14-inch optical disks. To replace data on a MO disk requires the first head pass over the disk to erase the data, a second pass to let the track cool, and the a third pass to record the new data.

Desktop requirements will soon be met by the 2.5 or 3.5 inch optical disk. Sony offers the 2.5 inch MD-Data disk that holds 140 MB, the equivalent of 15 minutes of video (using MPEG-1 compression). The current capacity of the 3.5 inch disk is 640 MB. These disks can spin faster and provide shorter seek times than their 5.25 inch counterparts. A desktop jukebox was introduced by Fujitsu at the 1996 AIIM show. It can hold up to 35 of the 3.5 inch disks and up to two optical drives providing 22.4 GB of storage.

The 5.25 inch disk currently holds 2.6 GB. This size of disk has become the media of choice for network and jukebox applications. Multifunction drives have been introduced by numerous vendors that will write to either 5.25 inch WORM or erasable platters.

Phase change technology can write directly over the data on a track and replace it with new data, therefore saving time required by the MO drives. The popularity of phase change media is expected to grow. In the fall of 1994, Matsushita introduced the phase change dual (PD) drive. This drive is being considered a standard in the European community. This drive records on a rewritable 4.72 inch phase change disk but it can also read a 4X CD-ROM to serve a dual purpose for the consumer. Two new jukeboxes are available from Pioneer. The 50 platter unit provides 32.5 GB and the 100-platter unit provides 65 GB.

Nikon has combined the two technologies (MO and phase-change) and is bringing a direct overwrite MO disk to the market. They market a 12 inch disk with the storage capacity of 8 GB. The transfer rate is 1.1 Mbps and the cost is \$1,100 per disk. This disk and drive can be installed in the ATG Cygnet jukebox and there are customers in Korea who are interested in using this system for the management of governmental records. Currently, Lockheed Martin is developing a 14 inch disk that can hold 12 GB and meet Mil Standard E-5400. It is not known when or if this product will be made commercially available. Kodak is also working to bring an erasable disk to the market before 2000.

WORM Optical Media

Non-erasable WORM media is available in 5.25-, 12-, or 14-inch sizes. By the end of 1995, the 5.25 inch platter held an average of 2.6 GB. There are numerous jukeboxes that are available to accommodate this size media. These vary in size from 10 to 1,054 platters.

Philips LMSI offers a 12-inch WORM drive that operates with a dual-head so that each side of the disk is available to the user simultaneously. The user is provided with 12 GB of storage without the need to flip the disk in a standalone environment. To compliment this drive, a 6-platter magazine fits as a single unit into a modified version of the drive providing 72 GB of storage with a disk swap time of less than 3 seconds. The removable magazine can easily be vaulted for security.

In 1995, Sony announced a 15 GB disk and dual-head drive that should be shipping by the third quarter in 1996. The drives will be available in one of two different jukebox configurations. For every drive, the jukebox frees up space for 10 platters. The smaller jukebox will handle from 1 drive and 76 platters, to 4 drives and 46 disks. The trade-off ratio is the same for the larger jukebox. The drive/disk combination will run

from 2 drives and 156 disks to 6 drives and 116 platters. Disks in 6 drives will provide the user with 90 GB on-line simultaneously.

Kodak's latest 14 inch optical platter has the capacity of 14.8 GB and incorporates a non-erasable form of phase-change technology. The 132 platter Kodak jukebox provides access to 1.9 TB. The new dual-sided 25 GB disk and drive is due out in the fall of 1996. That introduction will take the large jukebox to a 3.3 TB capacity.

Optical Tape

This media is one of the most exciting and versatile on the market today. It was originally introduced into the U.S. in 1986-87 by ICI Imagedata. A 12.5 inch open reel can hold 1 TB of data. As with other open reel systems, there is no way to automate the system as it stands today. The tape must be mounted on a floor-model drive that occupies 6 square feet. Transverse recording writes data perpendicular to the length of the tape. Of interest: if you were to print out 1 TB of ASCII data, the paper requirements would consume 42,500 trees. A single TB would accommodate either 1 million 500 page books, or 1600 CDs, or 2000-4 drawer file cabinets, or 5,000 9-track tapes. There are systems in UK, Canada, Republic of South Africa, and Australia in addition to the U.S. Applications include satellite data, oil data, medical images, transaction processing and document archives.

Other vendors, including Kodak and Dow Chemical, developed different types of optical tape. There are also several vendors working to perfect an optical tape cartridge or cassette subsystem. Most of these are members of the Optical Tape Study Group lead by Fernando Podio at NIST. The subsystem closest to commercialization will be coming from LOTS Technology. The engineers at this company are building their own drive to read Kodak optical tape in a 3480-sized cartridge. This drive will fit into any of the autoloaders which uses the 3480-type cartridge. The drive can record up to 1 TB on a single unit at 120 Mbps. LOTS Technology could expand STK's 6000-unit 3480-type library capacity to 6 PB. Sixteen libraries would provide 96,000 TB or 96 Petabytes of robotically-addressable storage. The 96 PB could replace 1,920 billion image documents -- roughly equal to the volume of pages in 9,600 Library of Congress buildings. Beta units should be available by the close of 1996.

Thin Film Media

The NT-1 cassette from Sony contains thin film media which is only 2.5 mm wide and 4.8 microns thick. The tape can record 45, 60, 90 or 120 minutes of digital music, up to 53 minutes more than an audio CD. The length of tape is 20 meters and it has already been specked out for data — 612 MB! The digital data tape would replace 3.4 9-track tapes, 4 file drawers holding 12,240 document images, or 306,000 ASCII pages that would take 3,570 pounds of paper to print.

Unfortunately, the system was withdrawn from the marketplace in late April, 1996. Only time will tell if it reemerges or not.

Conclusion

The abstract for this paper was submitted in February. Changes were made to the body of the paper when it was revised in early May. The July revision, was made due to the technology advances since that point in time. By the time this paper appears in the proceedings for September, more products will have been introduced, some vendors will drop out and other vendors will enter the market. If you have updates to add, give me a call and I will reflect them in future releases of my book. If you want to check what has changed since this paper was released, I invite you to contact me!

References

L. Kempster, *Media Mania! The Fundamentals and Futures of Removable Mass Storage Media*. Avedon Associates: Potomac MD, Sixth Revision, 1996

F. Jorgensen, *The Complete Handbook of Magnetic Recording*. McGraw-Hill: New York, NY, Fourth Edition, 1995

The Cornerstone of Data Warehousing for Government Applications

514-82
83196

Doug Kenbeek and Jack Rothschild

EMC Corporation

Hopkinton, MA 01748-9103

rothschild_jack@emc.com

<http://www.emc.com>

1-508-435-1000

1-800-424-EMC2¹ (in North America)

Abstract

The purpose of this paper is to discuss data warehousing storage issues and the impact of EMC open storage technology for meeting the myriad of challenges government organizations face when building Decision Support/Data Warehouse systems.

Introduction

Most technology advisors in government believe that data warehousing is a perfect match with government agencies. The reason is because data warehouses work best for large organizations with mission-critical data distributed on a variety of heterogeneous systems – as is often found with federal, state and local government agencies. Although slow to jump on the data warehousing bandwagon, agencies have begun developing full-blown data warehouses.

Most data warehousing planners focus their efforts on four foundation pieces - or cornerstones - of a data warehouse: (1) the operational data and its acquisition, transformation and integration into a data pool, (2) the database management system and associated servers for managing the data pool, (3) the client DSS applications, and (4) the storage system where the information resides.

One of these cornerstones if planned incorrectly will cause enormous waste and frustration and can make the entire DSS susceptible to collapse. Yet it is the one cornerstone that usually gets the least amount of thought and planning. The hidden

¹ EMC², ICDA, and Symmetrix are registered trademarks, and EMC, Centriplex, and SRDF are trademarks of EMC Corporation. Other trademarks are the property of their respective owners.

This paper is being distributed by EMC Corporation for informational purposes only. EMC Corporation does not warrant that this document is free from errors. No contract is implied or allowed.

Abbreviations used: DW - Data Warehouse, Data Warehousing; DSS - Decision Support System; OLTP - On-Line Transaction Processing

cornerstone is the storage system that physically manages the movement, placement, backup, and restoration of data.

Potential problems associated with data storage are acute because the DW places greater stress on the storage system in terms of data volume and seek functions than operational data from business process systems. And the value of all that data is entirely dependent on the protection and speed of data movement provided by the storage system. If it doesn't work well - the DSS is compromised.

A discussion of open storage technology and its impact on DW environments must take into account other drivers of information technology change. Trends in servers, hardware, software, data management, networking, geographical distribution of systems, I/O management, failure rates, procurement strategies, and disaster recovery are all important to consider when trying to understand the benefits of EMC open storage technology.

This paper briefly recaps the history of computing and storage, reviews some current trends, and then progresses to the problems associated with storage in today's expanding data warehousing operations. It concludes with a description of the key storage shortcomings inherent in DW environments and the EMC open storage features that can overcome both long- and short-term challenges when managing Decision Support/Data Warehousing implementations.

Information Technology Recap

There are three distinct phases in information processing: the automation of labor intensive tasks, online transaction processing, and data warehousing. In essence, these represent a transition from CPU-centric computing to data-centric computing to information-centric computing. This evolution parallels the transition from batch computing to "realtime" processing to the distribution of information and empowerment of knowledge workers. In many ways they are synonymous and represent similar challenges. Each phase addresses a business's return on investment and produces its own technology challenges.

Significant trends are related to these phases.

- Generation of data is increasing with the expansion of OLTP and DW.
- Computing is transitioning from a CPU-centric to an information-centric orientation.
- Management challenges are increasing exponentially with increased demand for information.
- Information is now the key to service enhancements for all government organizations, and is increasing the pressures and demands on suppliers and implementors.

Trends in Data Technology

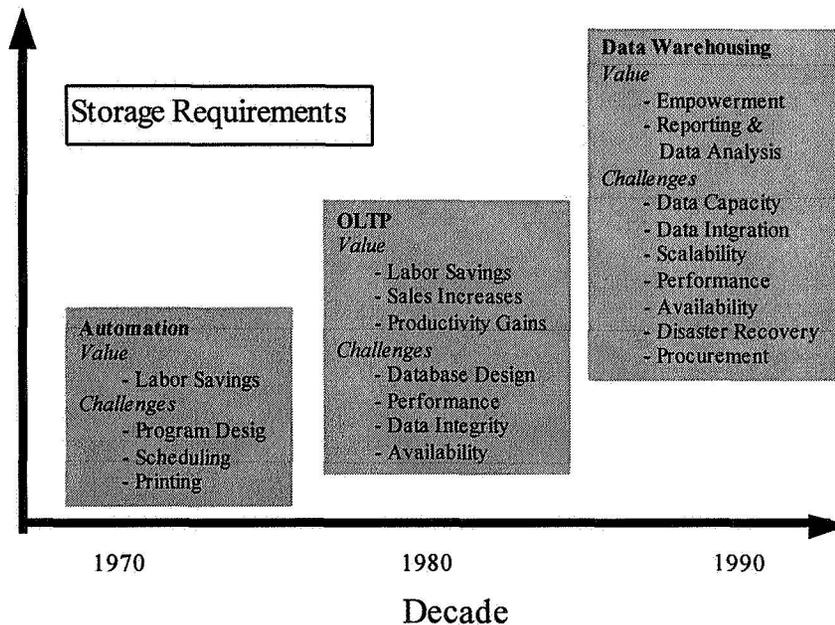


Figure I - Trends in Data Technology

The first phase, batch-oriented data processing, is not relevant to DW and not part of this discussion.

Phase two, OLTP, is now widely deployed with a trend towards client/server implementations and more widely distributing the users and data. This is where most RDBMS systems are now active as organizations continue to move operations online. Most legacy and second generation online applications are moving to this type of implementation. The databases in these environments are growing fast, with more than 10GB the norm and many growing to hundreds of gigabytes.

Data Warehousing

Phase three, the latest information processing trend, requires information managers to adopt a concept known as data warehousing. DW promises employee empowerment and creates the demand for a broad range of historical information presented in a useful format. So in addition to demanding access to critical operational data, end users also are seeking historical information to accomplish key job functions; analyze program impact and effectiveness, trends analysis, improve citizen services, and help identify and reduce fraud or other inefficiencies. This information is dominated by standard forms of textual or image data, but increasingly can include voice and video data. More complex data types, due to their large sizes, greatly increase the demands on the storage and communications systems.

Operational data, from which the data warehouse is constructed, is typically transported to a database in a centralized repository (data warehouse) where it may again be distributed to organizational servers. The operational data is scrubbed for inconsistencies and converged to eliminate duplication in the DW. All this movement, storage, and cleansing of data requires a high level of storage system performance and integrity.

Data warehousing creates historical data from operational data. Decision Support Systems gather DW data or summaries of it and transform it into easy to understand information. Since operational data is OLTP-oriented information gathered from the applications that run day-to-day operations, the DW database is systematically updated so operational data is represented to a known point in time. The speed of the DW update is dependent on the performance of the storage system. The more frequent and more voluminous the updates, the more critical the Decision Support Systems and the storage system.

The data quality, access time, and window of availability are of extreme importance since DSSs depend on the DW to produce their information. The DW storage system affects the integrity of data in the DW, the speed that the DW data can be accessed, the availability of the DW itself to the server system, and the efficiency of the updates to the DW.

EMC's family of Integrated Cached Disk Array (ICDA®) and high performance backup solutions directly address the storage requirements of a modern data warehouse while preserving the ability to choose best-of-breed technologies for other DW components.

Open System Server Dependence on MIPS

Storage subsystems for open systems have predominately followed a server manufacturer, CPU-centric model. The storage system being provided with the server. While it is often possible to substitute controllers and disk drives to increase capacity and/or performance, the limiting characteristics of these storage subsystems has not changed substantially.

CPU performance has considerably outpaced server I/O performance in the open systems arena as illustrated in Figure II.

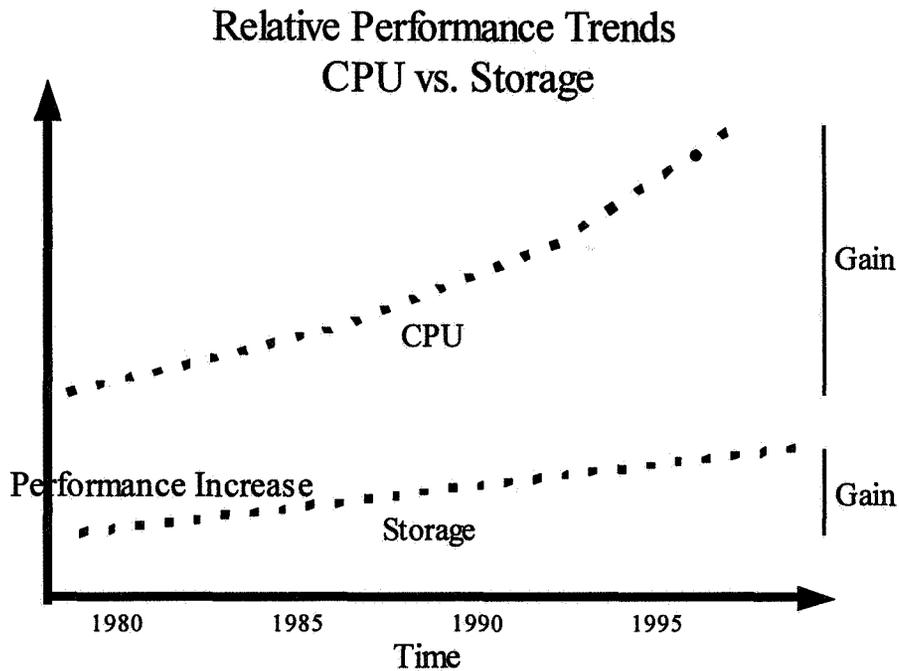


Figure II - Relative Performance Trends: CPU vs. Storage

Servers/Databases — CPU Focus

Generic and portable operating systems, industry hype, and database developments have contributed to the problem. Simply, storage subsystems have not been a focus of server hardware architectures. Open systems DW servers suffer from limited storage expansion capabilities, and often, increased storage requirements force customers to upgrade CPU types and cabinetry for increased data capacity. Many times, I/O communication channels are overloaded in fear of using up additional CPU or memory expansion slots. DW database performance is a function of both CPU and I/O performance, so high performance open storage improves overall performance and off-loads the CPU. This way the entire system is better utilized and more in balance, in many cases eliminating or deferring server upgrades.

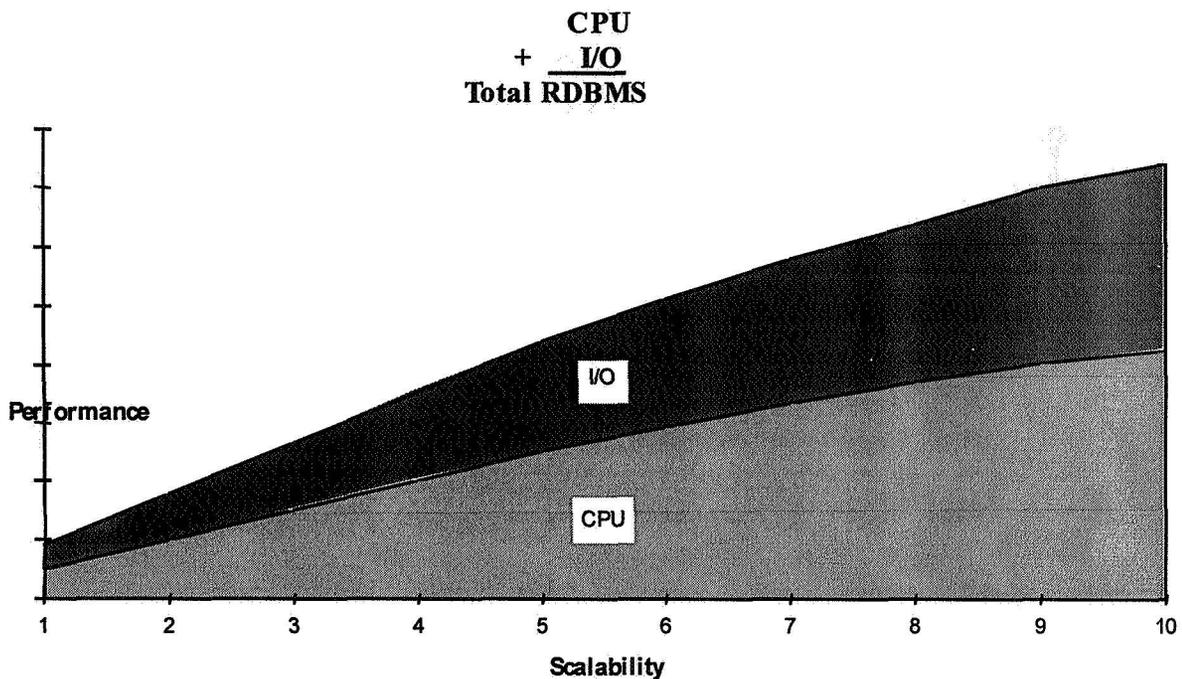


Figure III - RDBMS Performance

EMC accomplishes dramatic throughput improvements for updates and queries by providing large amounts of onboard cache memory in an intelligent, microprocessor-based storage system. By focusing on the storage component, EMC has optimized the performance and management of this unseen but critical DW component. EMC's ICDA system manages the disk drives, controllers and diagnostics as a coordinated system, relieving the CPU from these overhead tasks and delivering the highest levels of performance.

Open system hardware is made for generic use. Open systems servers may be used as communications gateways, clients, X-windows servers, firewalls, video servers, and OLTP or DW database servers. These applications range from CPU-intensive to disk-intensive environments. The majority of servers utilize the same software and hardware technology and are not primarily designed for storage-intensive applications. Server research and development expenditures bear out this fact, with the majority of investment dedicated to chip, operating system, and CPU design.

Server storage is off-the-shelf technology and "bolted on" using standard components and connections. There are a range of connectivity standards and device types, the most common being SCSI (Small Computer Systems Interface). Typically, an I/O card plugs into the server system bus and supports multiple SCSI disks per card. The storage system is built into the server cabinet, but not in a very integrated way. The disks operate independently or if there is an intelligent controller providing some coordination, it utilizes CPU cycles. Sharing of storage to boost utilization is rare between homogeneous servers and not supported between heterogeneous servers.

To address availability concerns, many server vendors have implemented RAID 0 (disk striping) and RAID 1 (disk mirroring) functionality in their systems. Although this helps increase performance and reliability respectively, there are still many points of failure within their storage subsystems and these features are again using valuable CPU cycles. The failure of a power supply, fan, SCSI channel, or controller may cause an entire bank of devices to fail. The use of RAID 1 technology within the server is questionable since many other components are unprotected and they can cause the mirrored disks to fail.

Database developers have created portable products that perform across many hardware architectures. Consequently, performance is software-oriented and highly dependent upon the speed of the processors. I/O bottlenecks and performance degradation are often addressed through CPU upgrades. CPU upgrades frequently cause a rippling effect in the remainder of the server system resulting in device and controller upgrades, downtime, and hardware incompatibilities. The reliance on MIPS for database performance has kept server emphasis on CPU technology and placed storage technology on the back burner.

Data warehousing is by nature both a storage-intensive and storage-expansive application area. Although open systems servers and databases tend to be CPU-oriented, EMC has developed the ICDA system to elegantly integrate intelligent storage algorithms with high quality storage hardware. This enhances the performance, scalability, availability, and reliability of the DW storage component. And these storage systems easily connect to every major open systems server without the need for special devices or drivers. EMC enables open systems DW servers and databases to scale and support small, medium, and large data warehouses effectively.

Distribution of DW Data

Decision Support Systems rely on the decentralization of information to the DSS user, but widely distributed applications and hardware have brought about difficult challenges in infrastructure configurations, availability, and systems management. With ever-improving communications bandwidth and technology, DSS applications running on intelligent clients can be distributed to the end user while maintaining data warehouses in a centralized or nearly centralized state. Single DW servers or small groups of DW servers provide many data security, availability, and operational advantages over a widely distributed DW scenario. There are good reasons for centralizing DW data while maintaining a highly distributed DSS environment.

In a distributed environment, storage devices are usually purchased independently for multiple server types at multiple sites. Server upgrades and consolidation may necessitate that storage devices be abandoned at worst or physically reconfigured at best. This disruptive cycle usually requires field engineers, OS administrators, and application experts to “qualify” new configurations and is both a labor- and training-intensive effort. For the DW implementation, it is therefore more effective to minimize the number of data

warehouses, centralizing the data as much as possible. This could provide dramatic impact for megacenters and other government consolidations currently under way. This increases utilization of the storage investment. For sites with multiple data warehouses or other co-located systems, EMC supports the ultimate in storage flexibility — hot re-allocation of storage devices to heterogeneous servers. The result is extremely high storage utilization.

Storage management operations differ among CPU server types within a vendor's range and usually differ among vendors. There are different routines for mounting, unmounting, striping, and mirroring devices. Methods differ in the way bad spots are mapped to existing and mirrored pairs, in the way failed mirrors are swapped out, and in the routines used to resynchronize the devices. In a mixed server environment, different disk storage subsystems require extensive training, configuration, logistical, and operational knowledge resulting in labor-intensive and error-prone operations. Databases that replicate full or partial data warehouses require storage management knowledge for every type of server involved. Obviously, the greater the DW distribution the greater the potential overhead. EMC uses a single management scheme regardless of the server attached, even if multiple concurrent open systems servers are running on a single ICDA system. This simplifies the storage management challenge even with distributed data warehouses.

DW Data Availability

Some argue that the DW data is not “business critical” and so should not be considered for protection. We assume that an organization's investment in the DSS/DW is substantial, both in dollars and human resources, and that the DW data itself is key to at least one aspect of a firm's management. Consequently it is important enough to protect.

Data warehouses need or will need to store large amounts of data, so the high number of storage devices required in either centralized or distributed servers results in higher error and failure rates. The number of components that comprise a system are directly proportional to the failures rates experienced.

Standard Server Storage

To avoid failures and associated downtime in the DW, many servers mirror their storage (RAID 1) requiring a like spare for each primary drive. However, a failure of a disk controller can also cause entire I/O channels to become unavailable. To avoid this, server vendors require that all mirrored channels reside on separate controllers. Storage devices must be load-balanced between the channels, that is, every other device on a similar channel are primary with the remaining devices mirrored spares. This scheme works fine until a controller fails, causing all I/O to be achieved on a single controller, reducing performance. In extreme cases, servers may use dual-port controllers to continue mirroring spares in case of a controller failure. All of these methods require intimate server expertise and prove to be a cumbersome and administrative-intensive solution.

It should be noted that server mirroring of DW storage subsystems of this size helps reduce some downtime, but additional storage devices, controllers, and channels can greatly increase the failure rate and further increase administration requirements. To minimize this type of DW storage failure and to simplify administration, EMC efficiently packages all the necessary DW storage components into a storage cabinet. A single ICDA system includes all disks, from 35 GB to 1.1 terabytes of storage, as well as duplexed fans, power supplies, backup batteries, and controller boards.

Electro-mechanical disks tend not to fail from one second to the next, rather, over a period of time. Monitoring storage devices and their associated components for errors usually requires manual filtering of device logs. EMC again outpaces server storage with extensive automatic diagnostics, reporting, and self-correcting capabilities. DW failures are detected and corrected in time to prevent loss of data.

Open systems server storage suffers from inconsistent management utilities and limited fault detection/correction operations. EMC overcomes these DW challenges, ensuring the delivery of DSS information.

DW Updates/Backup/Recovery

Government departments and agencies like Defense, Intelligence and Secretary of State are now operating longer and increasingly on a global basis. Data warehousing/decision support systems are following this trend. DSS applications are beginning to drive a constant demand for online information over flexible work hours, increasing storage requirements and distribution of DW data over multiple time zones. This has the effect of significantly decreasing both the DW update window and the archival window for DW database managers.

Since most data warehouses are read only, backup can be viewed as disaster protection. In data warehousing, data corruption or deletion is caused principally by a programming error or by actual physical damage to the storage system. Programming errors can occur during an update or during a database modification, a frequent occurrence at many DW sites. Updates to the data warehouse use DW server resources, slowing DSS queries, and may require shutdown of the database. A large data warehouse, a frequently updated DW, or a combination of these requires a high performance storage system to minimize update time. Unless the DW is offline for an extended period, high performance backup or online backup protects DW information from deletion or corruption.

For DW operations that run 7 x 24, online backups appear to be a solution, however they can create serious server performance degradation hindering productivity and workflow. Complicating the challenge, open systems DW databases (RDBMS) environments offer minimal archival utilities, usually limited to files recognized by UNIX® file structures. Database tables are treated as a single large volume that restricts the granularity of the restore. Restoring a single row of one table would require restoration of the entire database. Conversely, database-supplied archival methods ignore operating system files.

Information managers are often required to maintain multiple archive schedules and utilities for each DW operating system and database.

EMC storage systems support up to 32 concurrent channels and intelligent writes to disk, speeding DW updates. In addition, the EMC Data Manager backup system provides high speed backup at up to 78GB per hour, with support for RDBMS integrated online backup due in 1996. Additionally, the ICDA system's local mirroring augments data protection without a performance penalty, and EMC's unique remote mirroring feature is a reliable disaster recovery method. EMC solutions enable database managers quickly update the DW or quickly recover in the event of a disaster or data loss.

Data Centralization

Although data warehousing is different in many ways than OLTP systems, it is likely to benefit from one OLTP trend — the recentralization of data.

As many agencies downsized or consolidated their organizations and distributed computing, management difficulties increased, downtime increased, productivity decreased, and departmental computing hungered for empowerment. Soon they realized the burden of the operational realities, and that they were not prepared and did not have available the tools or expertise to manage their own environments. Distributed management tools for decentralized data were and still are in their infancy.

This may be the reason for an interesting and significant trend taking place in the industry. Government organizations continue to crave more and more information, as can be seen in the move to add decision support/data warehouse applications, yet are returning the management of distributed systems back to IT. To effectively manage this distributed data, IT is centralizing the management of storage while maintaining distributed applications. This tends to increase productivity and lower procurement and operational costs. Figure IV illustrates the trend to centralize data centers found in the commercial market. Although slower in its initial implementation of DW, government entities are expected to see similar results.

Data Center Centralization Trend

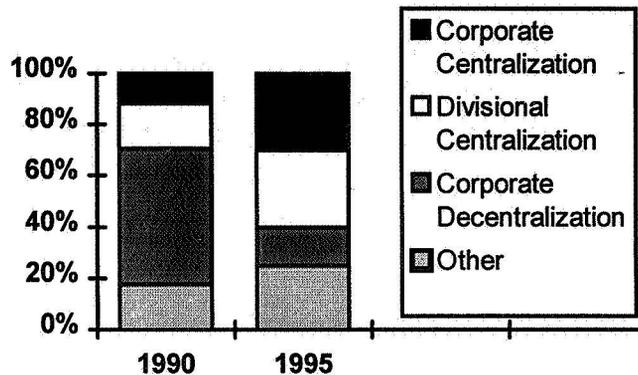


Figure IV - Data Center Centralization Trend

In less than five years, a major paradigm shift has taken place. The challenges mentioned here have database managers and the information industry evolving architectures that massively or regionally centralize data. In 1990, more than fifty percent of all companies were creating distributed architectures. In five years, less than fifteen percent of the industry is planning to continue decentralizing corporate information. More than two-thirds of industry is implementing strategies that centralize information in some fashion. This trend is reflected in numerous government agencies from defense megacenters to civilian data centers.

The trend is more than just a consolidation strategy. We are seeing a major change in the procurement, investment, and management strategy of storage. The industry is moving from a CPU-centric to an information-centric mind-set and data warehousing is leading the charge. Automating manual tasks and online operations no longer supply the competitive edge to businesses. More and better information, the promise of data warehousing, is the key to successful ventures, products, services, productivity, and roll-outs.

There have been two significant changes in the procurement and investment of hardware and software in the open systems market. Open architectures, such as UNIX, enabled the customer to purchase hardware independently of the manufacturer, protecting software investments. Relational database products enabled the buyer to further protect the information investment, procuring hardware and software solutions independent of the information.

The industry is now recognizing that an open storage strategy, adopted as an autonomous entity, is a natural extension of the open systems model. The storage subsystems should be procured, maintained, and upgraded independently of the CPU, operating system, and database in much the same vein that a network is not dependent upon database or

hardware vendors. EMC has pioneered this model in the large data center and now is offering the same advantages for open system servers.

Information is becoming the primary focus of organizations and will become the single focus prior to this century's conclusion. The drive toward DW is evidence of this inevitability. A robust storage architecture is fundamental to the availability and management of this information. Builders of data warehouses who recognize this paradigm change and implement intelligent storage management strategies make their organizations more competitive.

Intelligent Open Storage Solutions for the DW

As discussed, data warehousing has some storage attributes in common with operational systems, but it also has its own unique requirements.

The following characteristics are necessary for DW storage architectures to achieve the benefits of an information-centric strategy.

- High Data Integrity Performance
- Open Architecture
- Scalable/Very Large Capacity
- Continuous Availability
- Intelligent Management
- Disaster Recovery

High Data Integrity Performance

In the past, economically protecting data and performance have been mutually exclusive goals. The use of RAID 1 (disk mirroring) technology provides consistent performance, but requires twice the amount of disks. Other RAID implementations provide data protection with only one extra disk for each four or five operational disks, but are weaker performers. EMC ICDA systems offer industry-leading RAID 1 performance and RAID-S, a high performance RAID 5 implementation.

EMC open storage systems have implemented very large caches (up to four gigabytes) that contain recently used data as well as buffering for the latency of writes to multiple devices; excellent for DW updates. This cache is nonvolatile, as a power loss would be catastrophic resulting in lost data. EMC systems also provide the ability to multiplex I/O, that is, convert synchronous requests from servers into parallel reads and writes further increasing performance. This use of RAID technology combined with large nonvolatile caches and parallel I/O, provides a high performance, high availability DW storage environment.

Many databases require the use of extensive server memory to mimic I/O caching. This is undesirable as the operating system, applications, and network are also competing for

limited memory resources. This caching memory must also be duplicated for each individual database server. EMC storage systems remove these I/O constraints and have the ability to increase performance substantially with a proper configuration. DW database tests have shown from 1.5 to 4.0 times performance gains depending upon operation (load, create, index, join, scan, transaction, check) on the ICDA system.

Open Architecture

An open storage architecture is relative to the storage system design and the technology used in open systems servers. It allows for ease of connectivity with multiple servers and protects the DW storage investment. EMC uses a modular approach, utilizing “best-of-breed” standard components. This lets the DW storage system adopt technology improvements in line with industry standards and trends. Storage systems based upon proprietary interfaces isolate and limit the DW as well as add cost through specialized management and shortened life cycles.

In most cases, open systems servers use SCSI (Small Computer System Interface) as both the interface and device standard. SCSI interfaces allow servers to be attached through standard SIC (SCSI Interface Cards) controllers. IBM® has created a new interface called SSA, but it has not been adopted as an industry standard at this time and could be a lock-in strategy for customers. A competing industry-driven standard is based on a fiber interface, but is still in the development stages. EMC is tracking both technologies closely and will integrate based upon market demand.

EMC’s open architecture is a proven design with thousands of customers. It is ideal for DW/DSS implementations because it is flexible enough to address the unknown twists and turns the DW is likely to take as it grows and matures.

Scalable/Very Large Capacity

A DW storage solution should allow for simultaneous connectivity of servers accessing channels to all storage devices. EMC ICDA systems are highly configurable, allowing dedicated or shared access to devices without rewiring, cabling, manual switches, or removal of drives. This is the difference in implementing a DW storage system as a logical information center versus a physical configuration. The storage system is separate from the server to accommodate multiple attachments and remove dependencies on the server vendor’s design. EMC systems’ physical assimilation of storage includes attachments for multiple servers each with multiple channels. The DW can grow in servers or storage and be accommodated by the ICDA without the need to trade-out existing storage.

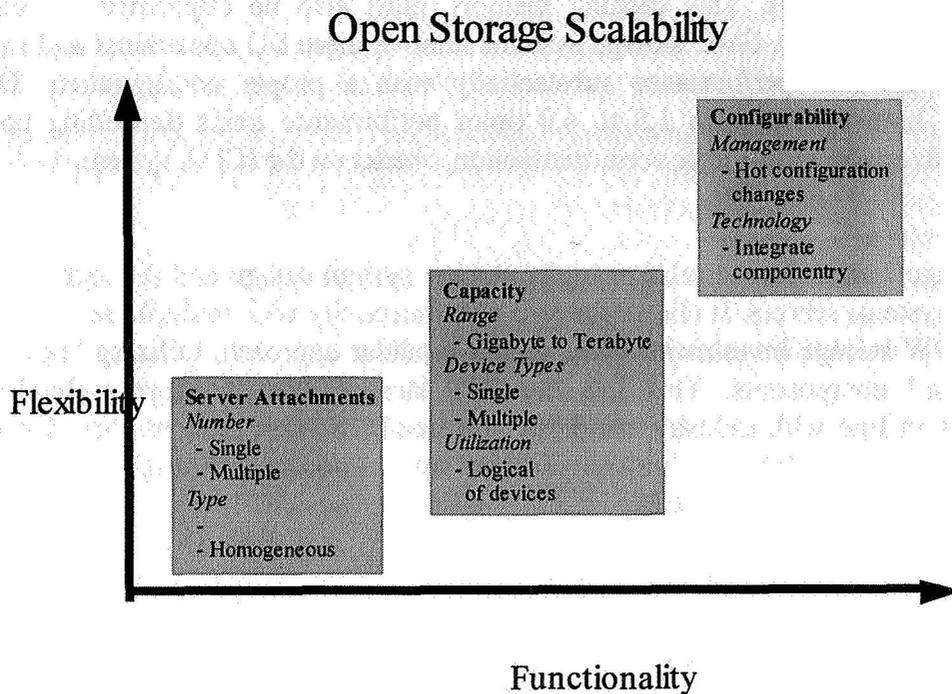


Figure V - Open Storage Scalability

EMC scalability/capacity features include:

- The ability to increase server attachments,
- The ability to increase the number of channels per server,
- The ability to increase the number of devices per channel,
- The ability to sustain DW performance while upsizing the storage configuration, and
- The ability to grow the DW to hundreds of gigabytes in a single system.

EMC's separation of storage from the servers and its terabyte capacity gives the DW managers the ability to logically assign, switch, remove, share, and manage storage independent of the servers. This functionality has significant positive implications when faced with server configuration limitations, upgrades of servers, server additions, and server consolidations. The storage investment is maintained independently of the server investment and configuration changes can be accomplished reliably and easily.

Continuous Availability

Large DW storage systems can be prone to failure as previously discussed. Consolidating storage systems into single or multiple storage environments does not preclude the need

for continuous availability, rather, it forces the issue. A single component failure of a consolidated system without redundant components can cause increased downtime as the failure may effect all devices. A single fan failure in a cabinet serving multiple servers could cause all server applications to become unavailable.

EMC ICDA systems use redundant components to handle the entire load based upon a sibling failure. For example, one power supply can carry the entire load of a system in case of a failure. This is also true of fans and controllers and even buses within an EMC storage system.

Availability not only means operational, but sustained performance. Open system servers' I/O subsystems have mirrored components to some degree, but suffer from economies of scale. They must duplicate each I/O subsystem with multiple matched pairs. EMC consolidated storage requires that only a few matched pairs are necessary for an entire ICDA system.

EMC's RAID implementations discussed previously protect access to the DW by preventing the halt of a DW server due to a failed disk and also enable the recovery of data from the failed drive. The EMC advantage is that the ICDA is operating as a system, not using valuable server CPU resources to manage the RAID and other availability features.

In addition, EMC storage systems can allocate "hot" spares in case of a failure. This spare can be allocated as a replacement device for any failed storage component. Hot spares practically eliminate the vulnerability of a hard failure by narrowing the time window of repairing the faulty device.

EMC's continuous availability features utilize modular technology to both repair and upgrade systems. All components are field serviceable and cause minimal disruption. Continuous availability and storage consolidation increase DW access, permitting volumes and databases to be reallocated to other servers in case of a server failure. EMC's high availability architecture delivers information availability as an economical added value of open storage consolidation.

Intelligent Storage Management

Storage system monitoring, detection, and reporting, combined with collaborative support and management standards are an integral part of EMC storage products. In this way, DW storage problems are not catastrophic as redundant systems or intelligent algorithms recover the failed component. EMC open storage systems also include online access from a 7 × 24 support organization to monitor and diagnose problems instantaneously with minimal disruption.

In addition EMC provides a centralized management console for configuring and managing the definition of attachments, channels, physical and logical groupings, and RAID levels as needed. This results in simplified, proactive storage management.

Backup/Disaster Recovery

As mentioned earlier, backup windows for offline archivals are rapidly decreasing and performance degradation for online backups inhibit further data storage expansion. Offline windows can be expanded with increased I/O performance and the throughput of online archives improved. The increased performance of EMC open storage systems may in itself suffice for increased backup demands.

For environments requiring massive data recovery, the EMC Data Manager closely couples the backup/recovery system with the ICDA system. This high performance product automates online data archiving transparent to the application and minimizes operations, training, and skill sets of administrative staff. It also provides security of all distributed data by consolidating and managing it as a single logical entity.

EMC has a unique feature — Symmetrix Remote Data Facility (SRDF™) that duplicates disk information transparently to a second local or remote location to provide continuous business operations in the event of a storage center disaster. This is accomplished with a robust fiber communication interface that supports sustained high performance data transfer over T3 lines.

Summary

Government entities are seeing the continued expansion of OLTP and the emergence of data warehousing. This is forcing a rapid transition from a CPU-centric to an information-centric information infrastructure. Dramatic decreases in storage device costs coupled with greatly increased demand for information has quadrupled storage server requirements and is enabling IT staffs to build scalable open systems data warehouses. Storage is a key technology of the data warehouse and therefore a critical element in its successful implementation.

Standard, server-supplied storage technology has failed to keep pace with DW requirements. This is partly due to server vendors' MIPS-centric development efforts and the generic design of open system servers. Decreased availability, poor management tools, inconsistent information, and end-user management delusion are forcing companies to consolidate information or recentralize data. The data warehouse adds to the problem by creating one or more additional data pools. Deployments of open system servers utilizing RDBMS DW software are confronted with the same problem. Keeping pace with the information demand while retaining the current investment in open systems technology is a major challenge.

EMC's solution to the DW storage problem is achieved by implementing a strategy that decouples the storage system from the server. This storage consolidation strategy gives the DW the flexibility to expand as the business requires — procuring servers, memory, and storage in a cost-effective manner while providing continuous access to the data pool from multiple servers. This increases DSS effectiveness and enables the implementation of a comprehensive storage management scheme.

Optimal information management is the key to competitive business strategies and EMC's open storage architecture fulfills the requirements necessary for successfully adapting data warehousing to this environment. Data centers that rely on existing server storage systems will find it difficult to cost-effectively manage their information. Open storage systems are not a trend in the industry or in data warehousing, but a major computing paradigm shift.

Business Value

EMC's intelligent storage systems are a DW advantage for organizations because they enable open systems topologies that offer the advantage of inexpensive server MIPS, ideal for the DW. An ICDA system does this by removing the storage limitations (performance, capacity, scalability, reliability, and manageability) that have previously hindered DW implementations.

Decoupling storage is investment protection and storage optimization lowers DW costs. High availability storage increases the reliability of the DSS applications and increases the DW ROI. Multiserver support and high availability deliver more information fulfilling the promise of the DW — competitive advantage.

Conclusion

Intelligent open storage is a cornerstone of every DW environment — a foundation technology. It improves the DW implementation through high capacity delivery of DSS information, more reliable information access and better storage management. A superior storage system addresses the key, hidden storage issues discussed and delivers solid business value.

The following open storage checklist provides a basis for evaluating DW storage products. Important DSS/DW-dependent applications require a check-off in the advantageous column. In addition, the service, upgradability, and storage reputation of the supplier should be heavily weighed.

Data Warehousing - EMC Open Storage Checklist

Requirement:	Desirability: Limited	Acceptable	(EMC) Advantageous
Host/Server Support Type	Single Homogenous	Multiple Homogenous	Multiple Heterogeneous
Device Sharing	None	Multiple Homogenous Hosts	Multiple Concurrent Heterogeneous Hosts
Number of Hosts Supported	Less than 4	4 to 15	Greater than 15
Platform Support	Single Vendor	Sun [®] , HP9000 [®] , IBM/RS/6000 [®]	Sun, HP9000, IBM/RS/6000, DEC [®] Alpha, Sequent [®] , Pyramid, SGI, Compaq [®] , AT&T/GIS [®] , IBM/SP2 [®]
SCSI Channels Supported	Less than 4	4 to 24	Greater than 24
Maximum Storage Capacity	Less than 100GB	101 to 256GB	Greater than 256GB
RAID Support	RAID 0	RAID 0 & 1	RAID 0, 1, & 5
High Availability Features	No High Availability Features	Duplexed Fans Duplexed Power Supplies Alternate SCSI Path Fault-Resilient Cache	Duplexed Fans Duplexed Power Supplies Duplexed Controllers Alternate SCSI Path Fault-Resilient Cache RAID 1,5 Support
Dynamic Spares	Unavailable	Single Spare	Multiple Spares
Field-Replaceable Components	Unavailable	Selected Components	All Components
Online Swappable Components	None	Selected Components	All Components
Maximum Cache Size	Less than 256MB	256 to 512MB	Greater than 512MB
Maintenance & Diagnostics Support	Error messages Field Service	Error messages Field Service Remote online support	Onboard diagnostic processors Auto error reporting to service provider Remote online support Field Service Self-maintenance Option
Proactive Support Features	No Fault Detection or Reporting	Fault Detection and Reporting to Local Location	Automatic Fault Detection and Reporting to Remote Location
Device Assignments	Hard-wired	Physical Assignment	Logical Assignment Physical Assignment
Operating System Support	Single Support	UNIX [®] Novell [®]	UNIX Novell NT [®] OS/400 [®] Mainframe
Disaster Recovery Support	High Speed Backup	Remote Mirroring High Speed Backup	Remote Mirroring Hierarchical Storage High Speed Backup
RDBMS Support	None	Oracle [®] Informix [®] Sybase [®]	Oracle Informix Sybase MS-SQL [®] DB2 [®]

515-82
83197

Incorporating Oracle On-Line Space Management with Long-Term Archival Technology

Steven M. Moran and Victor J. Zak

Oracle Corporation

Advanced Programs Group

President's Plaza

Suite 200

196 Van Buren Street

Herndon, Virginia 22070

smmoran@us.oracle.com and vzak@us.oracle.com

703-708-6778

Fax: 703-708-7919

Abstract

The storage requirements of today's organizations are exploding. As computers continue to escalate in processing power, applications grow in complexity and data files grow in size and in number. As a result, organizations are forced to procure more and more megabytes of storage space. This paper focuses on how to expand the storage capacity of a very large database (VLDB) cost-effectively within a Oracle7 data warehouse system by integrating long term archival storage sub-systems with traditional magnetic media. The Oracle architecture described in this paper was based on an actual proof of concept for a customer looking to store archived data on optical disks yet still have access to this data without user intervention. The customer had a requirement to maintain 10 years worth of data on-line. Data less than a year old still had the potential to be updated thus will reside on conventional magnetic disks. Data older than a year will be considered archived and will be placed on optical disks. The ability to archive data to optical disk and still have access to that data provides the system a means to retain large amounts of data that is readily accessible yet significantly reduces the cost of total system storage. Therefore, the cost benefits of archival storage devices can be incorporated into the Oracle storage medium and I/O subsystem without losing any of the functionality of transaction processing, yet at the same time providing an organization access to all their data.

Introduction

As organizations rely more and more heavily on historic/legacy data for trend analysis and data mining for competitive advantage purposes, it is imperative that the organization has ready access to all its data. Maintaining data on-line, both historic and current, however, comes with the price of additional hardware costs (i.e., magnetic disk devices and their controllers). As data ages, it may not be accessed or updated as frequently as current data, yet still needs to be accessed on a periodic basis. An alternative means to effectively manage and store the data becomes necessary to ensure the organization has

access to its data. The use of lower cost archival storage media for long term archival storage provides the means to control costs and have ready access to all data. How will relational data stores, such as Oracle's architecture handle the slow response times that are typically associated with archival optical devices?.

This paper will discuss a prototype system which used both magnetic media and near-line optical technology with the Oracle7 relational database management system. The architectural design contained multiple Oracle tablespaces, storing on-line transaction data on magnetic devices and storing archival information/transactions on optical disks, both accessed from the same Oracle instance. An Oracle instance consists of an area of allocated memory named the System Global Area (SGA) and a number of Oracle processes. To test the prototype, a C program and Oracle's PL/SQL modules, managed the movement of aged data from on-line tablespaces to archival tablespaces.

The goal of the prototype was to prove to a customer that Oracle's relational database management system can effectively manage their proposed system architecture, consisting of 10 years worth of data (approximately 1.2 terabytes), stored on both magnetic and optical media. The customer needed to ensure that data stored on magnetic and optical media can be accessed transparently by Oracle in a reliable and feasible manner while meeting their performance criteria. The customer requires a cost-effective means to store and provide access to their data which is expected to grow over the years.

Architecture

The prototype architecture was designed using the Oracle7 database, an optical juke box and the Archival Management and Storage System (AMASSTM) file system software. AMASS is a product of EMASS Incorporated. The AMASS file system is completely transparent and provides direct access to both optical jukeboxes and high-speed tape libraries on workstations and departmental servers. The AMASS architecture implements a block-based, direct access paradigm for virtual storage, creating what appears to be unlimited disk capacity. AMASS makes the drives and media (volumes), normally considered off-line storage, appear as a single, on-line logical device with a single mounted file system. The AMASS file system is implemented at the virtual file system (VFS) layer of the UNIX Kernel. Incorporation of the AMASS file system at the VFS layer provides system call transparency to host applications. The core modules of the AMASS file system are: Metadata On-line Index, Cache I/O module. The AMASS architecture contain file information in a fnode structure which are maintained in an on-line index database index.

The Oracle architecture consisted of a total of four tablespaces. An Oracle database is divided into logical units called tablespaces. A tablespace is used to group related logical structures together. One or more datafiles (physical operating system files) are explicitly created for each tablespace to physically store the data of all logical structures in a tablespace. The prototype architecture allocated one datafile per tablespace. Two of the Oracle tablespaces were based on datafiles created on a magnetic drive using the UNIX

file system and the other two tablespaces' datafiles were created on the optical disks through AMASS. It should be noted that the datafiles on the optical disks were on separate disks.

Oracle & AMASS I/O Architectures

Oracle and AMASS I/O architectures match up perfectly and allow coexistence of both products. Both systems use a cached block I/O management design to increase performance and minimize unnecessary I/O to a storage device.

Oracle manages its data in the database buffer cache section of the SGA. Database buffers store the most recently used blocks of database data. These buffers can contain modified data that have not yet been permanently written to disk. Users connect through user processes and communicate with the database through server processes. Oracle creates the server processes to handle requests from connected user processes. User I/O requests, via application programs or dynamic access, are handled by system processes which handles read (server process) and write (database writer (DBWR)) requests. Oracle can be configured to vary the number of user processes per server process. In a *dedicated server* configuration, a server process handles requests for a single user process. A *multi-threaded server* configuration allows many user processes to share a small number of server processes.

To access the data residing on the optical disks, the AMASS file system handles I/O requests for the Oracle processes instead of UFS (UNIX File System). AMASS implements an I/O cache area similar to Oracle's SGA. The AMASS cache consists of raw partitions which are used as the staging area to handle read/write requests between the media and the file system. If the data the user requests resides in cache, the request is satisfied immediately, otherwise I/O processes request the appropriate media be loaded into a drive and then the data is subsequently read into the cache. The data is then returned to the requesting user process, which in this case, would be the Oracle server process. This block of data will also then be stored in the database buffer cache of the SGA until it is swapped out.

Reads: If a user's request for data exists in the database buffer cache of the SGA, results are returned immediately. If the requested data is not resident in the database buffer of the SGA, a server process requests the proper database blocks from either the UFS or AMASS datafiles and returns the data to the Oracle server process. If the data is physically resident on the AMASS datafile and the requested data blocks are in AMASS cache, the data is read into the database buffer, if not, the blocks need to be physically accessed from the optical media.

Writes: Write requests between both architectures (UFS and AMASS) are handled in a similar manner. The SGA and AMASS cache have similar queuing rules. All Oracle database transactions - inserts, updates or deletes - are processed within the SGA. The changed database blocks are not immediately written to their respective datafiles. All

committed transactions are saved to Oracle redo log files to be used for database recovery if necessary. In all write operations, the Oracle server process, DBWR, flushes the dirty database blocks in the database buffer cache to the data's respective datafiles. Datafiles resident on the UNIX file system are updated immediately. The datafiles on the AMASS file system are initially written to the AMASS cache. AMASS has its own processes that are responsible for flushing its cache to its storage media, in this case, the optical disk. AMASS has one *libio_#* process for each drive within the library. AMASS maintains a sorted queue of write requests in which the *libio_#* process reads to determine which blocks are available to be flushed to the optical disk. Additionally, Oracle and AMASS have different schemas to guarantee the SGA or AMASS cached data is not lost during system anomalies.

Figure 1 illustrates the high level I/O flow between Oracle and AMASS. Figure 2 illustrates how data blocks are passed through AMASS cache, the UNIX VFS layer, the Oracle SGA and finally to the user.

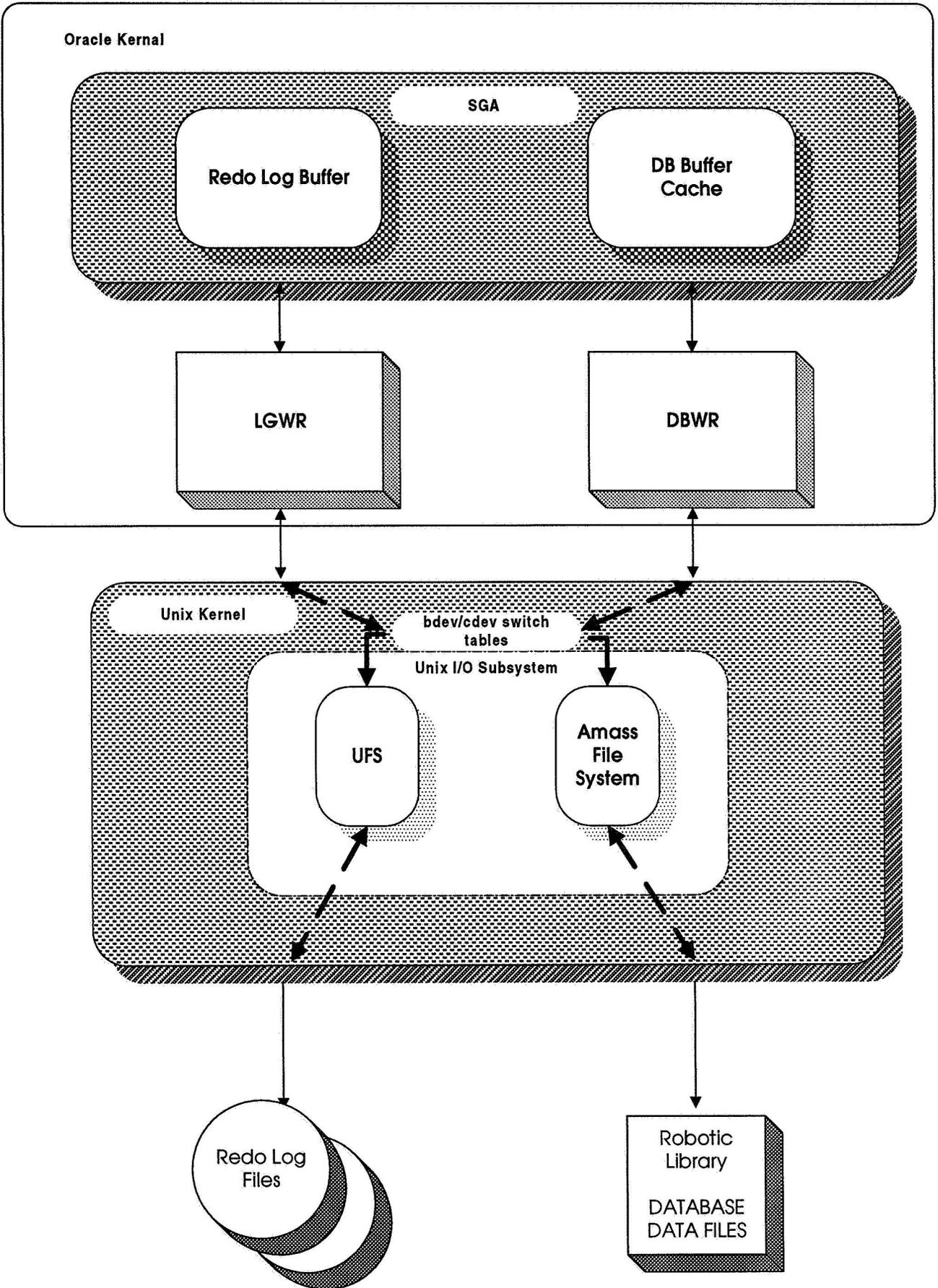


Figure 1: High-level Architecture I/O

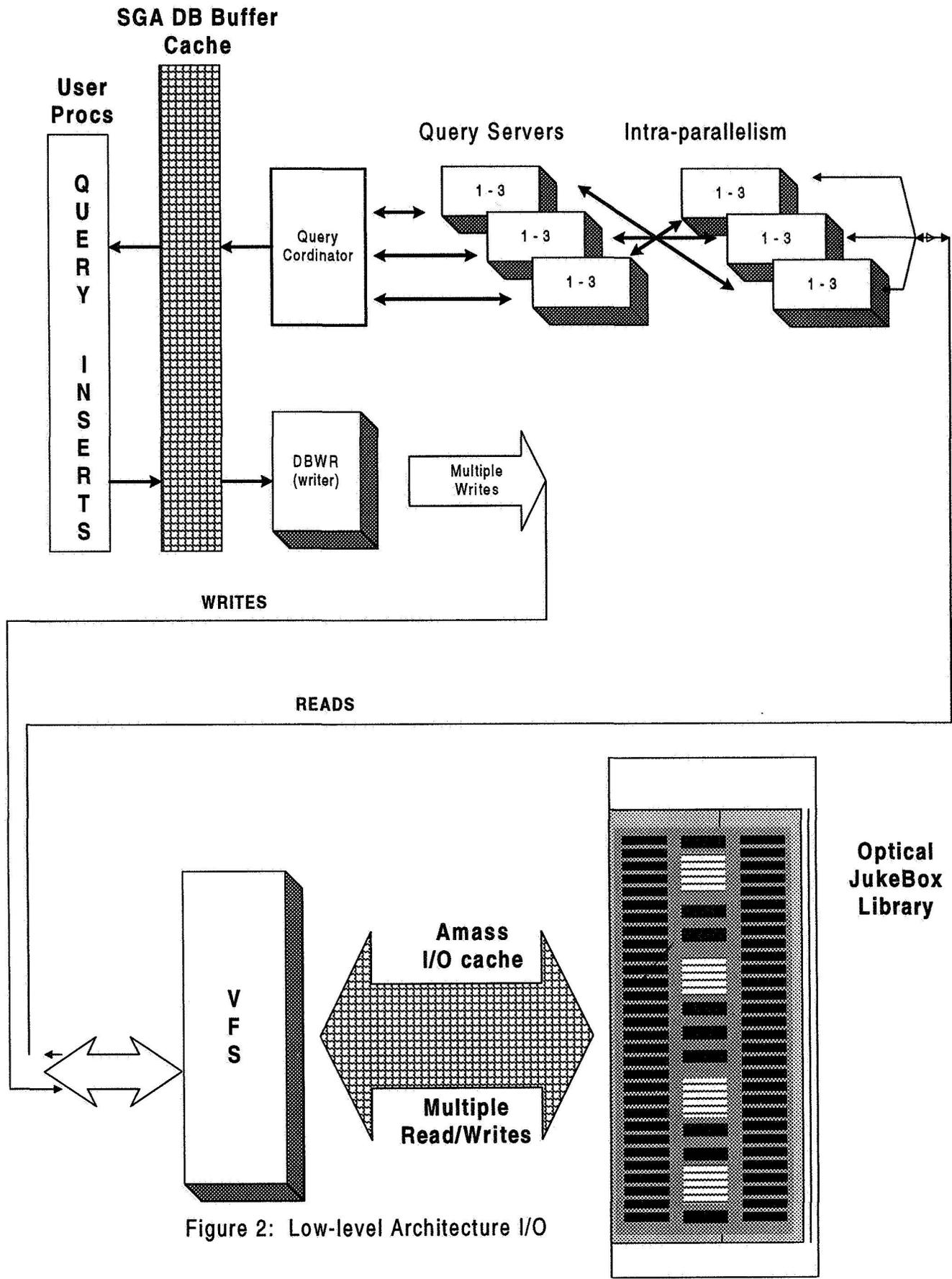


Figure 2: Low-level Architecture I/O

Prototype Proof of Concept

Prototype Environment

Configuration 1: Functionality Test

The first test was conducted on an SGI Challenge L Server. The server had 2 processors, 32 Megabytes of main memory and a 2 gigabyte disk drive sharing a controller with other drives. All the Oracle software, datafiles and redo logs were on the same disk drive. All monitoring and startups were initiated and processed on the server as opposed to a client.

Configuration 2: Performance Test

The second run was conducted on an SGI Challenge Server. The server had 4 processors, 756 Megabytes of main memory and six 2 gigabyte dual-headed barracuda disk drives striped across three controllers. All database monitoring and startup scripts were initiated from a client workstation.

Both configurations used the same optical disk system. The system consisted of eight 1.2 gigabyte drives, each one having its own SCSI ID. One controller was used for the optical disk system. The optical disks write at a rate of 300 KB/sec and can be read at a rate of 600 KB/sec. All transactions to and from the optical disk system are interfaced through an AMASS cache. The AMASS cache consisted of a 16 gigabyte RAID 5 system. The AMASS cache consists of 9 tunable cache blocks per open file/Oracle data file.

Both configurations used SGI's IRIX 5.3 UNIX operating system. The Oracle configuration consisted of Oracle7 release 7.2.2 of the DBMS, parallel query option and SQL*Net 2.2.

Testing

The database contained two identical tables, T_ACTIVE and T_ARCHIVE, four indexes (two per table) and a single view, V_MENU. The view was a union of a common set of columns from the tables T_ACTIVE and T_ARCHIVE. Appendix 1 contains the DDL for the creation of these objects. The software designed to test the archive concept was composed of three parts.

Part I) Loaded table T_ACTIVE, to a magnetic disk via a PL/SQL module with 10,000 rows, a sequence was used to ensure uniqueness. After the table was loaded, the PL/SQL code updated the inserted rows to randomly spread out the value of the column; 'changed_date.'

Part II) A C program selected a number of rows from T_ACTIVE that met a date condition on the 'changed_date' column. The rows that met the condition

were written to table T_ARCHIVE on an optical disk and its associated indexes were updated. The rows written to the optical disk were then deleted from T_ACTIVE. The tablespace that contains the archived table is usually maintained as a 'read-only' tablespace. Prior to running the second PL/SQL code, the tablespace is placed in 'write' mode. After the data to be archived was written, the tablespace was placed back in 'read only' mode.

Part III) Issue a query that selects rows from the view V_MENU. The query was designed to select rows from both T_ACTIVE and T_ARCHIVE.

Functionality Test

The first test used an Oracle configuration with 2 KB database blocks and all the default settings for the various database (init.ora) parameters. Minor tuning was conducted on the Oracle kernel to provide incremental improvements (i.e., increasing the number of rollback segments, etc.). The results of test one proved that the concept was viable, however, performance was unacceptable. The following test results are for the first test, a summary of the results for both tests are shown in Table 1.

Loading the table T_ACTIVE --	0:06.0 (read 6 minutes)
Migrating data to optical drives --	1:13.37
Querying both active and archived data --	0:01.13

The results of the functionality test precipitated discussions for the performance test. It was determined that the environment needed to be upgraded and tuned. The hardware architecture needed to be upgraded to increase main memory and add more disk drives. In addition, the Oracle database was not tuned on its initial installation (i.e., default settings were used) and significant improvement should be gained by altering a number of the adjustable parameters. The AMASS software and its associated cache have numerous adjustable parameters that were not fully utilized. It was determined that the performance test be conducted taking into account the following recommendations generated as a result of the functionality test:

- Rebuild the Oracle kernel with an increased block size
- Add memory to the SGI server and increase the number of database buffer blocks
- Separate indexes on the T_ARCHIVE table to a separate tablespace residing on a different optical disk
- Separate indexes on the T_ACTIVE table to a separate tablespace residing on a different magnetic disk
- Perform inserts with array processing
- Turn on read-ahead on the optical jukebox
- Tune the checkpoint interval on the database
- Have the SGI server dedicated to Oracle and not concurrently running the storage management software
- Initiate processing from a client workstation instead of a server

Performance Test

The performance test was conducted on a new set of hardware as described above and through a client workstation. The Oracle instance was rebuilt with a 16 KB block size and larger redo log files and a larger temporary tablespace. With additional magnetic disks, separate data and index tablespaces were created leaving the Oracle software libraries on their own magnetic device. On the optical jukebox, two tablespaces were created, one for the data and one for the indexes. The tablespaces resided on different optical platters and thus different drives.

The first run of the performance test used the same software as in the functionality test. In this test, T_ACTIVE was first loaded with 10,000 records and then with 100,000 records to test loading times. The times to load T_ACTIVE were as follows:

Load 10,000 rows -- 0:04.20

Load 100,000 rows --0:09.29

The initial PL/SQL script used to load table T_ACTIVE was modified into a C program with no array processing and one commit point. This load took approximately 1 minute and 2 seconds (0:01.02). The code was then further modified by removing the database's sequence processing function and by not performing the update to the 'changed_date' column. The load under this scenario took 30 seconds (0:00.30). Since the test required various 'changed_date' values, the section of code used to vary the data in this column was replaced with Oracle's DECODE process. The resulting load with this modification took 50 seconds (0:00.50). The same modified code was used to load 100,000 records, this run took 6 minutes and 59 seconds (0:06.59). In terms of time, the two different loads were essentially linear, actually, the 100,000 record load was a bit more efficient than the smaller load. Comparing the 10,000 record loads between the functionality and performance test, the performance test showed an 86% improvement in load time. This improvement was attributed to application tuning and the hardware upgrade. Since the goal of this testing was to determine the feasibility of migrating data to optical disks, we accepted our loading results and shifted our attention to the migration process and did not revisit the loading process. The last load of T_ACTIVE was with 100,000 records. It is with this load that we tested the migration process. The results of the load portion of the performance test is as follows and an overall summary can be found in Table 1:

Load of 10,000 rows -- 0:00.50

Load of 100,000 rows -- 0:06.59

The migration PL/SQL code was run unmodified (i.e., same code as the functionality test) and took 44 minutes and 42 seconds (0:44.42). Instead of half the number of rows being migrated as in the functionality test, only a third of the records in the performance test were migrated. This difference was due to how we modified the loading script. It should be noted however, that we migrated 33,334 rows in the performance test vice the 4997 rows in the functionality test -- the results of this migration in itself was a significant improvement time per the number of rows loaded as shown below:

Functionality Test Migration -- 67 rows/minute
Performance Test Migration -- 750 rows/minute

The AMASS cache block was sized at 1 MB. Analysis of I/O monitoring suggested an increase of the AMASS cache block. Enlarging the block size should cause fewer writes to the optical disk system. A size of 64 MB was determined to be sufficient to continue testing. The migration process was rerun taking 5 minutes and 50 seconds to migrate 33,334 rows to the optical disk system. An 87% improvement from the previous run resulted.

Migrate 33,334 rows with 1 MB AMASS cache block -- 0:44.42
Migrate 33,334 rows with 64 MB AMASS cache block -- 0:05.50 (5718 rows/minute)

Considering the modifications to the Oracle kernel and the AMASS cache, the migration results have improved substantially. The Oracle Trace and TKPROF utilities were then utilized to fine tune the process. Because the migration PL/SQL code included two select statements, both tables were altered to be parallelized with a degree of 8. The migration code was rerun with this modification with a resultant time of 5 minutes and 29 seconds (0:05.29). Reviewing the output of the TKPROF process, it was found that the temp tablespace was heavily utilized. As a result, we increased the Sort Size parameter to 10 MB and added 16 batch writes to the init.ora parameter file. The migration process was rerun with the resultant time of 5 minutes and 1 second (0:05.01).

Further investigation of the TKPROF output revealed that the 'SELECT COUNT(*) FROM V_MENU' SQL statements took on average 1 minute and 42 seconds to run. This SQL statement was run at the start and end of the migration process. This means that approximately 3 minutes and 24 seconds or 67% of the migration time was devoted to counting the number of rows in T_ACTIVE and T_ARCHIVE. This SQL statement has a GROUP BY clause which is part of the view and not the physical table and thus results in two full table scans and does not employ any indexes. Therefore, due to the inordinate amount of time and processing that was consumed when querying the view, we replaced the provided SQL statement with the following SQL statement:

```
select count (*), "Online"  
from T_ACTIVE  
UNION  
select count(*), "Archive"  
from T_ARCHIVE
```

The migration process was rerun with a resultant time of 1 minute and 51 seconds (0:01.51). The modification of the migration process produced another 63% improvement in run time. All of this was attributed to the two queries, before and after the migration, took a total of .5 seconds vice 3 minutes and 24 seconds. The final breakdown, in time, of the entire migration process is as follows:

Alter Tablespace to read write --	.13 seconds
SELECT COUNT(*) --	.25 seconds
Select data to extract from T_ACTIVE --	29 seconds
Insert extracted data into T_ARCHIVE --	38 seconds
Delete extracted data from T_ACTIVE --	40 seconds
Alter Tablespace to read-only --	.13 seconds
SELECT COUNT(*) --	.25 seconds

Results

Reviewing the whole migration process, it was determined, that the migration from magnetic media to the optical media actually takes place when the optical media's tablespace is altered to read-only. When this occurs, the dirty pages in the SGA are written to the AMASS cache. At this point, the data is now archived. The actual write to the optical disk takes place when four of the nine AMASS cache blocks are full.

The results of the performance test were much more conclusive. The improved hardware and the extensive tuning of the Oracle kernel, AMASS system and the test software proved to be essential modifications for the test.

Table 1: Migration Test Results

Operation	Functionality Test	Performance Test
Load of On-Line data table (10,000 rows)	6 minutes	50 seconds
Migration of 4997 rows from On-Line to Near-Line (optical) and deletion from the On-Line device	1 hour 13 minutes 37 seconds	1 minute 51 seconds * migrated 33334 rows
Query of view (union of both On-Line and Near-Line tables) -- After instance reboot	1 minute 13 seconds	N/A (see Note below)
-- Data cached	20 seconds	N/A
-- Flushed cache	34 seconds	N/A

Note: The third part of the test (query using view V_MENU) was not conducted during the performance test. It was determined in the functionality test, that data on optical disk can be queried and subsequently selected along with data from magnetic media.

Considerations for employing the use of Long-Term Archival Technology

(1) Write efficient application code and tune it for high performance!

Probably the most important aspect of the archive function is that the application and associated SQL and PL/SQL code be as efficiently written as possible. During the development phase, the SQL and PL/SQL should be subjected to numerous tuning exercises under varying conditions. Proper access to the database significantly reduces the run-time of the archival process as was indicated in testing.

Utilize the array interface when inserting a large number of rows into tables. Use to your advantage the decode statement to provide a way to avoid having to scan the same rows repetitively, or to join the same table repetitively.

After the code has been written, use the utilities provided by Oracle. Run the ANALYZE command on the tables you'll be querying. The ANALYZE command collects statistics about the tables and stores them in the data dictionary. Determine if the optimizer is selecting the most efficient access path for your SQL statements by running EXPLAIN PLAN. The EXPLAIN PLAN diagnostic statement gives you an inside look at how the optimizer is planning to process your SQL statement. The results of the analysis may provide the impetus to use hints in your SQL statements. Hints are a mechanism allowing you to manually tune individual SQL statements, overriding the optimizer's decisions for that statement by including your own optimization hints within the SQL statement.

Additionally, through the use of the parallel query option, you are able to scan intensive queries in a parallel fashion. Set the degree of parallelism close to the total number of disk heads on the drives containing the datafiles of the tables and indexes you are using during the archive process.

(2) Strategic placement of database objects during physical database design!

It is critical that the database objects (tables and indexes) associated with the archived data be placed on separate tablespaces. The tablespaces, if possible, should be accessed from different controllers. In addition, separate tablespaces for tables and indexes reduces the contention during the insertion of records and associated indexes. To further reduce contention, stripe the datafiles across multiple disks and controllers.

(3) Tune the Oracle kernel (init.ora) for optimal performance!

The Oracle kernel must be tuned to take advantage of the inherent tunable features of the Oracle database. The first parameter to be set should be the DB_BLOCK_SIZE. This parameter must be set prior to installing the database. For the performance test, we set this parameter to 16KB, we found that performance was enhanced when this was reset from its initial setting of 2KB. The following parameters should be set in the init.ora file:

OPTIMIZER_MODE -- COST

The OPTIMIZER_MODE parameter tells the query optimizer, when set to COST, to use the cost based optimizer. If you are selecting more than 10% of your data, it is optimal for Oracle to use a full table scan to satisfy NCUNITS is an integer used in combination with the MAXIOSZ parameter to define the AMASS cache block size. The cache block size is determined by the following equation:
your query. Couple this with Oracle's parallel query option, significant performance improvements will be gained.

SORT_AREA_SIZE -- 10 MB

The size in bytes a user process has available for sorting. Performance improvement can be substantial. Allocated on a per user basis.

ASYNC_IO -- TRUE

This will allow parallel disk writes and have the potential of increasing performance 20%.

DB_WRITERS -- 8

Ability to perform multiple database writes. At a checkpoint the master database writer determines which blocks in the database buffer cache need to be written to disk. The master database writer divides up the work and notifies slave database writers to write blocks. A good value would be equal to the number of disks containing data files.

DB_BLOCK_WRITE_BATCH -- 16

The number of blocks a database writer passes at one time to the operating system to write to different disks in parallel and to write adjacent blocks in a single I/O. A good value would be equal to twice the number of DB_WRITERS.

(4) Tune the AMASS cache and associated system components for optimal performance!

Significant performance gains will be achieved by properly tuning the AMASS cache. AMASS has configurable options broken into the following four areas: cache configuration, performance, jukebox scheduling and miscellaneous. In the test of the archive concept, we tuned parameters that affected the cache and performance areas. Tuning in the cache area proved to be the most beneficial for the purposes of this test.

MAXIOSZ maximum input/output size

MAXIOSZ is the size, in bytes, that AMASS uses internally to read data from and write data to the optical media. It is recommended that MAXIOSZ be kept as large as possible to achieve maximum throughput rates. Initially, this parameter

was set to 1 MB and remained so throughout the test. This parameter is derived by the O/S vendors implementation of their scsi device driver.

NCUNITS number of cache units

NCUNITS is an integer used in combination with the MAXIOSZ parameter to define the AMASS cache block size. The cache block size is determined by the following equation:

$$\text{Cache Block Size} = \text{NCUNITS} * \text{MAXIOSZ}$$

The cache block is the basic unit used by AMASS to transfer data to and from the cache. When a file is read from or written to the AMASS file system media, it passes through the cache in cache block sized packets.

The initial test, had this parameter set to 1. It was subsequently set and maintained at 64. We found that a 64 MB cache block size increased the performance of the transfer to the optical disk .

NFNODES number of fnodes (file nodes)

The NFNODES parameter defines the number of files that can be open concurrently in the AMASS file system. This parameter is automatically calculated during AMASS configuration based on the number of cache blocks available in the cache disk.

The performance parameters define whether or not multiple cache blocks are read (READAHEAD) as a file is read and if more than one volume is a volume group can be written to at a time. For our purposes, we modified the READAHEAD parameter.

READAHEAD file read-ahead

The READAHEAD parameter is either set to enabled (1) or disabled (0). When enabled, AMASS will automatically read the requested block along with an additional three cache blocks of data from the AMASS file system media with every read request.

(5) Plan the periodicity of the archival process!

The timing of the archival process should coincide during low level activities of the production system. If numerous tables are be queried and subsequently deleted from, this will result in an increased load to the production system. Logic calls for query and modification type activities of a batch nature to be conducted during relatively quiet periods of system inactivity.

(6) Sizing of the Jukebox

The I/O bottleneck is directly related to the number of optical devices within your jukebox as related to Oracle users requests that do not have data in the SGA or the AMASS cache. Observed rates of 600 KB/sec on reads and 300 KB/sec on writes were observed and expected.

Theoretical Considerations

The implementation of optical media in an Oracle database environment, to the knowledge of the authors, has never been attempted in a production environment. Therefore, it is necessary to test and evaluate various database activities within this new environment to verify all database functionality and administration features operate as they do in an all magnetic media environment. The customer proof of concept we conducted did not have the time to do a thorough investigation of all the different possible scenarios a typical IS organization may use. Following are some of the areas that need further investigation to fully ensure the viability of using long-term archival storage media.

System Startup

The Oracle control file contains the names and locations of the datafiles which are opened and checked for integrity. Upon startup, Oracle opens and checks each datafile associated with a tablespace for database instance integrity. Therefore, for each tablespace space created, Oracle will cause the optical media to be loaded into the MO drive and verify the file's tablespace header information. Startup times will be slower the traditional Oracle disk instance.

Archive Considerations

Select & Insert (DML): An Oracle implementation with optical disks is best suited for transactions that do queries and initial loading of data with inserts. Data inserted onto optical disks should be archived data which is typically static and will not be altered in the future. Queries on the archived data will successfully transfer data at the expected read rates of the optical media, 600 KB/sec. Inserts exceeds performance requirements due to the AMASS cache and write algorithms implemented within the AMASS cache I/O architecture.

Update (DML): AMASS does not handle dynamic column/row size updates the same way as a disk device. In archive architectures the media is formatted and the archive software can not update a used block on the media. The process for an update of a data block is to copy the block into memory, mark the current media block as unusable, in memory update the block with new results, then write the new memory block(s) to the optical media. The only case where a block is not marked invalid is when an update does not increase or decrease the data within a block.

Oracle database block **row chaining** and **row migration** should be avoided or further dead space issues on media will result.

Performance

Writes are not limited by Oracle and AMASS, but enhanced, since Oracle allocates contiguous data blocks within an extent. AMASS writes will be done contiguously since the creation of the tablespace within Oracle opens the datafile location(s) and allocates a contiguous set of blocks on media.

The AMASS I/O cache and Oracle database buffer cache both use an LRU algorithm which does not contend with the I/O among both software kernels. Sizing of the database buffer cache (shared memory) and AMASS cache (rdsk) are related when tuning and sizing the architectures to meet system throughput requirements.

Matching AMASS cache block size to Oracle's database block size did not significantly improve performance, rather having a larger cache block sizes proved I/O gains for the large insert transactions and full table scan queries. This results due to the manner in which AMASS handles its cache blocks for I/O.

Read times can be enhanced within Oracle by setting the initialization parameter, `DB_FILE_MULTIBLOCK_READ_COUNT`, to match your queries. Additionally, the AMASS architecture was tuned for read-ahead.

Archive Technologies

The AMASS I/O architecture reads and writes data in blocks. Other archive architectures which use block I/O may work as well. Archive architectures that perform I/O using file format **may not** provide the same ease of use, ease of administration and good media space management.

Indexes

Depending on queries and mechanical robotic movement, the use of indexes may reduce performance significantly. Using full table scans may be the correct approach. Further testing is necessary in this area. Functionally, indexes were proven to work. In our testing, and by the advice of database tuners, indexes should be created and placed on the same optical disks as its base table. Separation of indexes spreads the I/O request across media and removes media thrashing caused by having two data structures on the same media. Another alternative is to create the archived table's indices on magnetic media. This should result in improved performance by removing the extra robotic movement time of loading the index tablespace in the drive.

Tape Technology

The use of tape technology was not tested but needs to be discussed. Optical testing was performed due to the customer's media life expectancy requirement. It is possible though, that tape technology could be used as the archival medium. Some of the factors that need to be considered and tested are as follows:

- The effect of Oracle startup and the effects of reading the header of each tablespace during the database integrity start up phase.
- Tape thrashing. Accessing a set of tables randomly, e.g., first table at end of tape, second in middle of tape. Additionally, user programming may be necessary when accessing table data in order to gain acceptable query and I/O transfer rates. User applications should consider querying tables in sequential order as they were written to tape.
- Use of indexes may not be beneficial at all, indexes would need to reside on disk.

Robotic Hardware

Jukebox sizing is based totally on the user's requirements. Some of the hardware considerations are as follows:

- Response time
- Simultaneous insert transactions
- Simultaneous query transactions
- Off-line media management acceptability

Conclusion

This exercise *proved the concept* of migrating data from magnetic devices to optical devices. Not only is it possible to do so, but performance-wise it is feasible. It was found that properly tuning the Oracle kernel, the AMASS cache system, system memory and I/O and the application code, the actual migration process is essentially a function of selecting requested data, inserting this data into another table and then deleting selected rows from the table on the magnetic media.

The prototype architecture, as shown in the proof of concept, provides the means to architect VLDB cost-effectively by using disk and optical storage devices using COTS products. This architecture should allow businesses to respond to customer needs by maintaining information on-line and near-online and having access to their data in a transparent manner.

Acknowledgments

Vanguard Technologies: Eric Eastman and Dave Donald
11211 East Arapahoe Road
Englewood, CO
800-840-6090
Lab Environment and Optical Library
AMASS Documentation Suite

Silicon Graphics Inc.: Liz Reynolds and Fred Beck
Denver, CO
Challenge L hardware

Emass Incorporated
10949 East Peakview Ave
Englewood, CO
800-654-6277
AMASS Software

Oracle Corporation: Joe Conway
Advanced Programs Group

References

Oracle 7.2 Server Tuning
Oracle7 Server Concepts
Open Data Warehousing with Oracle 7 Parallel Data Management Technology

Appendix 1

```
CREATE TABLE T_ACTIVE
(
MENU_NUM      NUMBER(9)
              CONSTRAINT CARMENUC_NUM_NN NOT NULL,
MENU_LABEL    VARCHAR2(2000)
              CONSTRAINT CARMENUC_MENU_LABEL_NN NOT NULL,
ADDED_USER_ID VARCHAR2(15)
              DEFAULT SUBSTR(USER,1,15)
              CONSTRAINT CARMENUC_ADDED_USER_ID_NN NOT NULL,
ADDED_DATE    DATE
              CONSTRAINT CARMENUC_ADDED_DATE_NN NOT NULL,
CHANGED_USER_ID  VARCHAR2(15)
              DEFAULT SUBSTR(USER,1,15)
              CONSTRAINT CARMENUC_CHANGED_USER_ID_NN NOT NULL,
CHANGED_DATE   DATE
              DEFAULT SYSDATE
              CONSTRAINT CARMENUC_CHANGED_DATE_NN NOT NULL,
CONSTRAINT    IAMENU0
              PRIMARY KEY    (MENU_NUM)
              USING INDEX PCTFREE 5
              TABLESPACE ONLINE
              STORAGE (
                INITIAL 150K
                NEXT 150K
                PCTINCREASE 0
              )
)
```

```
CREATE INDEX IACTIVE_CHANGED_DATE ON T_ACTIVE
(CHANGED_DATE)
TABLESPACE AINDEX
STORAGE (
  INITIAL 150K
  NEXT 150K
  PCTINCREASE 0
)
```

```
CREATE TABLE T_ARCHIVE
(
MENU_NUM      NUMBER(9)
              CONSTRAINT CARMENUC_NUM_NN NOT NULL,
```

```

MENU_LABEL    VARCHAR2(2000)
              CONSTRAINT CARMENUC_MENU_LABEL_NN NOT NULL,
ADDED_USER_ID VARCHAR2(15)
              DEFAULT SUBSTR(USER,1,15)
              CONSTRAINT CARMENUC_ADDED_USER_ID_NN NOT NULL,
ADDED_DATE    DATE
              CONSTRAINT CARMENUC_ADDED_DATE_NN NOT NULL,
CHANGED_USER_ID  VARCHAR2(15)
              DEFAULT SUBSTR(USER,1,15)
              CONSTRAINT CARMENUC_CHANGED_USER_ID_NN NOT NULL,
CHANGED_DATE    DATE
              DEFAULT SYSDATE
              CONSTRAINT CARMENUC_CHANGED_DATE_NN NOT NULL,
CONSTRAINT     IARMENUU0
              PRIMARY KEY    (MENU_NUM)
              USING INDEX PCTFREE 5
              TABLESPACE ARCHIVE
              STORAGE (
                INITIAL 150K
                NEXT 150K
                PCTINCREASE 0
              )
)

```

```

CREATE INDEX IARCHIVE_CHANGED_DATE ON T_ARCHIVE
(CHANGED_DATE)
TABLESPACE ARCHINDEX
STORAGE (
  INITIAL 150K
  NEXT 150K
  PCTINCREASE 0
)

```

Design and Implementation of Scalable Tape Archiver

Toshihiro Nemoto, Masaru Kitsuregawa, Mikio Takagi

Institute of Industrial Science, University of Tokyo

7-22-1, Roppongi, Minato-ku, Tokyo, Japan

{nemoto,kitsure,takagi}@tkl.iis.u-tokyo.ac.jp

Tel: +81-3-3402-6231

Fax: +81-3-3423-2834

5/6-82
83/98

Abstract

In order to reduce costs, computer manufacturers try to use commodity parts as much as possible. Mainframes using proprietary processors are being replaced by high performance RISC microprocessor-based workstations, which are further being replaced by the commodity microprocessor used in personal computers. Highly reliable disks for mainframes are also being replaced by disk arrays, which are complexes of disk drives.

In this paper we try to clarify the feasibility of a large scale tertiary storage system composed of 8-mm tape archivers utilizing robotics. In the near future, the 8-mm tape archiver will be widely used and become a commodity part, since recent rapid growth of multimedia applications requires much larger storage than disk drives can provide. We designed a scalable tape archiver which connects as many 8-mm tape archivers (element archivers) as possible. In the scalable archiver, robotics can exchange a cassette tape between two adjacent element archivers mechanically. Thus, we can build a large scalable archiver inexpensively. In addition, a sophisticated migration mechanism distributes frequently accessed tapes (hot tapes) evenly among all of the element archivers, which improves the throughput considerably. Even with the failures of some tape drives, the system dynamically redistributes hot tapes to the other element archivers which have live tape drives. Several kinds of specially tailored huge archivers are on the market, however, the 8mm tape scalable archiver could replace them.

To maintain high performance in spite of high access locality when a large number of archivers are attached to the scalable archiver, it is necessary to scatter frequently accessed cassettes among the element archivers and to use the tape drives efficiently. For this purpose, we introduce two cassette migration algorithms, foreground migration and background migration. Foreground migration transfers a requested cassette from an element archiver whose drives are busy to another element archiver whose drives are idle. Background migration transfers cassettes between element archivers to redistribute frequently accessed cassettes, thus balancing the load of each archiver. Background migration occurs the robotics are idle. Both migration algorithms are based on access frequency and space utility of each element archiver. To normalize these parameters according to the number of drives in each element archiver, it is possible to maintain high performance even if some tape drives fail. We found that the foreground migration is efficient at reducing access response time. Beside the foreground migration, the background migration makes it possible to track the transition of spatial access locality quickly.

I. Introduction

Recently, large scale tertiary storage systems are becoming more and more desired for multimedia applications or scientific data such as satellite images. Today, magnetic disks have become cheap with large capacity, however, this capacity is still not enough to archive multimedia data or satellite images. To archive huge data sets, magnetic tape archivers which have some tape drives and robotics for management of the tapes in them, are often used, but most of the current commercial tape archivers are not scalable and there is no way to migrate data from one archiver to another except by copying the data. Accordingly it takes a long time to redistribute data.

To address these issues and aiming towards scalable commodity archivers, we have been developing a scalable tape archiver for satellite images. It consists of some commodity element archivers and tape migration units between two adjacent element archivers. We believe a reasonable size tape robotics will become a commodity component in the near future. It is easy to add or remove element archivers to the scalable tape archiver at any time and any number of element archivers can be attached. To redistribute data, a cassette is transferred from one element archiver to another through the tape migration unit instead of copying data.

In this paper, we present a cassette migration mechanism for the scalable archiver and its performance evaluation. The scientific data such as satellite images are characterized not only by their size but also by access locality. Accordingly, when storing these data, efficient utilization of the tape drives and the proper positioning of frequently accessed cassettes substantially affects the performance. In order to achieve high performance in spite of changing access locality, two load balancing mechanisms, foreground migration and background migration, are introduced to the scalable tape archiver. The foreground migration transfers a requested cassette from an element archiver whose drives are busy to another element archiver with idle drives. Background migration transfers a cassette to redistribute frequently accessed cassettes between idle element archivers. The foreground migration is efficient at reducing access response time. Beside the foreground migration, the background migration makes it possible to track the transition of spatial access locality quickly.

I. Design of The Scalable Tape Archiver

The scalable tape archiver is composed of any number of small size tape archivers (element archivers) and cassette migration units connecting any two adjacent element archivers. Figure 1 shows the organization of the experimental scalable tape archiver using an 8mm tape jukebox, NTH-200B, as the element archiver. The NTH-200B has two Exabyte 8505 tape drives, a tape handler robot and a cassette rack with 200 slots. It also has a controller for its own tape handler robot and for the tape migration unit on its right. The host computer sends commands for holding, releasing and moving a tape and so on to the controller and receives the status of the element archiver through an RS-232C port. The tape handler robot takes a cassette from a slot in the rack or from the drives and places it in another slot or drive according to the command received. The tape drives are normal Exabyte 8505's and are connected to the host computer through a SCSI bus. The tape migration unit has a wagon to migrate a cassette tape to another element archiver. Cassette tape migration is executed as follows.

1. The tape migration unit brings the wagon back into the source element archiver, if the wagon is not currently in the source element archiver.
2. The tape handler robot in the source element archiver takes the cassette to migrate from a slot or a drive.
3. The tape handler robot places the cassette in the tape migration unit's wagon.
4. The tape migration unit sends the wagon from the source element archiver to the destination element archiver.
5. The tape handler robot in the destination element archiver picks up the cassette tape from the wagon, and places the tape into the appropriate slot or drive.

These steps are coordinated so that the counterweight of the tape handler robot does not interfere with the movements of the tape migration unit.

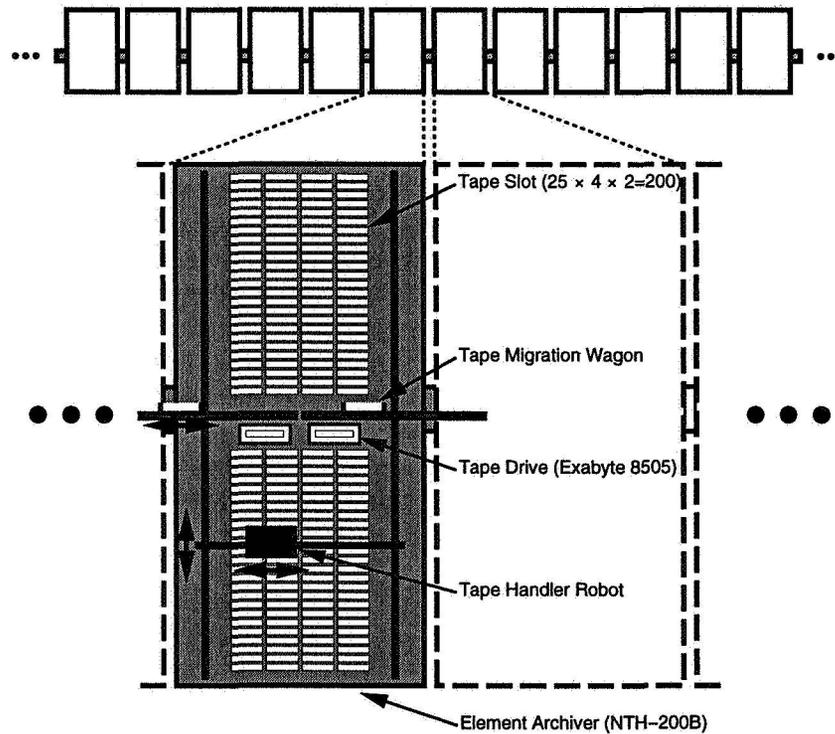


Figure 1: Organization of Experimental Scalable Tape Archiver using NTH-200B

I. Cassette Migration Strategy

A. Access locality

First, we describe the heat and temperature metrics [1]. The heat is the access frequency of a cassette or an archiver over some period of time. The heat of a cassette is the sum of its access frequencies and the heat of an archiver is the accumulated heat of the cassettes in it. Temperature is defined as the heat of a cassette or as the heat of the archiver divided by the number of tapes it contains.

High access locality hinders the efficient use of the archivers. If hot cassettes are concentrated on a few element archivers, the hot element archivers may receive too many

tape access requests leaving the cold element archivers idle. To reduce the concentration of accesses and to improve efficient use of the resources, it is necessary to scatter the frequently accessed cassettes around the scalable tape archiver. For this purpose, two load balancing mechanisms, foreground migration and background migration, are introduced to the scalable tape archiver.

A. Foreground migration

When a new access request is issued for a tape in an element archiver where all drives are currently in use, moving the tape to another element archiver and using a free drive can reduce the response time of the request. We call such migration foreground migration. When there are several element archivers which can accept a new cassette, that is, which have one or more empty slots and idle drives, and whose tape handler robot is idle, the following four strategies are used to select which element archivers to migrate the cassette to.

- **Random:** In the random strategy, the destination element archiver of the migrated cassette is selected at random.
- **Space Balancing:** When an element archiver is full, it cannot accept a new cassette to migrate into even if it has idle drives. Therefore, the destination to migrate is the element archiver in which the number of cassettes is smallest in the space balancing strategy.
- **Heat Balancing:** Selecting an element archiver whose heat is lowest to migrate a cassette to balance the heat of each element archiver.
- **Distance Minimizing:** The nearest element archiver is selected as the destination of migrated cassette. The intermediate element archivers between source and destination element archivers cannot serve any request while they relay the migrated cassette.

A. Background migration

When it is possible to migrate a cassette between two element archivers, that is, when all of the tape handler robots and migration units between the source and destination element archivers are idle, migrating a cassette can balance the number of cassettes or heat of each element archiver. We call such migration background migration. In background migration, the cassette is always migrated from the element archiver which has more cassettes to the one holding fewer cassettes. A migrated cassette is selected to balance the element archivers the most. For example, a hot cassette is selected for migration when the heat of the source element archiver is larger than the destination's and a cold one is selected in the opposite case. When more than two background migration can be executed at the same time, the following two basic strategies are used to determine the source and destination archivers.

- **Space Emphasizing:** The space difference minimizing strategy selects the pair of element archivers whose number of cassettes differs the most.
- **Heat Emphasizing:** The heat difference minimizing strategy selects the pair of element archivers whose heat difference is largest.

I. Performance Evaluation

A. Description of the simulation

To evaluate the basic performance of the scalable tape archiver, we execute computer simulations to measure average response time. The simulation parameters shown in Table 1 are based on the real scalable tape archivers using the NTH-200B. We assume that each cassette tape has fifty data in it and the size of each data is 100 MB. Accordingly, the read/write time of one data is always 200 seconds. The minimal cycle time is 487 seconds¹ on average. The interval time of request arrival depends on a negative exponential distribution. Because the destination element archiver should have a vacant slot for the migrated cassette, we selected 95% as the load factor. The scalable tape archiver consists of sixteen element archivers and the initial distribution of the cassette tapes in the scalable tape archiver is shown in Table 2. The access locality follows an 80/20 rule, that is 80% of the accesses are to 20% of the cassettes.

A. Simulation results

Figure 2 shows the average response time after 50,000 accesses from the initial cassette distribution. Compared to the result of no migration, response time is significantly reduced when only foreground migration is introduced into the scalable tape archiver. Furthermore, using background migration can produce in addition more improvement. Figure 3 shows the average response time at intervals of even 2,000 accesses where the request arrival rate is 0.045 requests per second. Between the two background migration strategies, there is no difference, but using background migration makes it possible to track the changing of access locality quickly.

Table 1: Simulation Parameters

Element archiver	
Number of element archivers	16
Maximum number of cassettes in an element archiver	200
Number of drives in an element archiver	2
Drive	
Drive setup time	35sec
Seek speed	25 MB/sec
Read/Write speed	0.5 MB/sec
Tape eject time	20 sec
Tape handler robot and tape migration unit	
Robot move time	2 sec
Robot move time with holding and placing cassette	14 sec
Wagon unit move time	9 sec

¹ Robot move time + robot move time with holding and placing cassette + drive setup time + average seek time + read/write time + average seek time (for rewinding) + tape eject time + robot move time + robot move time with holding and placing cassette

Table 2: Initial cassette tape distribution

Elem. Archiver No	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Hot Cassettes	8	8	8	8	8	88	88	88	88	88	88	8	8	8	8
Cold Cassettes	182	182	182	182	182	102	102	102	102	102	102	182	182	182	182
Total	190	190	190	190	190	190	190	190	190	190	190	190	190	190	190

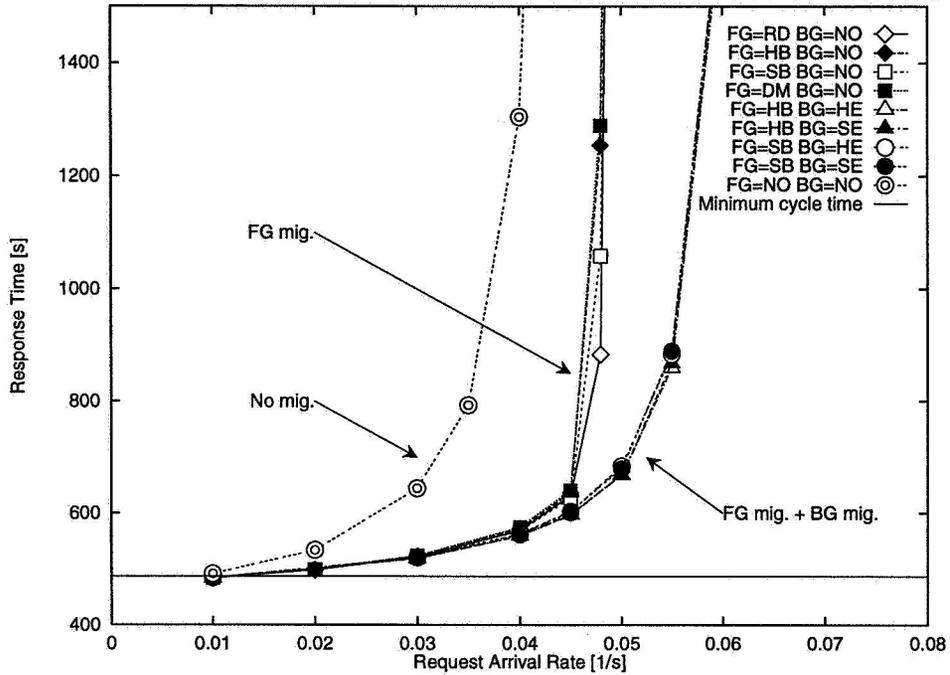


Figure 2: Average response time of initial 50,000 accesses

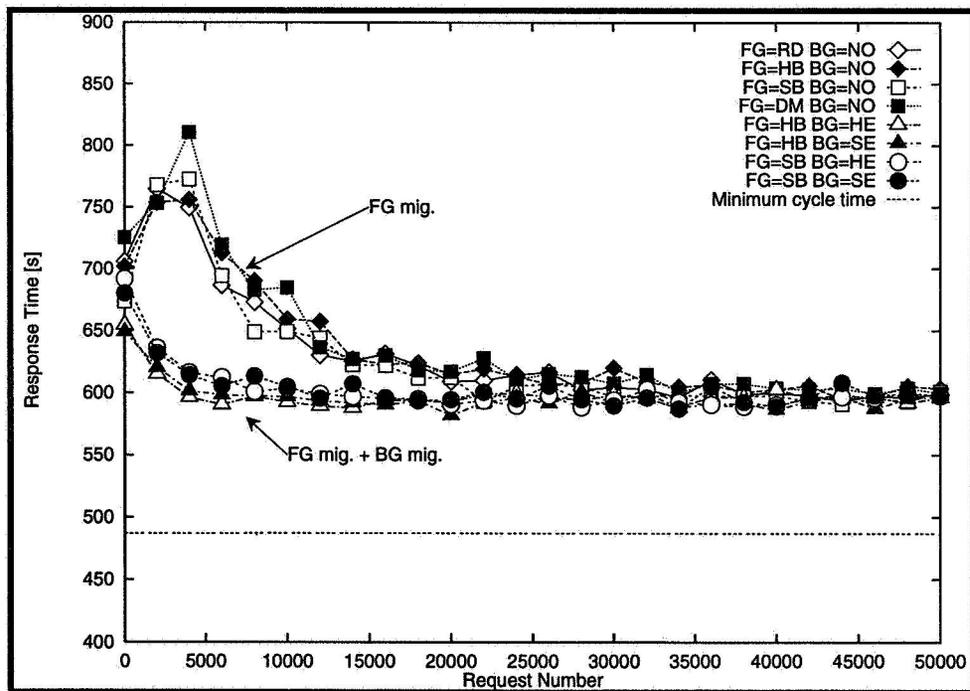


Figure 3: Average response time for intervals of 2,000 accesses

Figure 4 shows the average response time after 50,000 accesses from the initial cassette tape distribution when all tape migration wagons in the scalable tape archiver move slower. It takes 30 second to move the slower wagon from an element archiver to another, while it takes 9 second to move our experimental scalable tape archiver's. The cassette tape migration achieves better performance than using no migration even if the tape migration wagons are slower. The slower tape migration units do not deteriorate the performance very much. Therefore it is not necessary to use expensive high speed tape migration units. To connect some inexpensive small size commercial tape archivers with low cost tape migration mechanisms improves the performance significantly.

Figure 5 shows the average response time at intervals of 2,000 accesses from the initial state. In this simulation, a drive in the eighth element archiver fails when the scalable tape archiver receives 10,000 requests and it recovers after receiving 20,000 more requests. Figure 6 also shows the average response time at intervals of 2,000 accesses from initial state where both drives in the eighth element archiver fail and recover. The heat balancing strategy and the space emphasizing strategy are selected as foreground migration and background migration respectively. The average response time of the scalable tape archiver is not affected by the single drive failure significantly. Two drives failure deteriorates the average response time when the request arrival rate is 0.055. However, the scalable tape archiver can serve requests for the tape in the eighth element archiver, which has no drive in it, while ordinary archivers do not work in this situation.

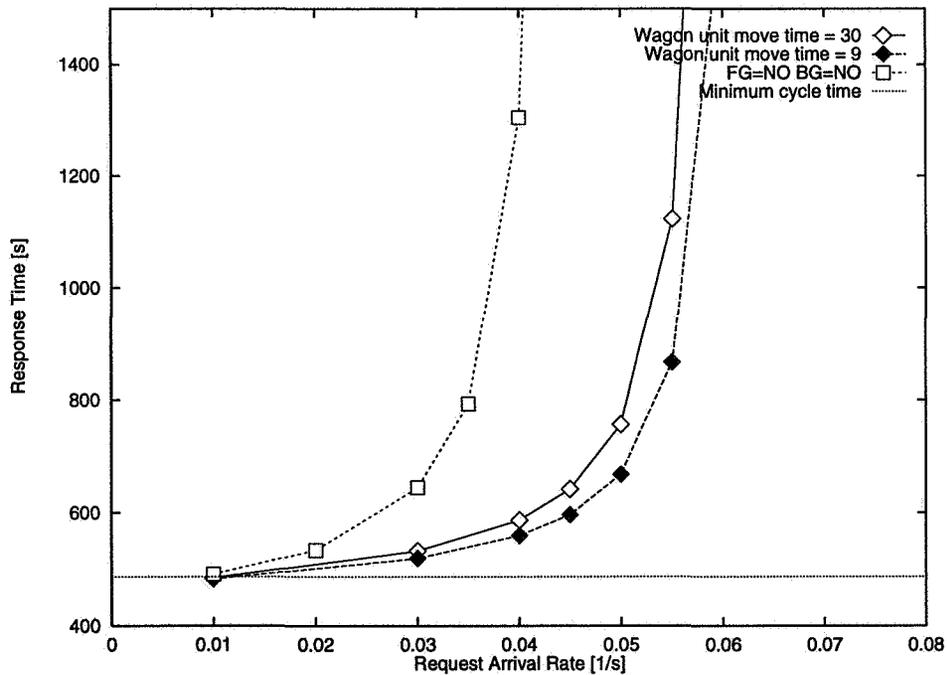


Figure 4: Average response time of slow migration wagon archiver

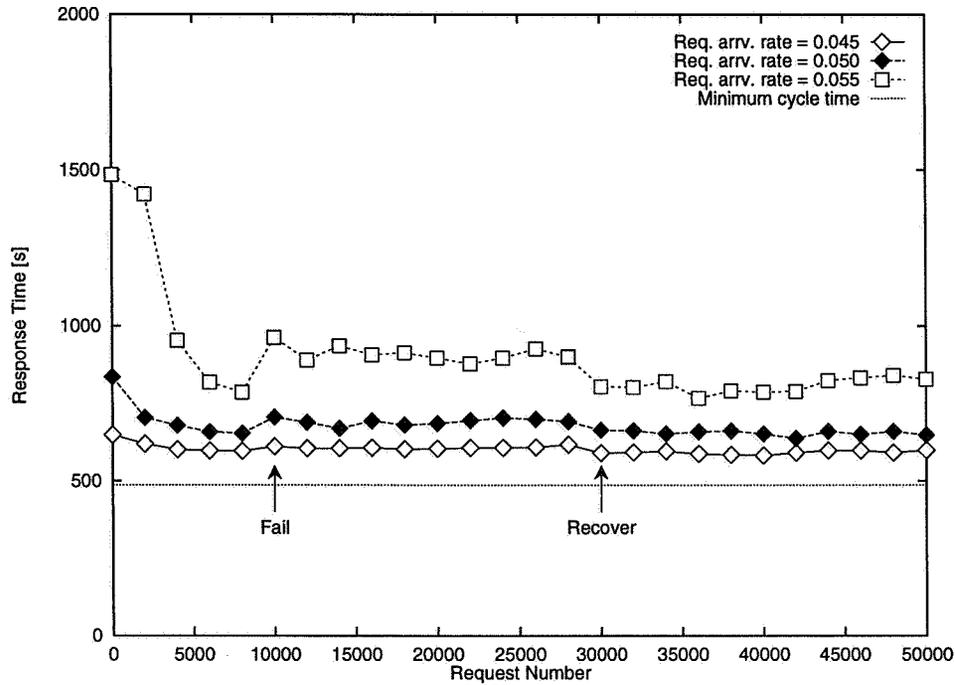


Figure 5: Average response time measured at intervals of 2,000 requests with single drive failure

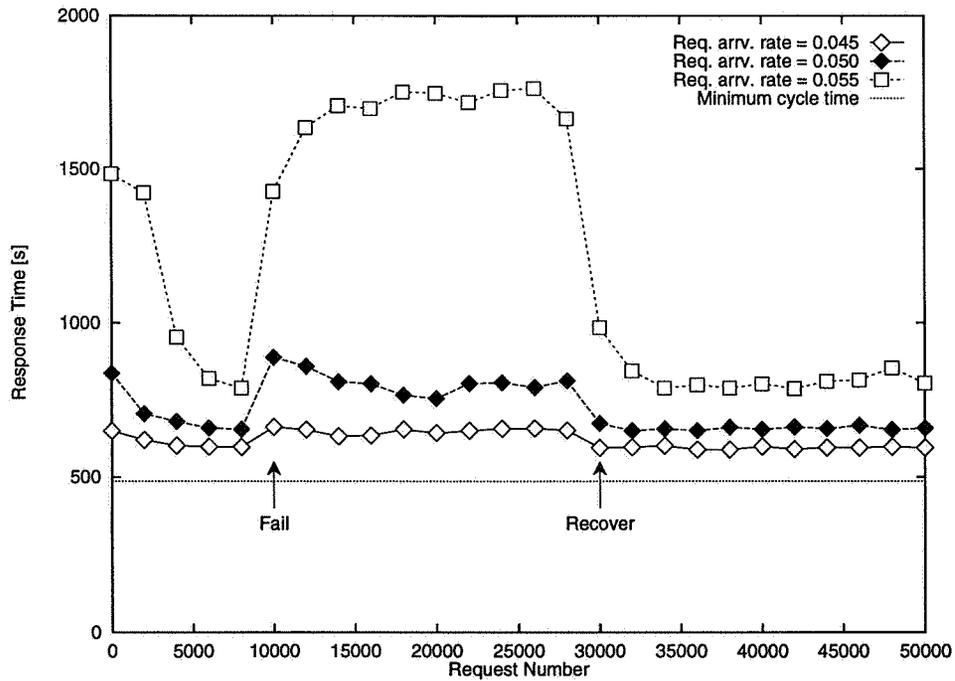


Figure 6: Average response time measured at intervals of 2,000 requests with both drives failure

I. Conclusion

In this paper, we described the design of a scalable tape archiver and the cassette migration algorithms for it. Only using foreground migration led to a large improvement in performance of the scalable archiver. In addition to foreground migration, background migration can improve performance even more. Using background migration together with foreground migration, the scalable tape archiver can continue to serve the requests, even when some drives fail.

We have already finished designing and developing the hardware of the scalable tape archiver and are now developing software for the scalable tape archiver. In the future we will examine the behavior of the scalable tape archiver with data striping.

References

1. Copeland, W. Alexander, E. Boughter, and T. Keller. "Data placement in bubba." Proceedings of the 1988 ACM SIGMOD International Conference on Management of Data, pp. 99-109, 1988
2. Weikum, P. Zaback, and P. Scheuermann. "Dynamic file allocation in disk arrays". Proceedings of the 1991 ACM SIGMOD International Conference on Management of Data, pp. 406-415, 1991.
3. Nemoto, Y. Sato, K. Mogi, K. Ayukawa, M. Kitsuregawa, and M. Takagi. "Performance evaluation of cassette migration mechanism for scalable tape archiver". SPIE Proceedings, "Digital Image Storage and Archiving System", vol. 2606, pp. 48-58, SPIE, 1995.

**Long-Term Archiving and Data Access:
Modelling and Standardization**

517-82
83199

Claude Huc, Thierry Levoir, Michel Nonon-Latapie

French Space Agency
CNES - CT/TI/PS
18 avenue Edouard Belin,
31055 Toulouse Cedex
France
huc@cst.cnes.fr
levoir@cst.cnes.fr
nonon@cst.cnes.fr
Tel :+33- 61 28 15 82
Fax :+33- 61 27 30 84

Abstract

This paper reports on the multiple difficulties inherent in the long-term archiving of digital data, and in particular on the different possible causes of definitive data loss.

It defines the basic principles which must be respected when creating long-term archives. Such principles concern both the **archival systems** and the **data**.

The archival systems should have two primary qualities: independence of architecture with respect to technological evolution, and genericness, i.e., the capability of ensuring identical service for heterogeneous data. These characteristics are implicit in the Reference Model for Archival Services, currently being designed within an ISO-CCSDS framework. A system prototype has been developed at the French Space Agency (CNES) in conformance with these principles, and its main characteristics will be discussed in this paper.

Moreover, the data archived should be capable of abstract representation regardless of the technology used, and should, to the extent that it is possible, be organized, structured and described with the help of existing standards. The immediate advantage of standardization is illustrated by several concrete examples.

Both the positive facets and the limitations of this approach are analyzed. The advantages of developing an object-oriented data model within this context are then examined.

1. Introduction

The observations and data gathered during spaceborne scientific payloads carried on board satellites or interplanetary probes are archived on the ground in the form of digital data, which are generally accessible to the PI teams or to larger communities. After more 30 years of experience in this field, the following facts have been observed:

- the volume represented by this data is always on the increase;
- some of the data has been lost because the physical medium became unreadable;
- other data has been or is likely to be lost because its structure was dependent on operating systems which are now obsolete;
- other data has been lost because an exhaustive and correct data description was no longer available;
- it has turned out to be impossible to keep as many access softwares in operating order as there are sets of data - essentially for reasons related to cost. Consequently, some of the data, while not actually lost, is no longer accessible;
- knowledge - or rather human expertise - concerning the oldest data is quickly disappearing.

Within this context, a safeguard plan for conserving data archived on 70,000 magnetic tapes at the French Space Agency (CNES) has recently been implemented. This data represents, for the most part, a priceless scientific heritage which should remain of great interest for several decades to come, or even longer. The cost of producing this data represents, in fact, the cost of all scientific space missions since the 1960's, which is, needless to say, enormous.

In practice, most of the observations made above are valid in many other fields (scientific, cultural, audio-visual, industrial, etc.). They boil down to the contradictions between the need to archive data in the long term and the speed at which the technology being used becomes outdated. Generally speaking, the loss of digital data is very often 'insidious', as the digital data is not physically 'visible'. Due to this fact, its degradation does not strike the mind as strongly as the deterioration of a book, for example, whose characters get less and less readable with time, or like an historical monument which crumbles to the ground.

This analysis led us to undertake a thorough technical study of the problems posed by long-term archiving. We reached the conclusion that the setting up and maintenance of long-term archival services can only be achieved if certain stringent conditions are imposed **on the archival systems and data**.

In order to avoid any ambiguity in the vocabulary, let us specify that by the term **archival system**, we mean a hardware and software system responsible for the main archival functions: insertion of the data supplied by the data producers, conservation of the data and anything needed for interpreting it, access to the information concerning the data and dissemination of the data to the users. Such a system is itself a component of an archival service, which is the human organization which, in particular, maintains this system in operating order.

Briefly, it may be said that archival systems should respect two main requirements:

- **the independence of their architecture** with respect to technological evolution: any archival system relies on rapidly evolving technologies and must thus be able to evolve along with these technologies. Nevertheless, its architecture should be such that technological evolutions in one field (the physical media containing the data, for example, or else the user interface) should not have repercussions leading to an uncontrollable chain reaction throughout the system. The system components must thus not be correlated among themselves. This characteristic led us to reflect on the modelling of archival services, and to design a Reference model for these services [4].

- **the genericness**, i.e. the capability of ensuring an identical service for heterogeneous data. The primary aim of this genericness is to reduce the volume of software to be maintained.

At the same time, the data should also take into account two other requirements:

- **its independence with respect to any technology**: the data should be capable of an abstract representation which is completely independent of the technology being used,

- **the application of standards** to the data in terms of structuring, organization, description, etc. The application of such standards is a necessary condition if the objective of genericness, defined at the system level, is to be attained.

An archival system prototype was developed by CNES in 1995 to test such an approach.

later in this article we will analyze - through the lessons learned in our experiments - the consequences of the requirements specified for the system level and for the data and metadata level.

2. The problem on the system level

It has been seen that any archival system is based on rapidly evolving technology. Our purpose should thus be to construct a modular system in which each component is sufficiently independent from the others to be able to evolve individually without calling either the system architecture or the principles of inter-component communication into question.

At the level of the MMI component, the emergence of W3 is a typical example of rapid technological change. The use of X11-type client-server systems has very quickly become outdated. Many access systems have thus become obsolete. Only those systems designed with an independent MMI component were able to adapt easily to this new technology.

These considerations first led us to look for a solution in terms of a general model for a long-term archival service which would be totally independent of technological advances. Other teams, in particular in the USA, have taken a similar approach and it soon became clear that we shared a common view of the problem on the first level of the model.

This first level of the archival service model was no more than an outline and an elaboration of an actual Reference Model which is currently being used within an ISO-CCSDS framework [4]. We shall thus limit ourselves to a rough description of this first model, and then go on to describe a system prototype developed by us in conformance with this preliminary model version, along with the lessons learned from this prototype.

It seems useful first of all to define the limits of an archival service and to identify the external elements interacting with this service. The following four external elements may thus be distinguished:

- the data producers,
- the data users,
- the system administrator,
- the authority responsible for choices and for decisions regarding policy and financing.

At the level of the model diagram shown below (figure 1), we have not included the latter since its interaction at the level of the archival system is limited.

The service itself consists of five major functions (or sub-services) with respect to the data : (see figure 1)

- **ingest**, which serves as the interface between the data producers and the service. This function controls the conformance of data and metadata provided by the producers with respect to the requirements defined by the archival service (standardization, etc.) and performs the actual insertion of this data and metadata into the service.
- physical data **storage**, involving an interface which hides the internal architecture. This storage may be designed to comply with the IEEE Mass Storage System Reference Model,
- data **management**, based on the organization of metadata,
- **access** to metadata which makes it possible for the service to check the user's access rights and for the user to be aware of the available data and to define a query,

- **dissemination** which makes it possible to retrieve the data from the storage service, to extract the parts in which the user is interested, and to deliver these parts to him.

Note: The distinction between external and internal elements must be made clear before the limits of a long-term data archival and access system may be defined. In our approach, the formatting of the data into a normalised and long-lasting format is done by the data producer rather than by the archival system. Similarly, any processing for the purpose of data analysis is the responsibility of the data user, while the service is uniquely responsible for delivering the data corresponding to the user's query.

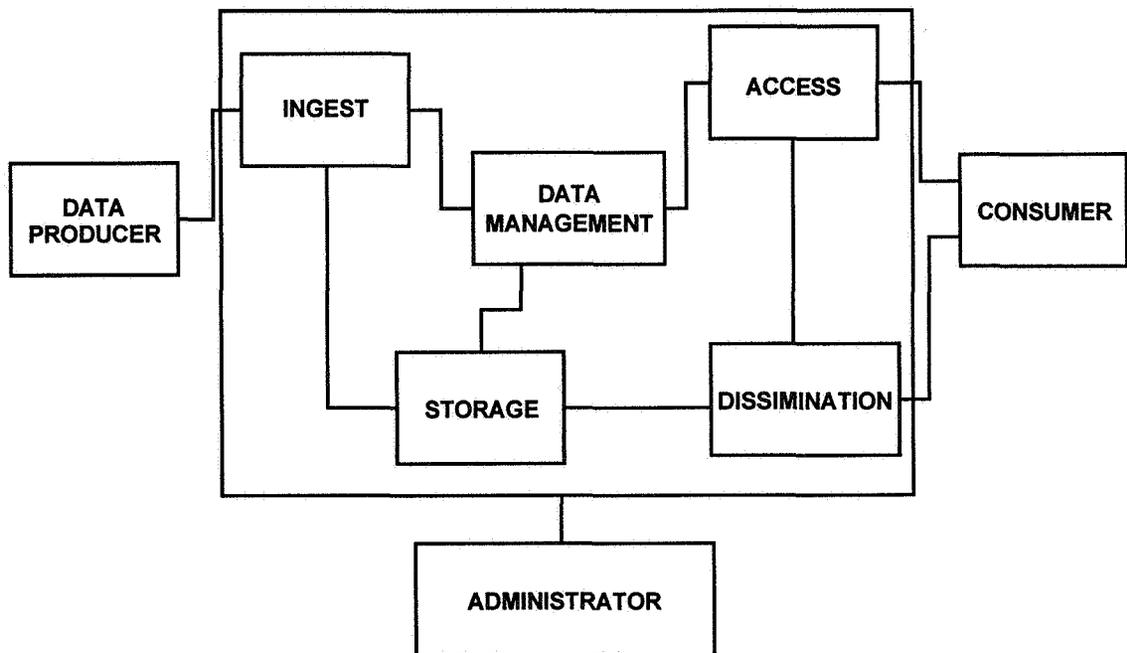


Figure 1: Preliminary view of the archival service model

Architecture of the first prototype

An archival system prototype was designed essentially on the basis of the modelling principles defined above, by making use of components already installed and used by CNES. Each component will be specified below, along with its description.

The system proposes access to chronologically ordered data. Within the system, a 'set of data' is characterized by a set of homogeneous data acquired in the same experiment, and having undergone the same processing. The principal components of a user query are the set of data and one or more time intervals with which it is associated.

- The storage system used is STAF (*Service de Transfert et d'Archivage des Fichiers*, or "File Transfer and Archival Service"). This system for the long-term physical preservation of data was set up at CNES 2 years ago. It functions in a heterogeneous environment, and is based on a client-server architecture. The client, responsible for data archiving and retrieval, functions on different host systems (UNIX, NOS-VE, etc.). This type of architecture makes the storage technology, and thus its evolution, invisible to the user (see figure 2).

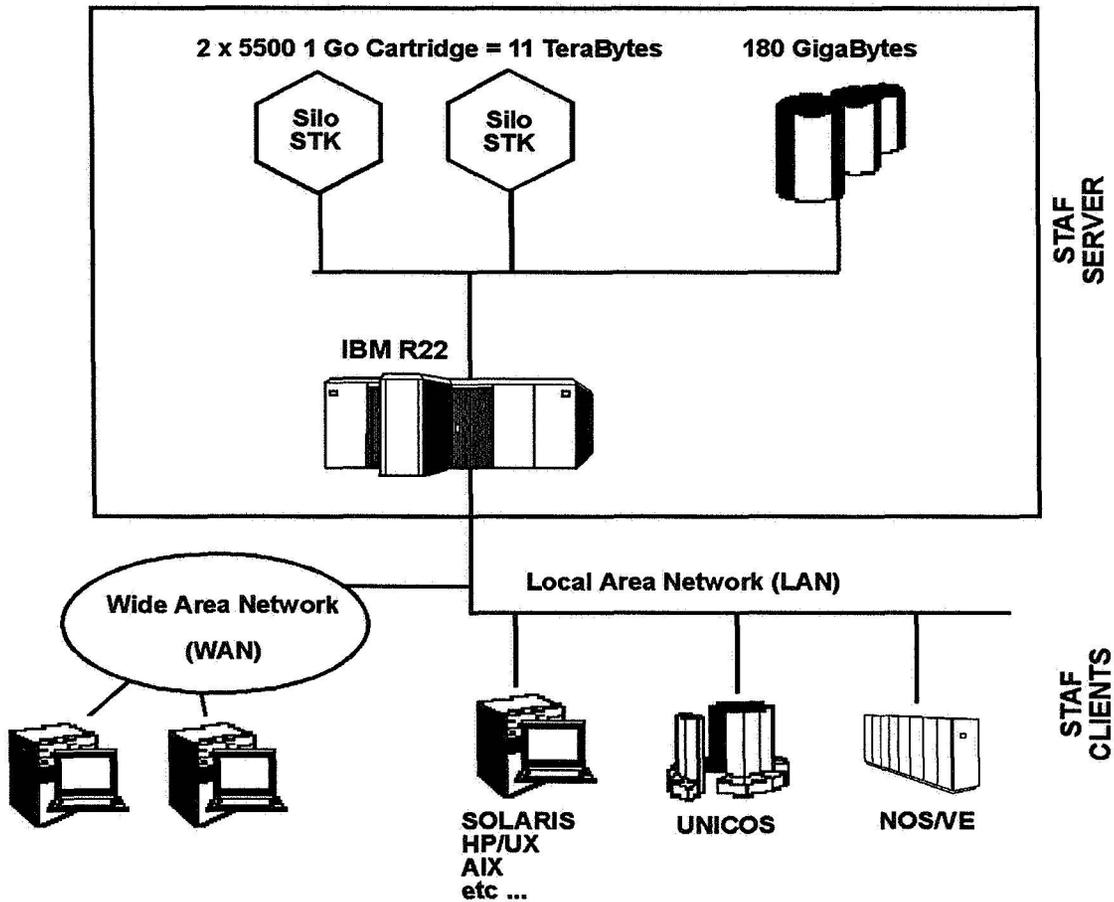


Figure 2: STAF diagram

- Metadata is managed by means of an ORACLE relational data base (cf. figure 3). This concerns for the most part the set of references for data placed in the storage service. The data base also manages :
 - the data protection,
 - the management of browse data (quick-looks),
 - the resources and quotas allotted to each user.

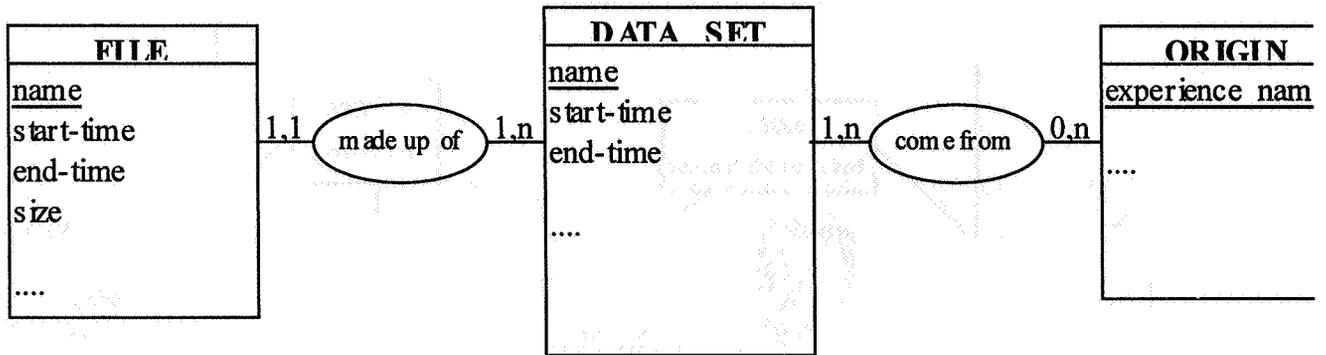


Figure 3: The main elements managed by the data base

- The data ingest service inserts only the metadata into the system, at the level of the data manager. This is performed by software based on the Oracle SQL*loader tools. The insertion of the data itself at the level of the storage function is independently performed by the data producers.
- The access service is based on a WWW server. The latter is linked to the data base by means of a cgi-bin written in Pro*C. The WWW server is responsible for checking the user's identification. This service is a critical point in the system, as it provides system access throughout the Internet. It has thus undergone a security study, to prevent any ill-intentioned intrusion.
- The data dissemination service is the set of generic programs enabling both the retrieval of data from the archive and its delivery, either by means of an FTP onto the user station or, at the level of the server station, into a W3 directory owned by that user. These programs, written in C, depend on a standardized date format, and use EAST descriptions [2] to extract the parts in which the user is interested from the archived files (see § 3.2).

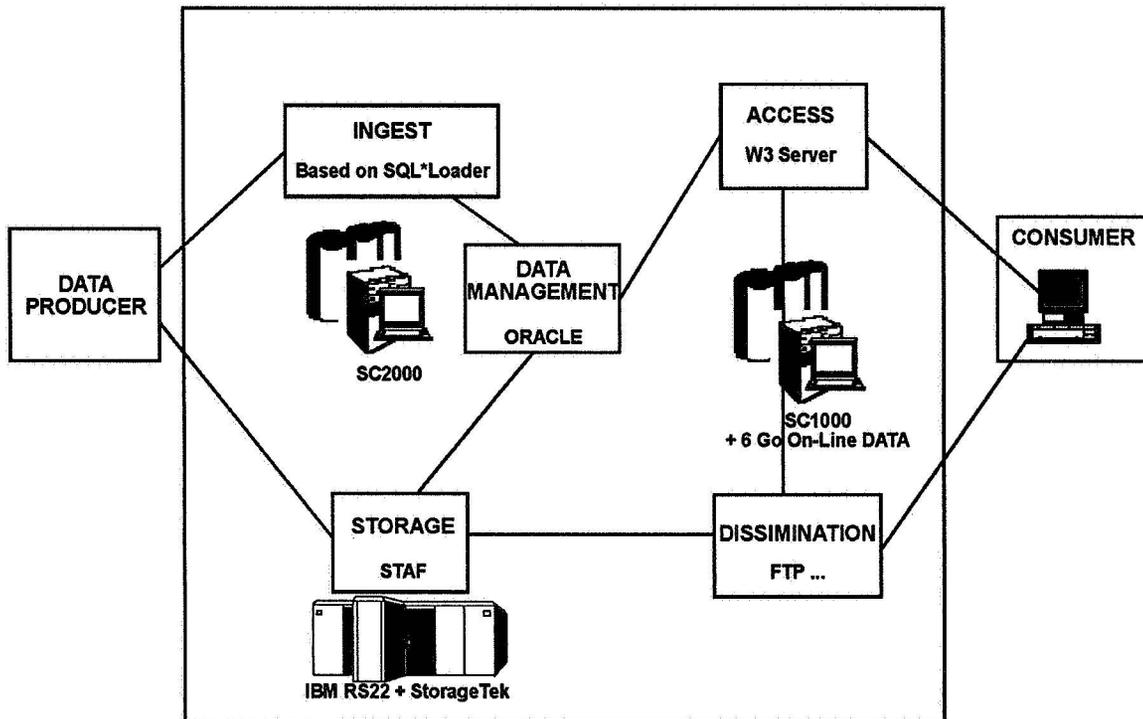


Figure 4 : The prototype architecture

3. The problem as far as the data and metadata are concerned

3.1 Fundamental rules for data independence with respect to the technology

The first fundamental rule, which is clearly necessary, is the independence of the data with respect to the machines, the operating systems and its environment in general. Any digital data item can be abstractly represented by a sequence of bits divided into fields. Each field may be subdivided into sub-fields, and the latter may be further subdivided until indivisible units of information are reached. The first rule requires in particular:

- that the bit sequence contain no information inserted by the operating system which created it: only the relevant bits defined by the user shall be included. Consequently, the use of any file structure into which the operating system has inserted information to help in administration or control is strictly forbidden.
- that the coding of elementary fields be performed in conformance with recognized standards (ISO/IEC 646 for characters, IEEE for floating-point numbers, standard representations of images and graphs, etc.) and that any representation specific to a given manufacturer be prohibited.

The second rule to be applied concerns the necessity of having an exact and exhaustive description of the bit sequence available: the position of each elementary field, a description of this field's coding, the nature and meaning of the information contained in this field. This second rule prohibits, for example, reading and writing data through the blind use of software tools which do not provide thorough knowledge of the bit sequence in its abstract representation.

While these requirements are elementary, they are far from having always been respected. They apply both to data and to metadata, and are necessary if the aim of data independence is to be attained. They apply to the abstract representation of the data rather than to its actual physical storage, which depends on the technology available at a given moment. They may naturally meet with difficulties related to a lack of standards in a given field.

3.2 The application of advanced data standards and the key to system genericness

The infinite diversity of information representations which may be imagined is such that it is certainly useless to try to provide advanced and generic data access facilities without first investing in the standardization of these representations. Let us consider two simple examples which we have encountered, concerning the standardization of times and dates on the one hand, and the standardization of descriptions on the other hand.

Standardization of times and dates : in certain scientific disciplines such as Space Physics, data is often organized chronologically. We discovered in the older data that the variety of time and date representation formats was almost as large as the number of existing data sets. Given such a situation, when a user is interested in data for a given time frame, two options exist :

- either to supply the archived files containing this time frame, which is hardly satisfactory,
- or to develop and implement at the archival system level a specific extraction program for each set of data, an unrealistic approach with respect to the long-term perspective.

It quickly became obvious that a standardization of times and dates would resolve this problem in a satisfactory manner. We therefore selected the standardization proposed by CCSDS [1], which is more complete than the ISO standard in this field. For our first prototype, we were able to develop a general program for the extraction of data corresponding to one or more time frames defined by the user from one or more files. This program makes it possible, as shown in figure 5 below, to extract only that data which corresponds strictly to the time frame requested by the user. The extraction function is entirely independent of the archive structure:

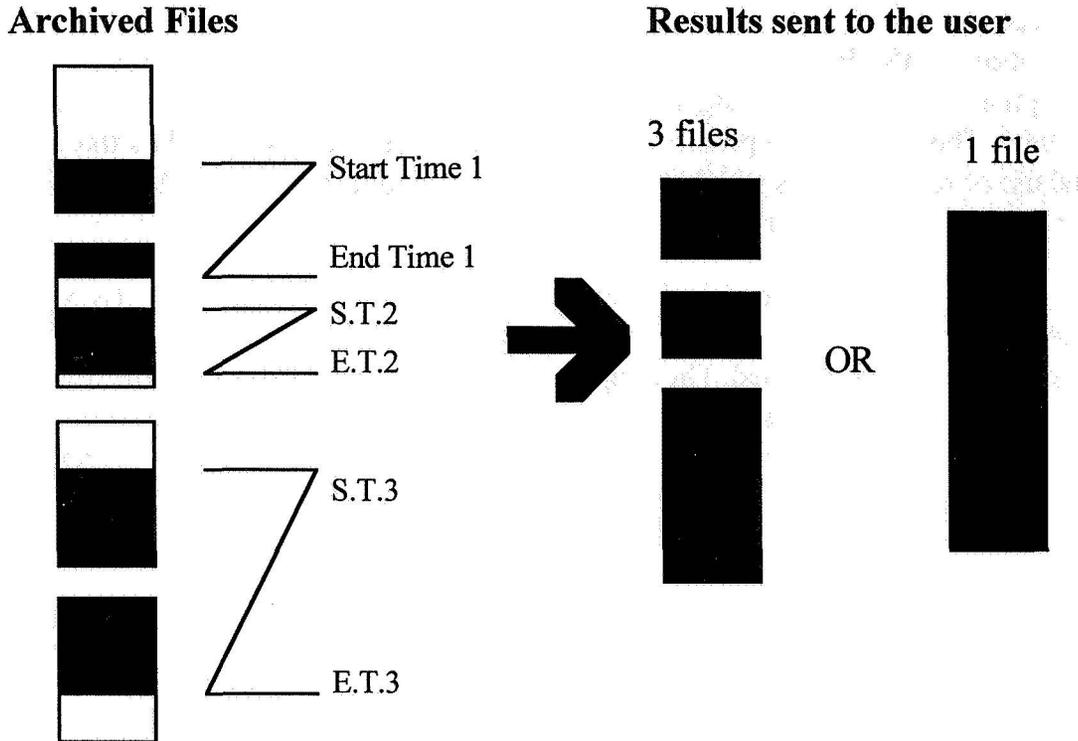


Figure 5: Extraction of chronologically ordered data

Standardized data descriptions : we discovered that the data was often described either incompletely (certain fields are left out of the description) or incorrectly (due to changes in the data creation program which were not carried over to the description documents). Moreover, the form of the description generally differs from one project to the next. The beginnings of a solution to this difficult and crucial problem in long-term archiving were found through the standardization of data description languages.

Our experiment, in our first prototype, was based on the EAST language (Enhanced Ada SubseT, [2]). This is a formal language around which certain general tools have been or are currently being developed. Worth mentioning in this field, in particular, are the Data Description Record Generator, the Data Generator, the Data Interpreter and the Data Formatter [3].

The interactive creation of data descriptions in EAST is performed with the help of a graphical interface, and the use of these descriptions for reading and writing data makes it possible to guarantee, during construction, the consistency between the data and its description.

Within this framework, we experimented with the use of a generic tool enabling the user to select a subset of information fields present in the archive.

The use of this tool involves two stages:

- a first stage in which the user selects the fields in which he is interested. A hierarchical tree representation of the different data fields is constructed on the basis of the EAST description. Using this representation, the user identifies and marks the fields in which he is interested (WWW interface).
- a second stage in which data is extracted from an archive and then 'filtered' so as to preserve only those fields requested by the user. (cf. Figure 6)

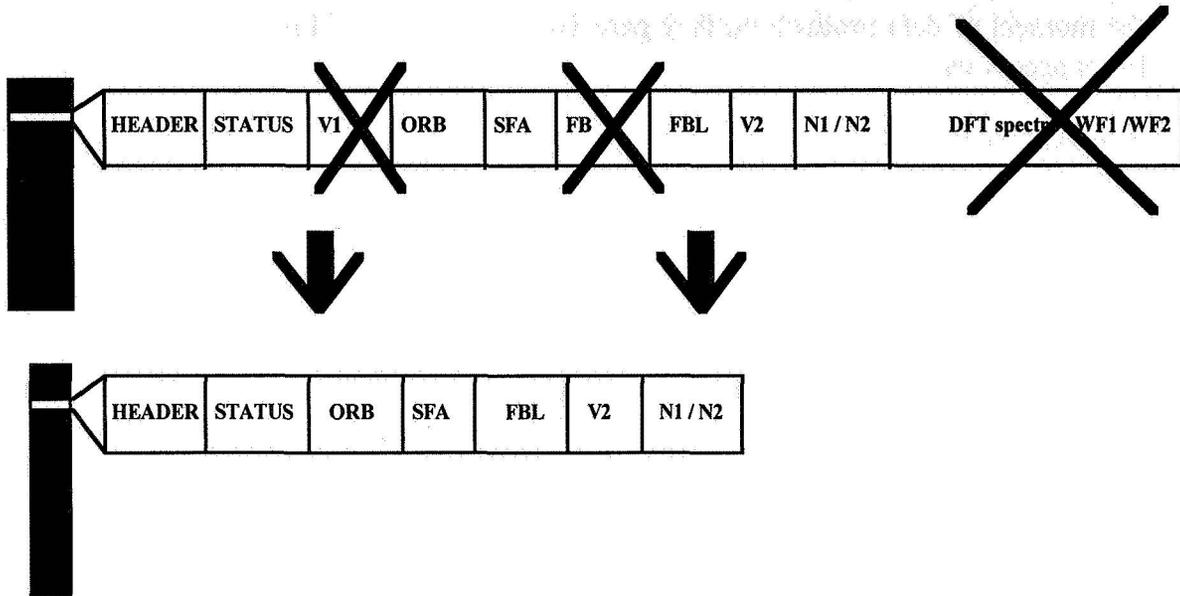


Figure 6 : Field extraction

The above are two meaningful examples with which we have experimented. They illustrate the correlation between the level of data and metadata standardization which we were able to attain and our capacity to preserve and keep the data accessible in the long-term.

4. Learned lessons

Positive points to be retained from the experiment with this first prototype

The aim of genericness within the field of chronologically organized data was achieved. Once the data producers began to respect the requirements set forth with respect to time and date standardization, we noted that it became very easy to access new data and hence that our approach had not simply been idealistic. At the present time, the system offers access to 32 different sets of data acquired during 5 space missions (INTERBALL,

Sweden VIKING, ISEE1, VOYAGER, GEOS). No specific tool had to be developed to make this data accessible through our prototype.

The service provided to the user is much better than the simple extraction of data from an archive, and since the volume of data transmitted to the user corresponds only to that data in which he is actually interested, there is a much more free space on the network to perform these transmissions.

Scientists do not naturally apply standards simply on principle. On the other hand, in a case such as that of times and dates, when the experiment teams applied the standard at the moment of data production, they perceived it as a way of immediately obtaining a better access service.

The limitations

The use of a relational model is the main limitation of our system. Adding a new selection criterion other than those defined at the moment of installation involves serious modifications both in the relational model of the metadata management function and in presentation at the level of the access function. This limitation curbs the open-endedness of the system.

5. Conclusion: towards an object-oriented data model

Without going into the details of work currently being performed on the object-oriented modelling of a long-term archival service, we shall explain a few important concepts:

- Data with shared characteristics can be collected into sets known as 'collections'. These collections can then be grouped together into 'collection groups', the collection groups themselves can be grouped together as well, and so on. Moreover, a collection may belong to several distinct collection groups. This representation led us to the construction of a directed graph.
- In order to define a query, a user will navigate through a directed graph which groups the data together according to scientific field, selection criteria or any other shared characteristic. To reach the data itself, each group offers selection criteria by means of which the user may select a given daughter group. This approach will provide the user with an infinite number of possibilities when searching for interesting data: if he wishes to create a new search route, he need only install the new groups needed to propose it.
- The lowest level group is a data collection grouping together a set of elementary logical data objects, while these logical objects are themselves made up of storage objects, i.e. in the general case, files. This approach, which may seem complicated at first glance, provides the system with considerable flexibility. A collection could correspond to a virtual data set, created at the same moment as it is being accessed.

For example, a set of image data can, at the level of storage, be made to correspond either to files containing several images or to files containing only a part of an image: through this approach, this becomes invisible to the system, which enables access to a collection of images reconstituted during this access, either by cutting up a file or by concatenating several files.

- Selection criteria for specific cases are available at the level of the groups or collections, and the same approach could also permit transformation or delivery criteria which could be applied to the data collections.

References

- [1] CCSDS: Time code Format, CCSDS 301.0-B-2, blue book, issue 2, April 1990
- [2] CCSDS: The data description language EAST specification, CCSDS 644.0-R-1, Red book, November 1995
- [3] B. Larzul, D. Minguillon, P. Mazal: EAST Technology for an automated processing of Space Data (to be published at the SPACE OPs symposium, Munich, November 1996)
- [4] CCSDS: Reference Model for Archival Information Services, June 1996, Version 4.0 (Available from <http://www.gsfc.nasa.gov/nost/isoas/overview.html>)
- [5] C. Huc: A strategy for the long term preservation of space mission data, JTS95 Symposium, London, January 27-29, 1995.

Automated Clustering-Based Workload Characterization

Odysseas I. Pentakalos Code 930.5 NASA GSFC Greenbelt MD 20771 odysseas@cesdis.gsfc.nasa.gov 301-286-4403	Daniel A. Menascé Dept. of CS George Mason University Fairfax VA 22030 menasce@cs.gmu.edu	Yelena Yesha Dept. of EE and CS Univ. of Maryland Baltimore County Baltimore MD 21228 yeyesha@cs.umbc.edu
---	--	---

1. Introduction

The demands placed on the mass storage systems at various federal agencies and national laboratories are continuously increasing in intensity. This forces system managers to constantly monitor the system, evaluate the demand placed on it, and tune it appropriately using either heuristics based on experience or analytic models. Performance models require an accurate workload characterization. This can be a laborious and time consuming process. In previous studies [1,2], the authors used *k*-means clustering algorithms to characterize the workload imposed on a mass storage system. The result of the analysis was used as input to a performance prediction tool developed by the authors to carry out capacity planning studies of hierarchical mass storage systems [3]. It became evident from our experience that a tool is necessary to automate the workload characterization process.

This paper presents the design and discusses the implementation of a tool for workload characterization of mass storage systems. The main features of the tool discussed here are:

- *Automatic support for peak-period determination*: histograms of system activity are generated and presented to the user for peak-period determination.
- *Automatic clustering analysis*: the data collected from the mass storage system logs is clustered using clustering algorithms and tightness measures to limit the number of generated clusters.
- *Reporting of varied file statistics*: the tool computes several statistics on file sizes such as average, standard deviation, minimum, maximum, frequency, as well as average transfer time. These statistics are given on a per cluster basis.
- *Portability*: the tool can easily be used to characterize the workload in mass storage systems of different vendors. The user needs to specify through a simple log description language how the a specific log should be interpreted.

The rest of this paper is organized as follows. Section two presents basic concepts in workload characterization as they apply to mass storage systems. Section three describes clustering algorithms and tightness measures. The following section presents the

architecture of the tool. Section five presents some results of workload characterization using the tool. Finally, section six presents some concluding remarks.

2. Workload Characterization

One of the important steps in any capacity planning and performance modeling study is workload characterization. The purpose of this step is to understand the characteristics of the workload submitted to a system and determine a synthetic description—called workload model—of the global workload. To make these concepts more specific, let us turn our attention to a mass storage system subject to two types of requests: ftp gets and ftp puts. Imagine that the system is observed during a few hours of operation the following information about each request is gathered:

- type of request (get or put),
- request arrival time,
- size of the file involved in the request, and
- time at which the file transfer completed.

If the system is sufficiently busy during the observation period you may collect thousands of such tuples. The question is what to do with this information? To use this information in a predictive performance model, one needs a more compact representation than a list with thousands of entries, one per request. If one looks at all requests, we may find that one can aggregate them into a reasonably small number of groups of “similar” requests. The notion of similarity is formalized in the next section. In this section we consider an intuitive meaning to the term. Within each group, each request is characterized by a pair (Z, S) where Z is the time since the last arrival of a request of the same type (get or put) and S is the file size. Suppose that one draws a scatter plot, such as the one in figure 1, where the x axis represents values of Z and the y axis represents values of S . As one can see, there is a natural grouping or clustering of points that have similar values of Z and S . Each *cluster* can then be represented by the coordinates of its center, called the *centroid*.

In the case of the example of figure 1, the centroids are: $(Z_1= 2.48 \text{ sec}, S_1= 4.26 \text{ MB})$, $(Z_2= 4.95 \text{ sec}, S_2= 107 \text{ MB})$, and $(Z_3= 13.76 \text{ sec}, S_2= 45.2 \text{ MB})$. The other information we obtain from the clustering exercise shown in figure 1 is that 41.2% of requests fall into cluster 1, 26.8% fall into cluster 2, and 41.2% fall into cluster 3. One can now drop all measurements and work with the more compact representation of the workload provided by the three clusters.

Another important aspect in workload characterization is the determination of the interval during which measurements are obtained. For capacity planning studies and system sizing, one usually looks for the periods of time when the system is more heavily utilized, or the *peak period*. This is usually obtained by looking at histograms of system activity, for example number of requests submitted or number of bytes transferred, during each

hour of the day for many days. The peak period is the time interval or sets of time intervals during which the load on the system is high compared to other intervals. Consider figure 2 that shows a histogram of number of get requests submitted to a mass storage system during a period of one day. As one can see, the peak period is between 9AM and 6PM. Thus, this is the period during which measurements should be collected.

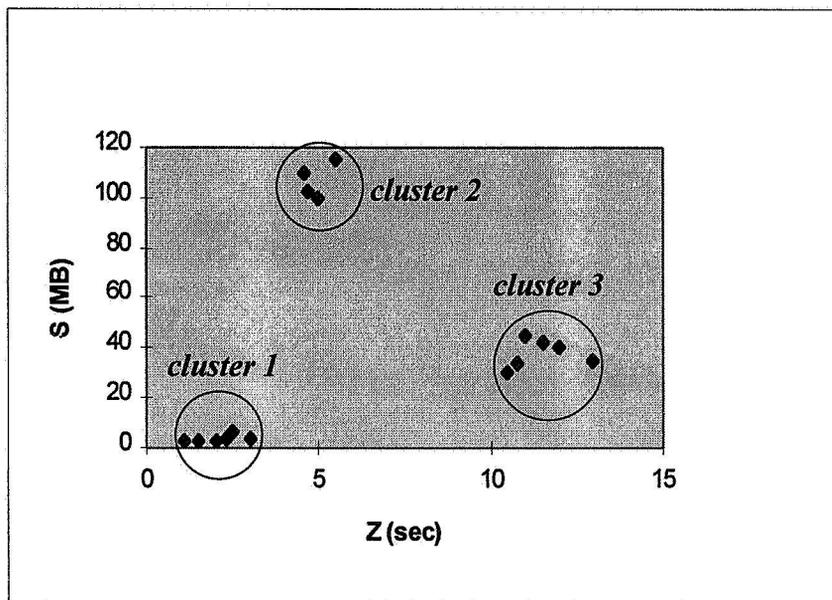


Figure 1 - S versus Z scatter plot.

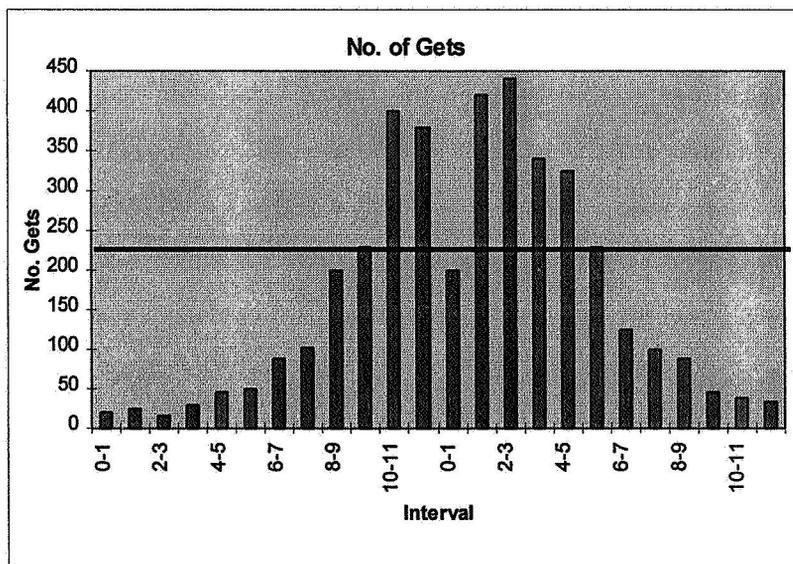


Figure 2 - Histogram for determination of peak period.

So, in summary, workload characterization is composed of the following steps:

1. Determine the basic type of requests (e.g., ftp gets and ftp puts).
2. Collect measurements on system activity for each type of request over a period of several days and plot a histogram to determine peak periods for each type of request.
3. Collect the measurements needed to characterize the workload during the peak period (e.g., measure file sizes, inter-arrival times for requests).
4. Cluster the measurements obtained into a small number of groups or clusters using a clustering algorithm (see next section).

The process described above can be quite laborious and time consuming. The purpose of the tool described in this paper is to automate the whole process for mass storage systems. The next section discusses in detail the algorithms used to perform clustering analysis and the criteria used to determine the number of clusters to use in workload characterization.

3. Clustering Algorithms

During the process of workload characterization of a mass storage system, logs of the requests which arrive at the system to be processed are analyzed. The objective of clustering is to classify the individual requests into a relatively small number of classes which impose on the average a load on the system similar to that of the actual workload. This classification is made based on a measure of similarity or proximity between the requests. A log with n data points consisting of d components each can be described as a set of vectors $\vec{x}_i = (x_{i,1}, \dots, x_{i,d})$ for $i=1, \dots, n$, where $x_{i,k}$ is the k -th feature of the i -th data point in the log. The proximity between data points is described most commonly in terms of an $n \times n$ matrix, called the *proximity matrix* where entry $d_{i,j} = d(i, j)$ is a distance metric (or dissimilarity measure) between the i -th and j -th data point. The three most commonly used distance metrics are: the Euclidean distance $d(i, j) = \sum_{k=1}^d (x_{i,k} - x_{j,k})^2$, the Manhattan distance $d(i, j) = \sum_{k=1}^d |x_{i,k} - x_{j,k}|$, and the sup distance $d(i, j) = \max_{1 \leq k \leq d} |x_{i,k} - x_{j,k}|$.

A number of approaches have been proposed in the literature for clustering data and those clustering approaches are themselves classified using various characteristics. A clustering algorithm is *exclusive* or *nonexclusive* based on whether the data points are allowed to belong to only one or more than one cluster, respectively. An exclusive clustering algorithm is *intrinsic* or *extrinsic* based on whether the classification is done based on only the proximity matrix or based on the proximity matrix and category labels assigned to the data points. In extrinsic clustering the objective is to determine the discriminant surface which separates the points based on their categories. An exclusive, intrinsic clustering algorithm is *hierarchical* or *partitional* based on whether the resulting

classification is a nested sequence of partitions or a single partitioning. Finally, an exclusive, intrinsic algorithm is *agglomerative* or *divisive* based on whether the algorithm proceeds by gradually merging clusters into larger and larger classes or by subdividing larger clusters into smaller ones.

The specific algorithm used in our tool can now be described using the above terminology as an exclusive, intrinsic, partitional, agglomerative algorithm. The problem of exclusive, intrinsic, partitional clustering can be stated as given n points and a desired fixed number of clusters K , select the partition of those points into clusters such that points in one cluster are more similar to points in their cluster than to points in the other clusters. The number $S(n, K)$ of ways to partition n points into K clusters is given by:

$$S(n, k) = \frac{1}{K!} \sum_{i=1}^K (-1)^{K-i} \binom{K}{i} i^n$$

Therefore, an exhaustive evaluation of all possible partitions is not feasible. The centroid of cluster C_k is given by:

$$\bar{m}_k = \frac{1}{|C_k|} \sum_{\vec{x} \in C_k} \vec{x}$$

The within-cluster variation e_k for cluster C_k is the average distance of all points in the cluster to its centroid. Thus,

$$e_k = \frac{1}{|C_k|} \sum_{\vec{x} \in C_k} d(\vec{x}, \bar{m}_k)$$

and the tightness E_K of a particular clustering is defined as the average of the within-cluster variation normalized by the maximum value of all within-cluster variations. Hence,

$$E_K = \frac{1}{K \max_{j=1}^K \{e_j\}} \sum_{k=1}^K e_k$$

Note that both e_k and E_K are numbers between 0 and 1. The closer to zero the value of the tightness, the better the clustering quality is. Typically, an iterative algorithm is used to minimize the global metric E_K . One disadvantage of iterative algorithms is that occasionally they terminate at a local minimum rather than at the desired global

minimum. In practice, one can get more confidence at the quality of the solution by repeating the algorithm with various different starting partitions and ensuring that the algorithm terminates at the same solution. The K -means algorithm is one such agglomerative, iterative algorithm. It is defined as follows:

1. Start with an initial assignment of points to K clusters.
2. Compute the centroid \bar{m}_k of each cluster C_k for $k=1, \dots, K$.
3. For each point \bar{x} in the collection do:
 - $j = \min_{1 \leq k \leq K} \{d(\bar{x}, \bar{m}_k)\}$ /* find the cluster closest to point \bar{x} */
 - let i be such that $\bar{x} \in C_i$
 - if $i \neq j$ then
 - $C_j = C_j \cup \{\bar{x}\}$. /* add \bar{x} to the j -th cluster if not already there */
 - $C_i = C_i - \{\bar{x}\}$ /* remove \bar{x} from the i -th cluster */
 - end if
4. Recompute the centroid \bar{m}_j of each cluster C_j for $j=1, \dots, K$.
5. Repeat steps 2 through 4 until no point changes its cluster assignment during a complete pass or a maximum number of passes is performed.

A number of variations exist for the K -means algorithm based on how the initial cluster centroids are selected and on whether the cluster is recomputed after each re-assignment of a point or after an entire pass through the points has completed as described above [4,5,6,7]. The version of the algorithm used in our tool is the one proposed by Forgy[8].

4. Architecture of the Tool

Figure 3 shows the architecture of the workload characterization tool. Modules are shown in ovals and internal databases are shown in parallel solid lines. The figure also shows the Hierarchical Mass Storage System (HMSS) log and a file containing a description of the log format as the two input files to the tool. Users interact with the tool through a Graphical User Interface (GUI). Through this interface they can request the tool to open a specific log, generate histograms for peak period analysis, do workload characterization through clustering, and view file access statistics and the results of the workload characterization process.

The *Filter* process reads the log descriptor and reads the HMSS log and stores the information in a Measurements DB. This feature of the tool allows it to be used to characterize workloads for virtually any HMSS provided one can specify the log format using the log descriptor. Once the information is entered into the DB it can be used as input to the *Histogram Generator* and *Clustering Engine* modules. The *Histogram Generator* plots a histogram of system activity for a selected day and time and a selected type of request (get or put). The *Clustering Engine* reads the data points from the Measurements DB, runs a K -means clustering algorithm, and saves the data into a

Workload DB. This module also generates tightness measures to illustrate the quality of the clustering generated. The *Report Generator* module reads the workload model from the Workload DB and generates a workload description in the format expected by Pythia—a tool for performance prediction of mass storage systems developed by the authors [3]. The *Report Generator* also stores that information in a format needed by the *Results Manager* for presentation of file access statistics and workload model characteristics to the users of the workload characterization tool.

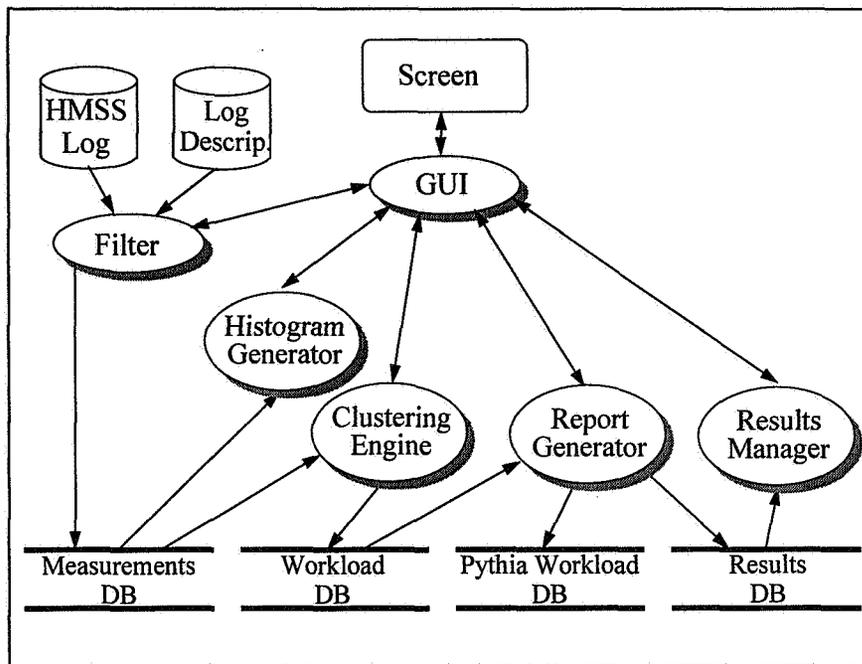


Figure 3 - Architecture of the e Tool.

5. Using the Tool

This section describes the operation of the tool by going through a sample session of workload characterization. Figure 4 shows the main screen of the tool. The menu bar has three options. The File menu provides options for opening a log of data, saving the results of the workload characterization, and quitting from the tool. The View menu provides options for generating a histogram, viewing the tightness measure as a function of the number of clusters, and generating a workload characterization. Finally, the Help menu provides help on the use of the tool. The Data Range window describes the range of the data values contained in the currently selected log file. Initially, the range will be empty, but as soon as a log file is opened from the File menu option, the entries will be filled automatically to describe the range. The user may change the range under consideration at any time.

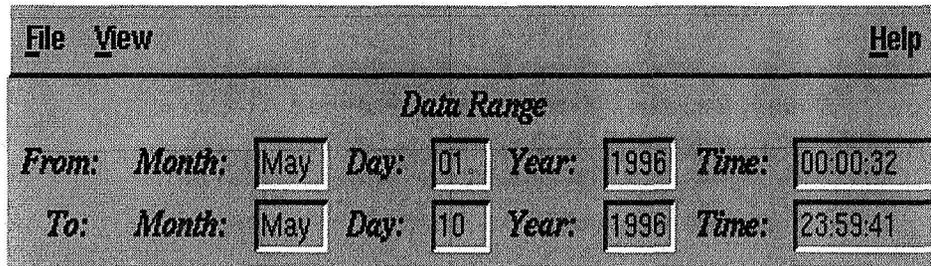


Figure 4 - Main Screen of the Tool

The next step after opening a log file is to view a histogram of the data so that the peak period for this workload can be detected. This is done by selecting the Histogram menu option from the View menu. Figure 5 shows the window that comes up for selecting which workload (get or put) should be considered in the histogram. Figure 6 shows a sample histogram for get requests. The results shown are for the first day in the data range when the histogram computation was executed. The user may modify the data range and run the histogram again. Pressing the “Previous Day” button, loads the data for the previous date and displays the histogram, and pressing the “Next Day” button loads the data for the next date. Pressing “OK” closes the histogram window. Browsing through the histograms for a number of days allows one to select the peak period during the day.

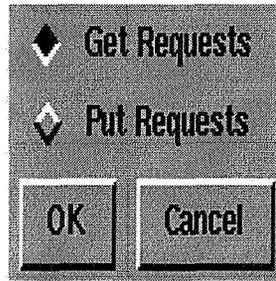


Figure 5 - Histogram Configuration Window

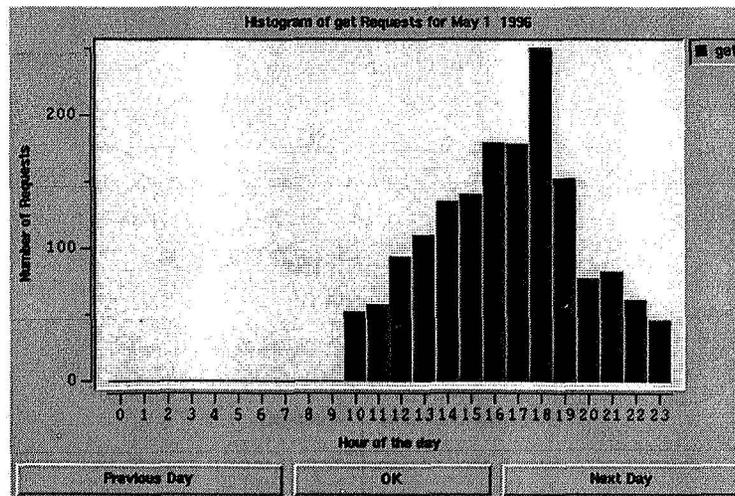


Figure 6 - Histogram Window

Once the peak period has been selected, the data range can be modified and the user can then select the “Tightness” option from the View menu. Figure 7 shows the window which appears for this option. Again, it allows the user to select the type of workload to be analyzed and also the range of clusters to be evaluated. Figure 8 shows the sample plot for a cluster range of two through ten for get requests. This plot is very helpful in determining the most appropriate value for the number of clusters. It allows one to visually select a local minimum for the tightness measure which compromises between a fairly accurate workload with as small a number of clusters as possible.

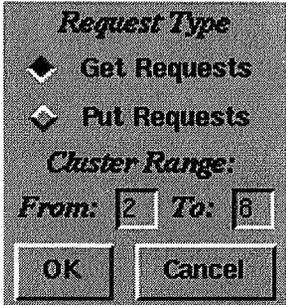


Figure 7 - Tightness Configuration Window

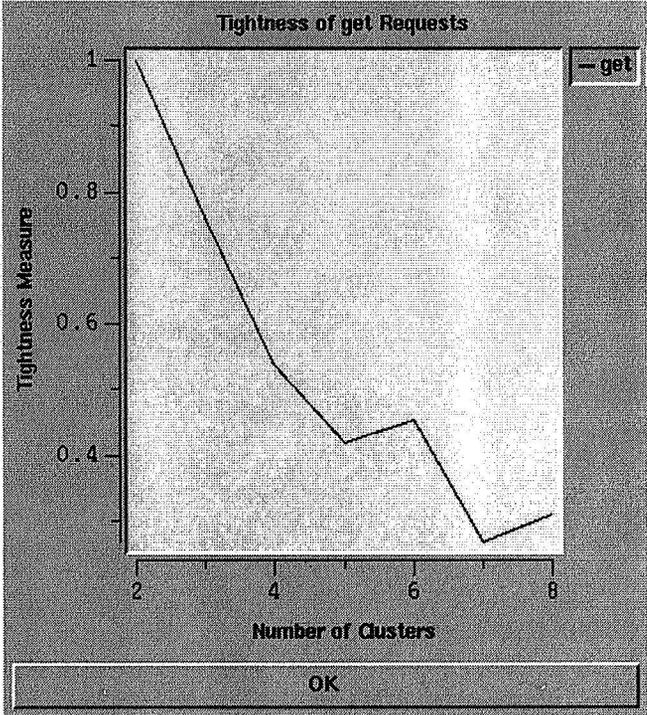


Figure 8 - Tightness Variation Plot

The final step in the workload characterization is to determine the cluster centroids, and cluster sizes, for a given number of clusters. This is done by selecting the “Workload” option from the View menu. The Workload Configuration window comes up, shown in

figure 9, which allows one to select the workload type, and number of clusters. Once the workload has been computed the Workload window pops-up which describes the workload parameters as shown in figure 10. The workload window includes information such as the name of the log file, the number of clusters selected, and for each cluster, describes the centroid, the number of points from the log which belong to the cluster, and the frequency.

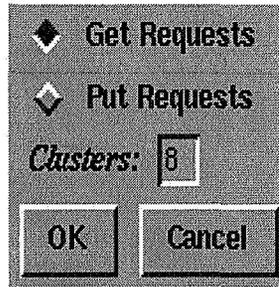


Figure 9 - Workload Configuration Window

Cluster	Centroid (KB)	Count	Freq.
0	26198.97	1169	19.62
1	114678.41	342	5.74
2	233948.49	64	1.07
3	2888.69	440	7.39
4	558963.69	16	0.27
5	7640.48	869	14.59
6	308.80	3052	51.23
7	1259004.45	5	0.08

Figure 10 - Workload Window

6. Concluding Remarks

This paper described the design and operation of a tool for automating the workload characterization of mass storage system workloads. The tool is based on a variation of the *K*-means clustering algorithm. In addition to characterization of the workload, the tool allows one to determine the peak-period of system usage by providing a browsing capability through histograms of workload requests, and also simplifies the task of selecting the number of clusters present in the workload by plotting a measure of the accuracy of the clustering as a function of the number of clusters.

Bibliography

- [1] Daniel A. Menascé, Odysseas I. Pentakalos, and Yelena Yesha, "An Analytic Model of Hierarchical Mass Storage Systems with Network-Attached Storage Devices," Proc. of the ACM SIGMETRICS'96 Conference Philadelphia, PA, May 23-26 1996.
- [2] Odysseas I. Pentakalos, Daniel A. Menascé, Milt Halem, and Yelena Yesha, "An Approximate Performance Model of a Unitree Mass Storage System," 14th IEEE Symposium on Mass Storage Systems, Monterey, California, September 1995, pp. 210--224.
- [3] Odysseas I. Pentakalos, Daniel A. Menascé, and Yelena Yesha, "An Object-Oriented Performance Analyzer of Hierarchical Mass Storage Systems," submitted to the 1996 Computer Measurement Group Conference, San Diego, CA, December 1996.
- [4] Michael R. Anderberg, *Cluster Analysis for Applications*, Academic Press, New York, NY, 1973.
- [5] Anil K. Jain and Richard C. Dubes, *Algorithms for Clustering Data*, Prentice Hall, Englewood Cliffs, NJ, 1988.
- [6] Leonard Kaufman and Peter J. Rousseeuw, *Finding Groups in Data*, John Wiley & Sons, Inc., New York, NY, 1990.
- [7] Daniel A. Menascé, Virgilio A. F. Almeida, and Larry W. Dowdy, *Capacity Planning and Performance Modeling: from mainframes to client-server systems*, Prentice-Hall, Englewood Cliffs, NJ, 1994.
- [8] E. Forgy, "Cluster Analysis of multivariate data: efficiency versus interpretability of classifications", *Biometrics*, 21, 768, 1965.

Digital Optical Tape: Technology and Standardization Issues

Fernando L. Podio
National Institute of Standards and Technology
Building 225, Room A261
Gaithersburg, MD 20899
fernando.podio@nist.gov
301-975-2947
301-216-1369 (fax)

Abstract

During the coming years, digital data storage technologies will continue an aggressive growth to satisfy the user's need for higher storage capacities, higher data transfer rates and long-term archival media properties. Digital optical tape is a promising technology to satisfy these user's needs. As any emerging data storage technology, the industry faces many technological and standardization challenges. The technological challenges are great, but feasible to overcome. Although it is too early to consider formal industry standards, the optical tape industry has decided to work together by initiating pre-standardization efforts that may lead in the future to formal voluntary industry standards. This paper will discuss current industry optical tape drive developments and the types of standards that will be required for the technology. The status of current industry pre-standardization efforts will also be discussed.

1. Introduction

The data storage industry needs to satisfy a substantial growth in user's requirements for high performance, large capacity mass storage subsystems. New applications and data types, and the need to store and quickly retrieve massive amounts of data require systems with large on-line and near on-line capacities, very high transfer rates and good archival media properties. A hierarchy of storage devices is needed to satisfy these requirements. Optical tape subsystems will fit well in that storage hierarchy. Optical tape media offers the potential for high aerial density, high transfer rates and an expected long archival life. As an emerging technology, however, the optical tape industry faces many challenges. Magnetic tape, a competing technology, is well established and also promises high performance products with high capacity and high transfer rates. The technological challenges that the optical tape industry faces are feasible to overcome. Standardization is also a challenge as it has been for other data storage technologies. Recent industry developments, and standardization issues will be addressed in the following sections.

2. Current Product/Developments¹

There is only one optical tape drive on the market today. The CREO/EMASS drive offers one Terabyte of data on a 35 mm reel of optical tape. The sustained transfer rate is 3 Mbytes/s. The media used by this drive is WORM (Write-Once Read Many). The specified access time is 65 seconds average for one Terabyte.

In 1995, the Advanced Technology Program (ATP) of the National Institute of Standards and Technology (NIST) awarded two projects related to this technology [US Department of Commerce News, 1]². One awardee was LOTS Technology, Inc. ("Digital Data Storage Technology via Ultrahigh-Performance Optical Tape Drive Using a Short-Wavelength Laser"). The project includes the development of optical tape read/write technology which could lead to systems capable of storing one Terabyte and capable of transferring that data at a rate of at least 100 Megabytes per second. In order to achieve this performance, the project includes the development of optical tape storage technology in which up to 180 tracks can be simultaneously written and read with multiple, independently controllable laser beams. It is planned that the first application of this technology would be an "IBM-3480"-style cartridge.

The other ATP awardee in this technology field was a joint venture of Terabank Systems Inc., Polaroid Corp., Science Applications International Corp., Xerox Corp., Carnegie Mellon University, Energy Conversion Devices, Inc., NASA Goddard Space Flight Center, and the University of Arizona ("Technology Development for Optical-Tape-Based Rapid Access Affordable Mass Storage (TRAAMS)"). (Motorola Corp. joined the venture later on.) This project includes the development of a thin-gauge erasable/rewritable optical tape on which a bit of data can be recorded on a spot six-tenths of a micrometer in diameter. The proposed tape handling methods would move the tape at 2,500 centimeters per second. The expected transfer rate is 6 Mbytes/s and when accessing multiple tracks of the tape it can reach up to 100 Mbytes/s. Assuming the ATP project is successful, the expected user capacity of the first product that could result from the use of this technology is 100 Gbytes.

In addition to the projects awarded by ATP, technology projections indicate the following developments (see Table 1). Table 1 indicates technology projections, **not** product announcements.

LOTS Technology's first development will hopefully result in a product with one 1 Terabyte capacity and a sustained transfer rate of 15 Mbytes/s using a WORM optical

¹ Certain trade names and company products are mentioned in the text in order to adequately describe industry developments. In no case does such identification imply recommendation or endorsement by the National Institute of Standards and Technology, nor does it imply that the products are or will be necessarily the best available for the purpose.

² The ATP cost shares high-risk industrial R&D projects to overcome technical barriers, rather than product development. If projects are successful, companies will develop products exploiting the technology with their own funds. In the descriptions of ATP projects in this paper, the technology projections reflect specifications of products that the companies hope to develop to exploit the new technology if the ATP projects are successful.

tape .5 in width and 400 m long. EMASS has plans for a line of optical tape drives. The first product is scheduled to have a capacity of 200 Gbytes at a 12.5 Mbytes/s sustained transfer rate. This product uses WORM media and specifies an average access time of 50 seconds, Chu [2]. (As of this writing, July 1, 1996, EMASS's parent company Raytheon has placed on hold this optical tape program and the company is for sale.)

Terabank's technology projections include doubling the capacity and the transfer rate of the first product that might result from the ATP technology to 200 Gbytes and 12 Mbytes/s. Support for WORM and rewritable media is expected. Researchers at Philips Research Laboratories have described a compact optical tape recorder with a 80 Gbytes capacity in a 8 mm cassette at approximately 30 Mbits/s transfer rate and 15 seconds access time, van Rosmalen, Kahlman, Put, and van Uijen [3]. Primelink proposes an optical tape drive with 36 Gbytes and 72 Gbytes with an access time of less than 1 second and a transfer rate for the first version of the drive of 15 Mbit/s, Newell [4].

Table 1 - AIIM Optical Tape Study Group Technology Roadmap(*)

System name	Average Access Time (sec)	User Capacity (GB)	Sustained Data Rate (MB/sec)	Year of Introduction	Tape Width Tape Length
CREO/EMASS OTR	65	1000	3	1991	w=35mm, L=800m
Terabank TRAMMS1	2	100	6	1998	w=8mm, L=120m
Terabank TRAMMS2	2	200	12	1999	w=8mm, L=120m
Terabank TRAMMS3	2	400	24	2001	w=8mm, L=180m
LOTS Lasertape 15T	20	1000	15	4Q 1997	w=0.5in, L=400m
Lots Lasertape 100T	20	1000	100	4Q 1998	w=0.5in, L=400m
Philips	15	80	3.75		w=8mm, L=110m
Primelink	0.05 - 2	72 - 1350	1.88	1995 - 2000	w=8mm, L=100m
EMASS Harrier *	25	200	12.5	4Q 1997	w=0.5in, L=450m
EMASS Osprey *	15	600	15	3Q 1998	w=0.5in, L=708m
EMASS Rewritable **				4Q 1999	
EMASS Extended Future **		800, 2400	80	2001	

(*) Technology projections, **not** product announcements

(**) Raytheon has placed on hold the E-MASS optical tape program

Other plans and developments were reported by NHK and Sony researchers. NHK researchers reported development of a high capacity optical tape recorder able to store information for high definition TV and future TV systems, Tokumaru, Arai, Yoshimura, and Oshima [5]. Sony researchers reported developments of a helical scanning optical tape recorder and new tracking methods for that recorder, Narahara, Kamtami, Nakao, Kumai, and Ozue [6].

Media manufacturers include: ICI Imagedata (dye polymer media)³, Eastman Kodak (WORM phase change media), Polaroid (phase change media) and Southwall/Dow (bubble forming media). Other companies have researched the use of magneto-optical (MO) media. Details of these media characteristics and formulations can be found in different publications, Ashton [7]. Changes might be expected in these media. Final media configurations cannot be designed independently of the drive manufactures' specifications and requirements.

3. Standards for Optical Tape Media/Subsystems

In order to establish optical tape as a recognized data storage technology, the optical tape industry will need to provide users with storage solutions that conform, if possible, to voluntary industry standards. A family of standards is required for any particular data storage technology. One classification of these standards includes:

- ⇒ Data interchange standards
- ⇒ Test methods standards
- ⇒ Data integrity standards

3.1 Data interchange standards

Data interchange standards promote the availability of competitive products and multiple sources of media. They also increase the user's confidence in the technology by assuring media/data interchange between products and the long-term availability of drives and media. The following model represents four levels of compatibility, Hogan [8]

- **Level 4 - Applications**

There are many application level standards. The following are some examples of this type of standards:

- ⇒ data compression schemes for raster-scanning documents
- ⇒ control codes for word processing
- ⇒ media error monitoring and reporting techniques for verification of stored data

³ ICI Imagedata will be withdrawing from the optical tape business over the next two years.

- **Level 3 - Logical format for the media**

Logical volume label and file structure standards are in this level. They facilitate the interchange of data among different information processing systems.

- **Level 2 - Physical format of the media**

This level of standards deals with the recorded characteristics of the media. It includes characteristics such as:

- ⇒ data and track format
- ⇒ track locations
- ⇒ modulation schemes

- **Level 1 - Media properties**

This level of standards specifies the unrecorded or unformatted characteristics of the media. It specifies the physical and optical properties and assures the interchange (read/write) of media among different drives. The media properties that must be standardized include characteristics such as:

- ⇒ mechanical properties
- ⇒ dimensional properties
- ⇒ optical properties
- ⇒ read/write/erase properties

3.2 Test methods standards

Test methods standards for the media characteristics help avoid conflicts between media suppliers and drive manufacturers, allows for easy documentation of round-robin tests, and allows testing for conformance to media interchange standards.

The type of characteristics that require test methods are:

- ⇒ mechanical properties
- ⇒ optical properties
- ⇒ recording layer properties
- ⇒ substrate properties
- ⇒ preformat properties
- ⇒ environments: operational and storage

3.3 Data integrity standards

Long term availability of data depends on many factors including data integrity, media life expectancy and the availability of subsystems that can access the media where data

resides. As noted above, data interchange standards improve the chances of the long-term availability of drives and media. Standards for data/media preservation are also required.

They include:

- ⇒ standard media error monitoring and reporting techniques to verify stored data
- ⇒ life expectancy standards

Media error monitoring and reporting techniques allow users to monitor the status of the stored data. An example of standard media error monitoring and reporting techniques (for optical disk drives) can be found in ANSI/AIIM MS59 [9]. Life expectancy standards allow users to select media according to their long-term requirements by comparing manufacturer's media life expectancy claims.

4. The Association for Information and Image Management International (AIIM) Optical Tape Study Group

AIIM formed the Optical Tape Study Group (OTSG) in response to industry interest in initiating technology discussions and pre-standards work. AIIM OTSG is an open forum where industry and users can together discuss technical issues such as media characteristics, metrology and data integrity of optical tape media/systems and user's needs. The Study Group has attracted broad industry and user's participation. OTSG does not have the responsibility for developing formal standards. However, the OTSG may generate, in the future, standard development projects proposals.

The OTSG has prepared a matrix of unrecorded media characteristics and it is planning to develop a document specifying these characteristics. Test methods for the media characteristics are also discussed and they will be documented. In addition, OTSG is specifying media error monitoring and reporting (MEMR) techniques to verify stored data on optical tapes. The documentation of these MEMR techniques is underway. Table 1 shows the Technology Roadmap developed by the OTSG. Table 2 shows industry and user organizations included in AIIM OTSG's participant's list. The OTSG held two meetings in 1995 and three meetings in 1996. The next meeting will be held October 3 - 4, 1996 at AIIM Headquarters, 1100 Wayne Ave., Suite 1100, Silver Spring, MD.

5. Conclusions

Any emerging data storage industry faces many technological and standardization challenges. The technological challenges are great, but feasible to overcome. Standardization will also be a challenge. In addition to the only existing commercial drive, several drive developments are taking place. Some of them might become commercial products in the coming years. The industry is not mature enough to address formal standards developments. However, recognizing that standards are necessary when products become available, the optical tape industry has initiated pre-standards work

through the AIIM Optical Tape Study Group. Drive and media makers as well as other organizations are discussing media characteristics, test methods for these characteristics, data integrity issues and user's needs. This industry anticipatory work may lead to formal standardization efforts.

Table 2 - Organizations Participating in OTSG

Ampex Data Systems	Apex	Boeing	Callicot	CIO
Customer Refocus	LDS Church	Eastman Kodak	E-MASS	E-Systems
Filetek	Genealogical Society of Utah	ICI Imagedata	IIT Reserach Institute	IRS
LaserTape	Library of Congress	Loral Federal Systems	LOTS Technology	MITRE
Mobil Oil	NASA/GSFC	National Archives	NML/3M	NIST
NSIC	Philips Research Laboratories	Polaroid	Primelink	Radix Systems
SAIC	Systems Engineering & Security	StorageTek	SUN Technologies	SyntheSys Research
Technology Solutions	Terabank Systems	TRW	University of Arizona	

6. References

- [1] US Department of Commerce News, August 15, 1995.
- [2] John Chu, Response to AIIM OTSG Request for Information, April 1996.
- [3] G. E. van Rosmalen, J. A. H. Kahlman, P. L. Put and C. M. J. van Uijen, SPIE, Vol. 2338, 1994.
- [4] Chester W. Newell, "Petabyte Mass Storage Memory System Using the Newell Opticell (tm)", Fourth NASA Goddard Conference on Mass Storage Systems and Technologies", March 1995, NASA Conference Publication CP-3295.
- [5] Haruki Tokumaru, Kiyotaka Arai, Shin-ichi Yoshimura, and Hideo Oshima, "Recording Experiment with Rotating optical Head for Magneto-Optical tape Recording System", SPIE Vo. 2514, 1995.

- [6] Tatsuya Narahara, Yoshiteru Kamtami, Takashi Nakao, Satoshi Kumai, and Tadashi Ozue, "A New Tracking Method for Helical Scanning Optical Tape Recorder", Jpn. J. Appl. Phys., Vol 32, Part 1 No. 11B, August 1993.
- [7] Gary Ashton, "New Opportunities in Optical Tape", Dataquest Conference, June 1996.
- [8] Michael D. Hogan, "Digital Data Interchange Reference Model for Removable Computer Storage Media", September 1988.
- [9] ANSI/AIIM MS59, "Media Error Monitoring and Reporting Techniques for Verification of Stored Data on Optical Digital Data Disks", March 1996.

520-82
83202

**Storage and Network Bandwidth Requirements
Through the Year 2000
for the
NASA Center for Computational Sciences**

Ellen Salmon

Science Computing Branch
NASA Center for Computational Sciences
NASA/Goddard Space Flight Center, Code 931
Greenbelt MD 20771
xrems@dirac.gsfc.nasa.gov
phone: 301-286-7705
fax:301-286-1634

Abstract

The data storage and retrieval demands of space and Earth sciences researchers have made the NASA Center for Computational Sciences (NCCS) Mass Data Storage and Delivery System (MDSDS) one of the world's most active Convex UniTree systems. Science researchers formed the NCCS's Computer Environments and Research Requirements Committee (CERRC) to relate their projected supercomputing and mass storage requirements through the year 2000. Using the CERRC guidelines and observations of current usage, some detailed projections of requirements for MDSDS network bandwidth and mass storage capacity and performance are presented.

Introduction

The mission of the NASA Center for Computational Sciences is to enable advanced scientific research and modeling for NASA-sponsored space and Earth science researchers by providing a high performance scientific computing, mass storage and data analysis environment. Science efforts supported by NCCS resources include climate data assimilation and other atmospheric and oceanographic sciences, orbit determination, solid-earth and solar-terrestrial interactions, magneto hydrodynamics, and astrophysics. The NCCS is a part of NASA/Goddard's Earth and Space Data Computing Division (ESDCD).

The NCCS has been directed to focus on use of Commercial Off-The-Shelf (COTS) products and "open" software that runs on more than one vendor's hardware platform. The Mass Data Storage and Delivery System currently uses UniTree software for hierarchical file storage management.

MDSDS Environment and Configuration

The MDSDS UniTree system became operational at the NCCS in October 1992. Since that time, MDSDS robotic storage has grown from the initial two StorageTek 4400 silos

with 2.4 TB total capacity (200-MB 3480 cartridge tapes) to 6 StorageTek silos with 28.8 TB capacity uncompressed (800-MB 3490E cartridges; in operation the MDSDS stores about 1 GB compressed data per cartridge). The MDSDS is currently adding an IBM 3494 robotic library with an additional 24 TB (10-GB 3590 tape cartridges).

The MDSDS UniTree software runs on a Convex C3830. While UniTree supports both NFS and ftp, access is limited to ftp to support throughput demands from the NCCS's supercomputers.

Local high-speed network connections join the MDSDS with the NCCS supercomputers, which are the primary sources and sinks for MDSDS data. The original Ethernet was augmented by an UltraNet/HiPPI connection in early 1993. By late 1994 the UltraNet/HiPPI connection had been replaced with two HiPPI/TCP connections, and FDDI was deployed to provide higher speed access to the rest of the NASA/Goddard campus.

The NCCS Supercomputers have undergone significant augmentation from the 4-processor Cray YMP (1.2 GFLOPs) in place October 1992. The current supercomputers are 3 Cray J90 systems, at present configured with 68 processors (13.3 GFLOPs) and to be upgraded to 96 processors (~19 GFLOPs) by spring of 1997.

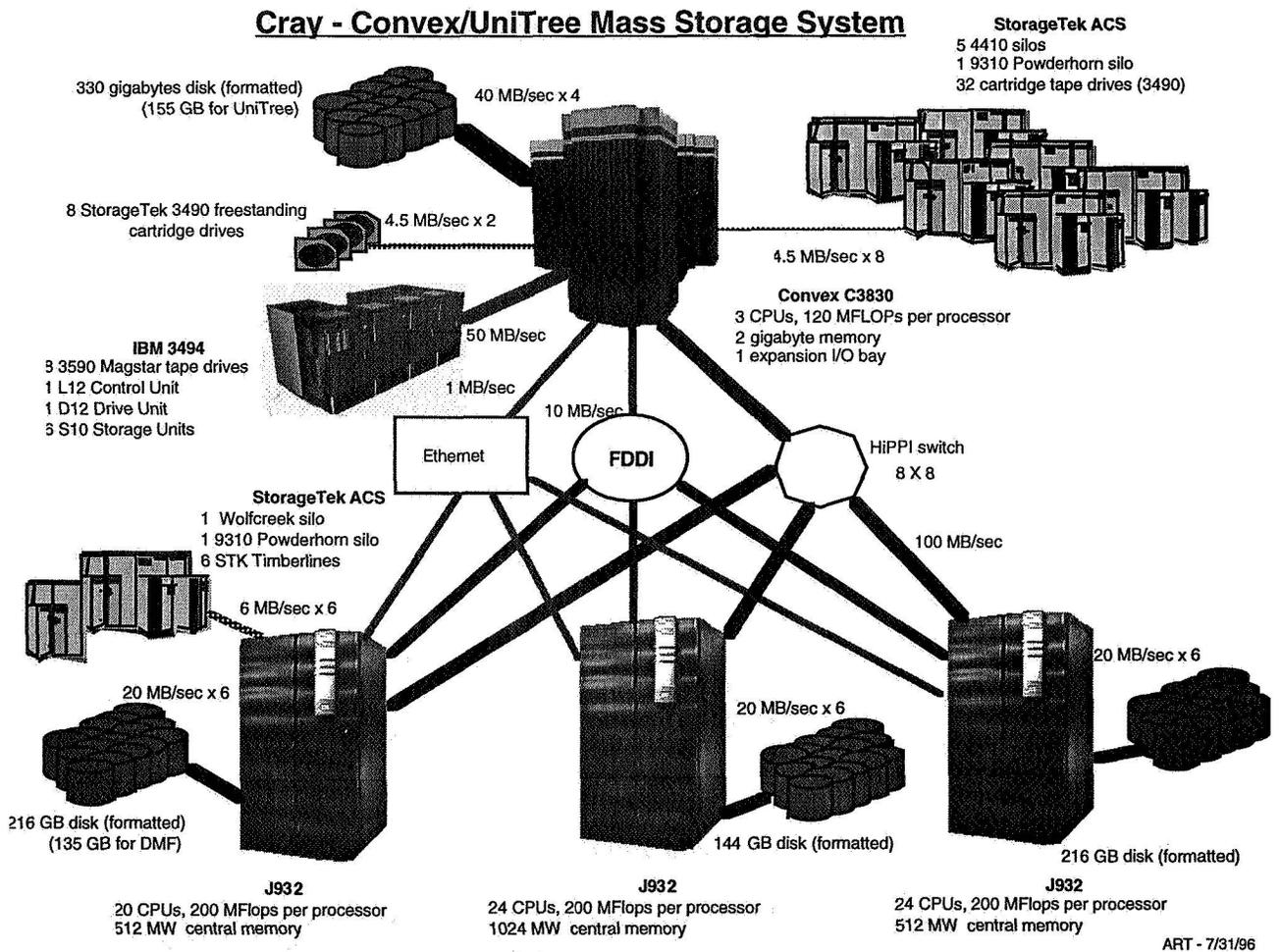


Figure 1. Cray-Convex/UniTree Mass Storage System

Requirements Input: Current MDSDS Usage Characteristics

Pentakalos et al. [1, 2] have studied and modeled the behavior of the NCCS UniTree system and Tarshish et al. [3, 4] have reported on its growth. In addition, ESDCD and NCCS staff maintain ongoing statistics to measure MDSDS usage and performance characteristics. The following characteristics are derived from ongoing observations and from some related studies.

Network Load

Much like a water plumbing system, a high-end storage and delivery system's performance must accommodate bursts that can be an order of magnitude or greater than rates averaged over time. For example, between June and August 1996, MDSDS aggregate network rates in excess of 17 MB/s have been observed in daily usage, whereas MDSDS average daily network traffic for June 1996 (71 GB/day) evenly distributed over an entire day would amount to 0.84 MB/s.

The middle of the working day is when the peak MDSDS request loads usually occur (cf. Pentakalos [2]) a characteristic common to other storage facilities (Behnke et al.[5]). It is especially typical for MDSDS traffic from sources other than NCCS supercomputers to peak during working hours. The non-supercomputer traffic to the MDSDS is significant (about 20-50 GB/weekday) but represents only about 23 % of all MDSDS traffic (*Figure 2*).

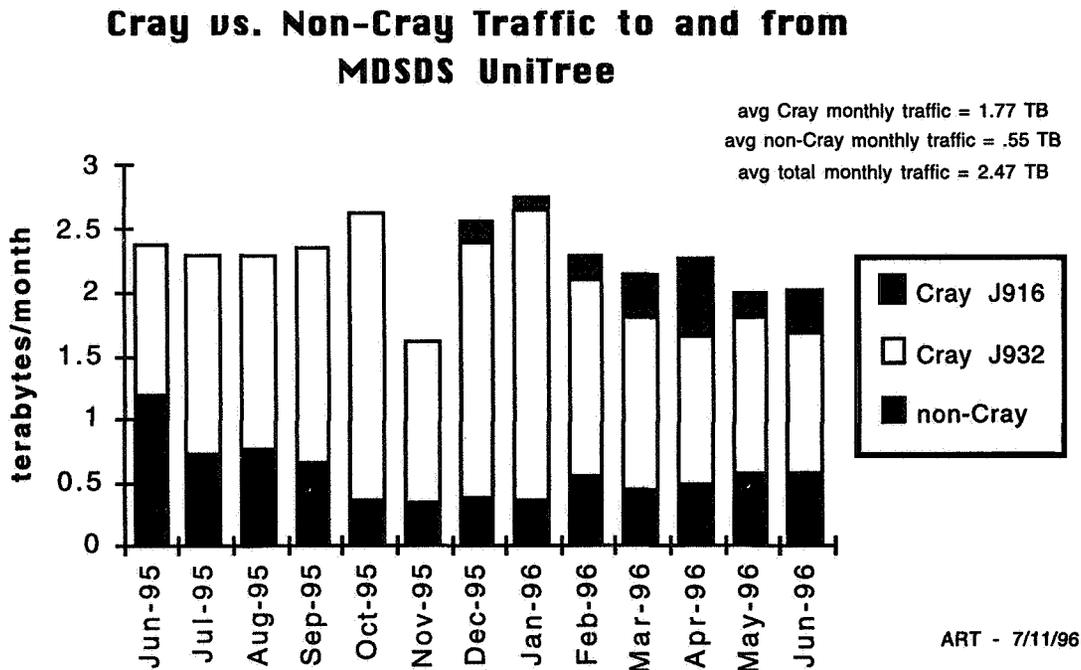


Figure 2. Cray vs. Non-Cray Traffic to and from MDSDS UniTree.

Relationship Between NCCS Supercomputing Resources Used and MDSDS Traffic

Historically, MDSDS transfer traffic has scaled approximately linearly with the increase in NCCS supercomputing CPU power once the user community has had time to adjust to new supercomputing technologies (*Figure 3*):

- In November 1993, shortly after delivery of the 6-processor Cray C90 (2.5 times the CPU power of previous 8-processor Cray YMP), a 30-day average for MDSDS traffic was ~36 GB/day.
- In late January 1996, Cray C90 usage was at its peak, and MDSDS traffic 30-day average reached ~101 GB/day.

The replacement of the 6 Cray C90 processors (1 GFLOP each) with 48 Cray J90 processors that were slower (0.2 GFLOP each) but much more plentiful resulted in some decrease in MDSDS traffic as users modified codes to improve throughput in the more parallel supercomputer environment. At this writing the third Cray J90 (20 additional processors) has been in place only a short time, but there are indications that MDSDS - supercomputer traffic is on the increase.

In addition, storage growth rates have tended to increase even if supercomputer CPU capacity stays the same for an extended period once users become accustomed to new supercomputing paradigms. As long as there are sufficient resources (both in available supercomputing capacity and in budgetary support), researchers tend to increase the resolution or complexity of models. In addition, the largest-volume users of the supercomputers tend to make ongoing refinements to code to maximize throughput.

*FY96 H2 is projected from first four months of the half

MDSDS Net Growth and New Data Added vs. Average Supercomputing GFLOPs

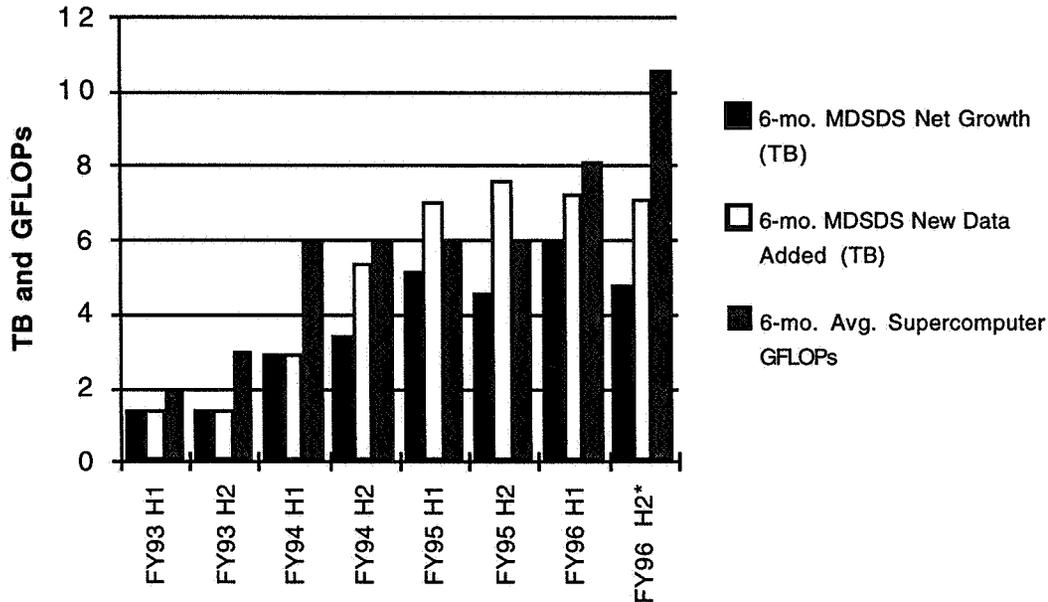


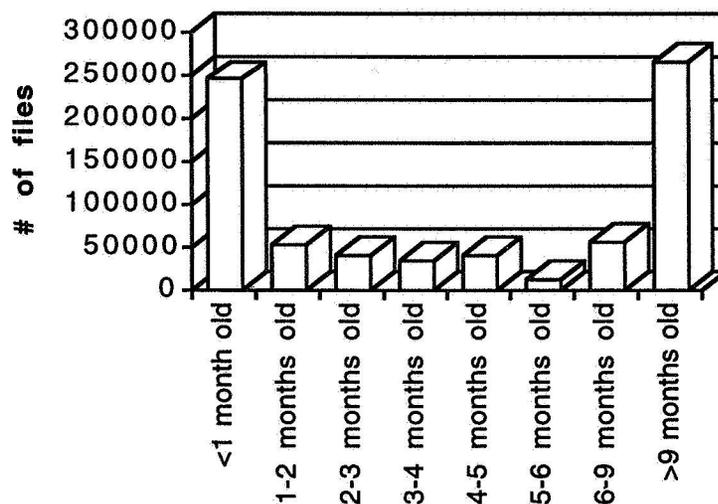
Figure 3. MDSDS Net Growth and New Data Added vs. Average Supercomputing GFLOPs.

Not a Black Hole for Storage: Retrieval Traffic Is Significant

As illustrated in the figures below, retrieve traffic is significant, so the MDSDS must be able to retrieve files quickly and efficiently. Clearly, the NCCS cannot afford to tune the MDSDS to optimize writing/storing at the expense of poorer performance for reading/retrieving files.

- NCCS users retrieve old files, not just recently created files (*Figure 4*).
- MDSDS users retrieve nearly as much data as they store (*Figure 5*). About 1.5 million MDSDS files were transferred between August 1995 and July 1996. 53% of the bytes transferred were stored, and 47% of the bytes transferred were retrieved. However, 62% of the *files* transferred were newly stored, and 38% were retrieved. This implies that on average, files retrieved are larger than files stored (averaging 24.8 MB and 17.1 MB respectively) and suggests that smaller files are somewhat less likely to be retrieved.

Age of MDSDS UniTree Files Retrieved Between 1/3/95 and 6/24/96

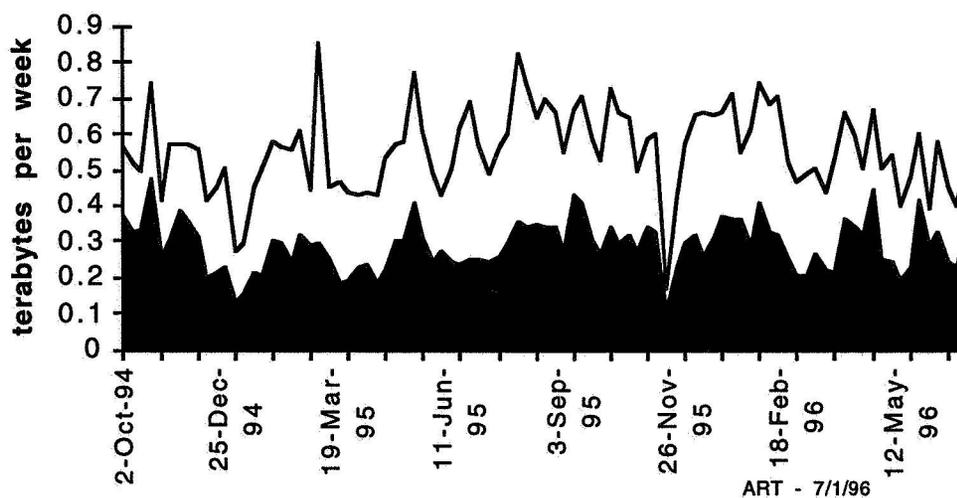


ART - 7/1/96

Figure 4. Age of MDSDS UniTree Files Retrieved Between 1/3/95 and 6/24/95.

avg stored = 42.23 GB/day avg retrieved = 29.21 GB/day (averaged over last 30 days)

Weekly MDSDS Data Transfer Traffic



ART - 7/1/96

Figure 5. Weekly MDSDS Data Transfer Traffic.

Working Set and Temporal Locality

The 3 months of MDSDS activity studied by Pentakalos [2] exhibited little temporal locality in the working set of the MDSDS. Six months during the period July 1995 through April 1996 were examined for the current study. The one-month working set during this period averaged about 1.8 TB, while total MDSDS traffic over one month averaged 2.6 TB.

- Of the 1.34 TB new data created in a month, on average only 0.25 TB (less than 1/5) would be retrieved in that same month; the remaining 0.48 TB of unique data retrieved would be more than 1 month old.
- On average 1.3 TB would be retrieved in a month; of that, 0.73 TB would be unique, so on average a given byte retrieved in a month would have been retrieved twice.
- However, 2/3 of the 0.73 TB unique data retrieved was retrieved only once, so the remaining 0.24 TB was responsible for all the repeated retrieves (0.86 TB of traffic). *Figure 6* shows the average proportions of bytes and files retrieved repeatedly over a 1-month period.

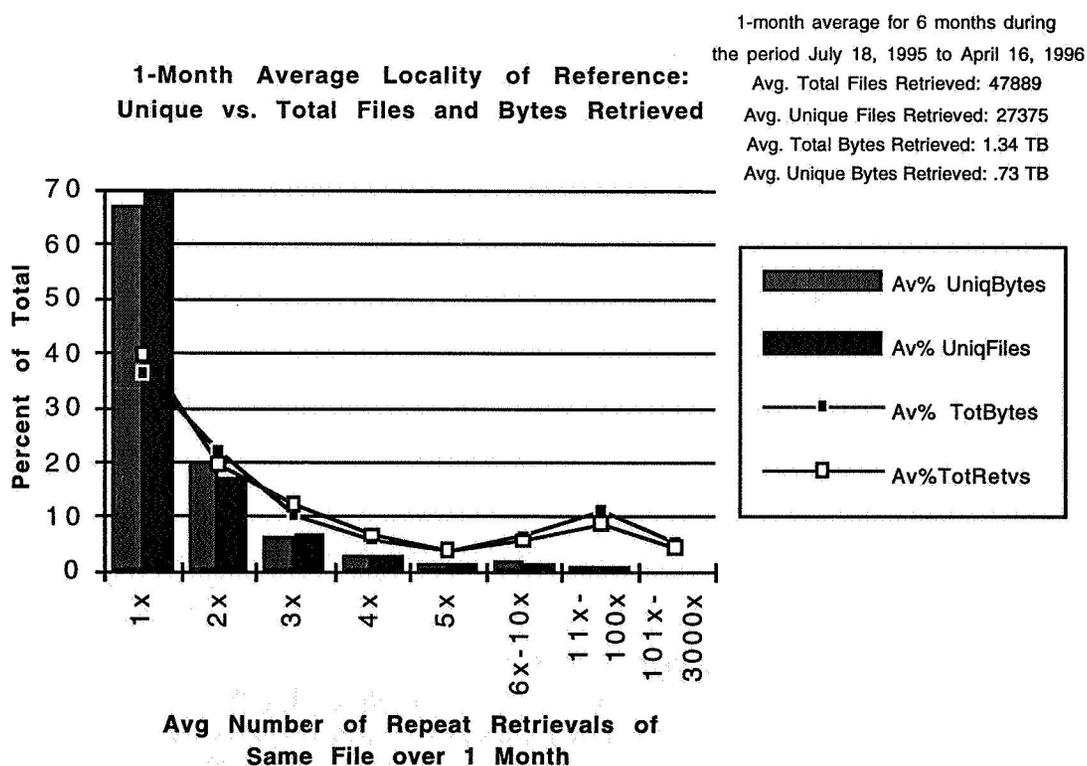


Figure 6. MDSDS 1-Month Average Locality of Reference: Unique vs. Total Files and Bytes Retrieved.

The likelihood that a file had been retrieved more than once increased when the time examined is expanded to six months: of the 10.3 TB retrieved during 6 months between

August 1995 and April 1996, 3.6 TB was unique data. The entire working set over 6 months was 9.5 TB. On average, at the time of retrieval:

- 1.2 TB (33%) of unique data was one month old or younger.
- 2.1 TB (58%) of unique data was 6 months old or younger; 1.5 TB (42%) of unique data was older than 6 months.

Figure 7 shows relative proportions of files and bytes retrieved repeatedly over the 6 months examined.

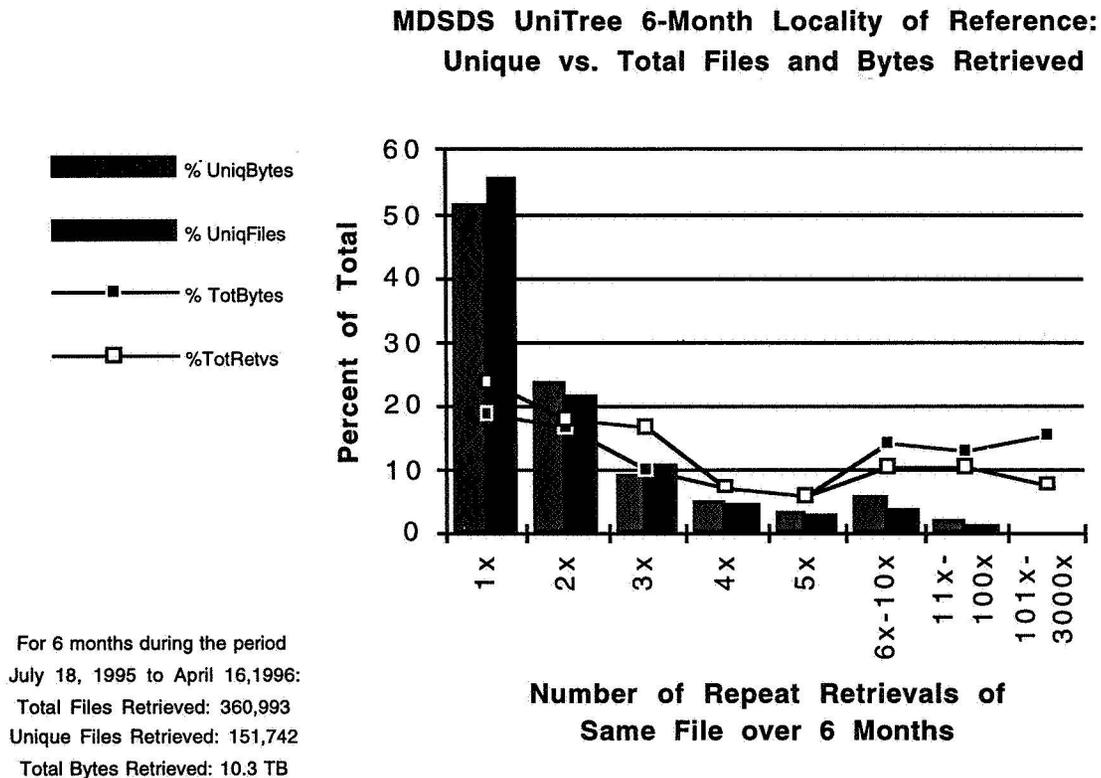


Figure 7. MDSDS UniTree 6-Month Locality of Reference: Unique vs. Total Files and Bytes Retrieved.

The MDSDS UniTree system's disk cache is configured to favor retaining most recently used files on disk, but at present there are no other reasonable means to set aside portions of the disk cache for special purposes (e.g., areas for files retrieved vs. files stored vs. files being moved to different tapes to consolidate free space on tapes). As a result, the cost to accommodate a RAIDed disk cache that holds six months' or even one month's working set would be prohibitive for the current machine architecture in the current budget climate. However, the re-use patterns may bear further to study to explore whether the near-online storage could be organized to optimize retrieval of the most frequently used data.

Distribution of File Sizes: Number of Files vs. Number of Bytes

While nearly a million *files* stored in the MDSDS system are 1 MB in size or smaller, the vast majority of the *data* stored in the MDSDS is in files of 50 MB and larger (Figure 8, Figure 9). This suggests that storage media with poor stop/start performance would not be suitable for the large number of small files, unless the file management software can compensate for small files, e.g., by grouping many small files together when writing, or by automatically directing small files to different media.

MDSDS File Distribution by File Size

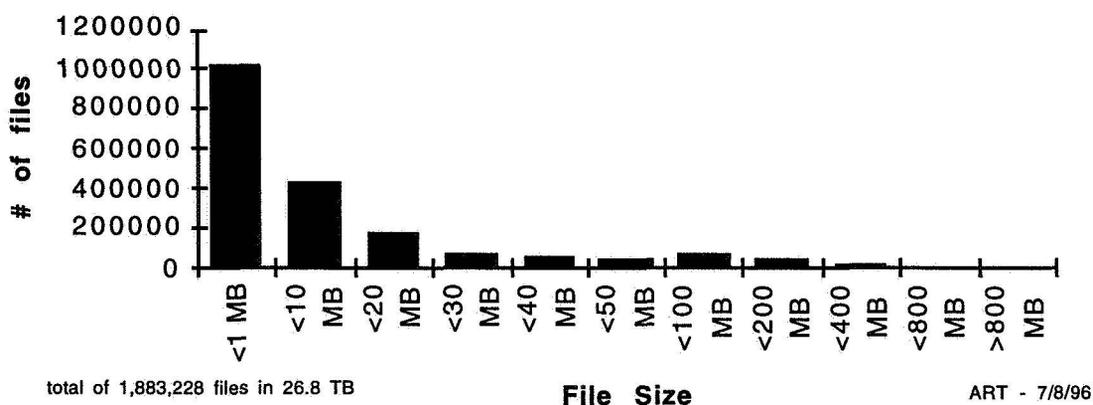


Figure 8. MDSDS File Distribution by File Size.

MDSDS Byte Distribution by File Size

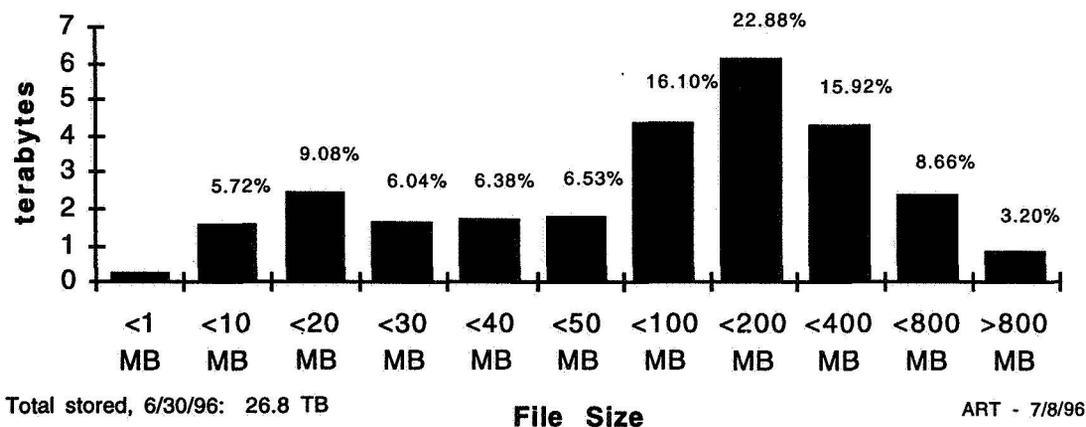


Figure 9. MDSDS Byte Distribution by File Size.

Repacking Is a Way of Life

MDSDS users delete nearly half a terabyte of data per month, an amount that approaches 1/3 the quantity of new data stored (*Figure 10*). Consequently, repacking (consolidating files from partly empty tapes to free those tapes for re-use) is a crucial activity. The I/O load from this tape repacking is significant and must be factored into the performance of the MDSDS system. Under the MDSDS's current release of UniTree software (Convex UniTree+ 2.0), at least 4 bytes must be moved for each byte copied from a tape being freed by repacking.

In addition, repacking is also the mechanism used to move MDSDS files to new storage media. This evolution to newer, more dense media is essential in order to (1) accommodate increasing volumes of new data and (2) ensure that older files continue to be readable as storage technology advances. In the 4 years since the MDSDS was deployed, there have already been 3 rounds of repacking to migrate files from 3480 cartridges (200 MB each) to 3490 cartridges (400 MB) to 3490E cartridges (800 MB). With the recent arrival of IBM Magstar tape drives, a new round of repacking will move some MDSDS data to the 10-GB IBM 3590 cartridges.

MDSDS Net Growth Rate, New Data, Deleted Data

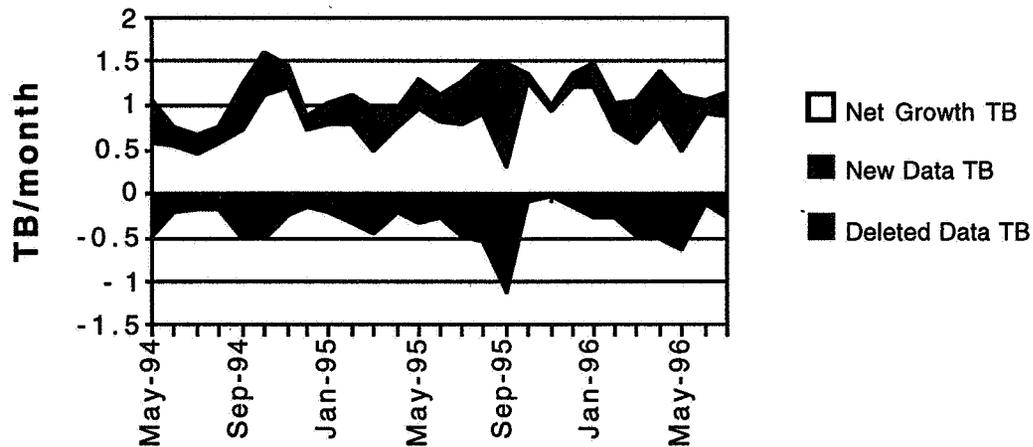


Figure 10. MDSDS Net Growth Rate, New Data, and Deleted Data.

Requirements Input: Some Projections for the Future

Several different sources have provided projected requirements information:

Near- and Medium- Term Research Program Changes:

The following changes are expected to have significant impact on MDSDS growth and volume of data transferred:

- Climate Data Assimilation researchers have announced plans to increase the climate model's vertical resolution and to save more diagnostic output starting in Fall 1996. These changes are expected to quadruple the amount of data produced by a climate model integration run.
- In FY98, the Climate Data Assimilation production work is currently slated to move off the NCCS supercomputers and onto EOSDIS-sponsored platform(s).
- A new Ocean Data Assimilation effort with projected requirements similar to Climate Data Assimilation production (CERRC [6]) is expected to begin (e.g., FY99 requiring 125 GFLOPs sustained and generating ~200 TB/year).

The NCCS MDSDS also anticipates providing long-term storage for the High Performance Computing and Communications (HPCC) program's Earth and Space Science (ESS) Cooperative Agreement effort, which will run through 1998-1999 (URL: <http://nccsinfo.gsfc.nasa.gov/ESS/>). Current plans show the Goddard Testbed machines and MDSDS connected via HiPPI, and perhaps later, via an ATM-to-HiPPI switch. ESS Grand Challenge Investigators who use those scalable parallel Testbed machines to be sited at Goddard are expected to store about 30 TB in the MDSDS system over the 3 years of the project.

Mission to Planet Earth Science-User Survey Results

In mid-1996, the Office of Mission to Planet Earth sent a survey to NASA Earth science researchers to help clarify the resources needed for computing and numeric modeling capabilities over the 5 years spanning 1997-2001. More than 200 researchers responded. Among the findings

- 35% wanted to double their model resolution (implies at least 4-fold increase in model output produced).
- 6% wanted to increase their model resolution by an order of magnitude (could easily increase model output by more than 2 orders of magnitude).
- 88% had requirement for access to high-speed networking and/or mass storage

NCCS Science Research Users: the CERRC Report

In January 1995 the Computer Environments and Research Requirements Committee (CERRC) reported on NCCS Earth and space science computing requirements for the years 1997-2004 after gathering information from researchers who use NCCS resources (CERRC [6]). The CERRC obtained input from both Goddard- and NASA-based researchers, and those at universities and non-NASA institutions (the latter use about

27% of NCCS resources at present). The CERRC report predates some of the recent Federal budget reduction exercises, and it describes computing requirements that presume the science investigations would be funded at reasonably favorable levels.

Briefly, the CERRC report relates the need for supercomputing CPU performance of 125 GFLOPs sustained by 1999 and 1 TFLOP sustained by 2004. The space and Earth Sciences research codes that drive these CPU performance requirements are expected to generate the need for 400 TB in robotic storage in 1997, and 2000 TB robotic storage by 1999.

NCCS Planned Supercomputer Upgrades

In response to the requirements detailed in the CERRC report and budget direction from management, the NCCS currently expects to make available to users increased supercomputing CPU power along the following lines:

- FY97: complete the 3-fold increase (to 19.2 GFLOPs) compared to FY95's 6 GFLOP Cray C90.
- FY99: an additional increase of nearly 5-fold to ~90 GFLOPs.
- FY00 and beyond: continuing incremental augmentations, as budgets and supercomputing technology costs allow.

Storage and Network Requirements

The CERRC report and ongoing observations of MDSDS usage contained the most concrete information on trends and projections, and so are the primary sources for the requirements presented here. As with all projections for the future, the results are only as good as the assumptions and initial data, so the NCCS monitors usage, program direction, and industry to revise and update the picture. Caveats aside, the results from requirements-projecting exercises have proven useful in resource and budget planning.

Sustained network rate requirements were derived directly from the CERRC report's [6] data traffic figures cited for the milestone years (500 GB to 1 TB daily data traffic in FY97 and 1-2 TB daily traffic by FY99). In FY97, this leads to the need for sustained bandwidth of 6-12 MB/s; FY 99 would require 12-24 MB/s sustained. Peak network bandwidth requirements incorporated the empirically observed need to accommodate burst rates an order of magnitude higher than sustained loads.

Growth rates for interim years were calculated 2 ways: in the conservative method, the growth rate is tightly coupled to the supercomputing power, and so remains constant in a year (such as FY98) in which there are no supercomputer augmentations. The "heavier traffic" estimate assumes that growth rates will continue to increase in the interim years (due to increases in resolution, complexity, and optimization for throughput, as noted above).

Total data stored (based on conservative and heavy growth rates) and the robotic capacity recommendations from the CERRC report are presented in *Figure 11*. Projections for growth rates and peak bandwidth are presented alongside expected increases in supercomputer power in *Figure 12*.

MDSDS Robotic Storage Forecasts

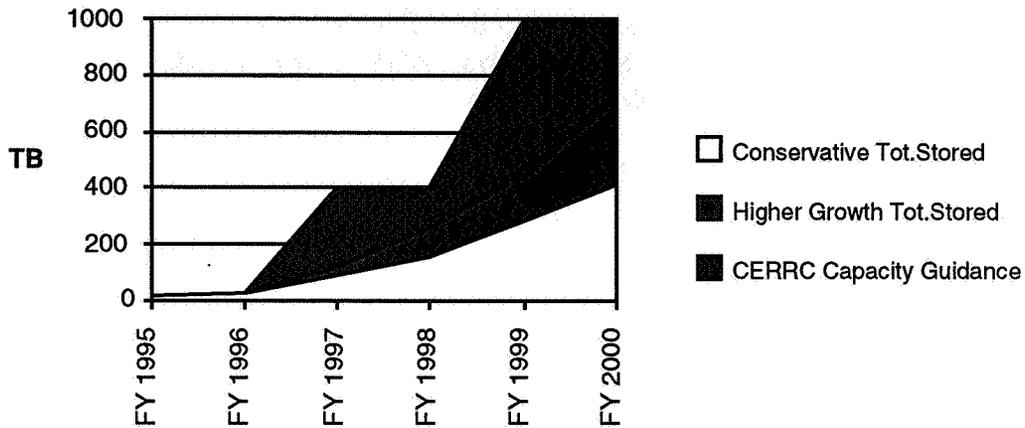


Figure 11. MDSDS Robotic Storage Forecasts.

MDSDS Required Bandwidth and Projected Data Added per Year, 1997-2000

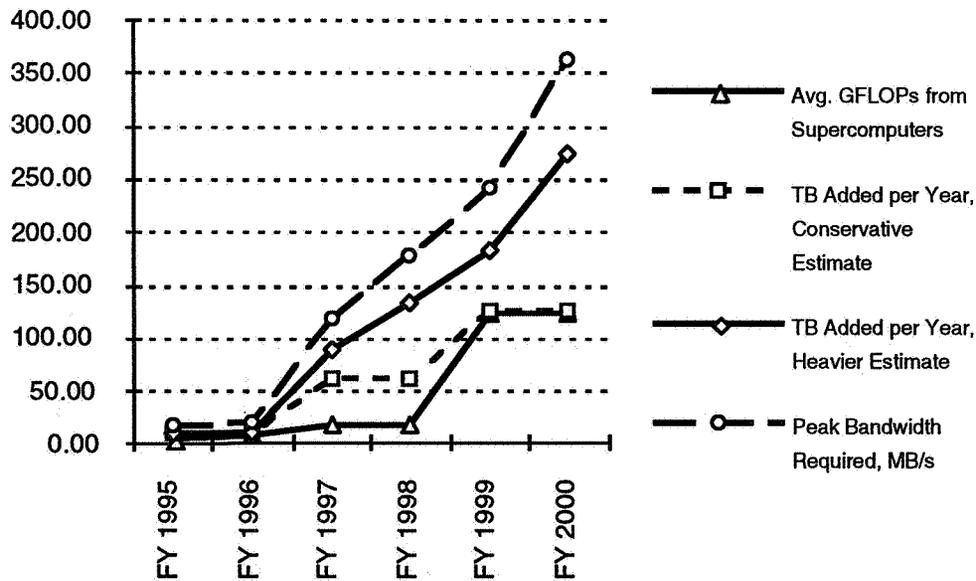


Fig. 12 MDSDS Required Bandwidth and Projected Data Added per Year, 1997-2000.

Acknowledgments

Many thanks to Adina Tarshish for her hard work and tenacity in gathering MDSDS statistics and creating charts. Thanks also to Nancy Palm, Michele Rienecker, Tom Schardt, Bob Theis, Pat Gary, and the authors of the CERRC report for their efforts at synthesizing abstract plans into concrete requirements.

References

- [1] O. Pentakalos and Y. Yesha, "Evaluating the Effect of Online Data Compression on the Disk Cache of a Mass Storage System," *Proc. of the Fourth Goddard Conference on Mass Storage Systems and Technologies*, pp. 383-391, 1995.
- [2] O. Pentakalos, D. Menasce, M. Halem, and Y. Yesha, "An Approximate Performance Model of a UniTree Mass Storage System," *Proc. of Fourteenth IEEE Symposium on Mass Storage Systems*, pp. 210-224, 1995.
- [3] A. Tarshish and E. Salmon, "The Growth of the UniTree Mass Storage System at the NASA Center for Computational Sciences," *Proc. of Third Goddard Conference on Mass Storage Systems and Technologies*, pp. 179-185, 1993.
- [4] A. Tarshish and E. Salmon, "The Growth of the UniTree Mass Storage System at the NASA Center for Computational Sciences: Some Lessons Learned," *Proc. of Fourth Goddard Conference on Mass Storage Systems and Technologies*, pp. 345-357, 1995.
- [5] J. Behnke and J. King, "NSSDC Provides Network Access to Key Data via NDADS," *Proc. of Fourth Goddard Conference on Mass Storage Systems and Technologies*, pp. 359-381, 1995.
- [6] The Computer Environments and Research Requirements Committee, "NASA Earth and Space Science Computing Requirements 1997-2004," 56 pp., 1995.

521-82

83203

NASDA's Earth Observation Satellite Data Archive Policy for the Earth Observation Data and Information System (EOIS)

Shin-ichi Sobue*, Osamu Ochiai, and Fumiyooshi Yoshida

NASDA EOC

1401 Numanoue, Hatoyama, Saitama 350-03, Japan

Tel: 81-492-98-1200

Fax: 81-492-98-1001

Abstract

NASDA's new Advanced Earth Observing Satellite (ADEOS) is scheduled for launch in August, 1996. ADEOS carries 8 sensors to observe earth environmental phenomena and sends their data to NASDA, NASA, and other foreign ground stations around the world. The downlink data bit rate for ADEOS is 126 MB/s and the total volume of data is about 100 GB per day. To archive and manage such a large quantity of data with high reliability and easy accessibility it was necessary to develop a new mass storage system with a catalogue information database using advanced database management technology. The data will be archived and maintained in the Master Data Storage Subsystem (MDSS) which is one subsystem in NASDA's new Earth Observation data and Information System (EOIS). The MDSS is based on a SONY ID1 digital tape robotics system. This paper provides an overview of the EOIS system, with a focus on the Master Data Storage Subsystem and the NASDA Earth Observation Center (EOC) archive policy for earth observation satellite data.

Introduction

The NASDA Earth Observation Center (EOC) is developing a new Earth Observation data and Information System (EOIS) to archive and distribute level 0 and processed data and information related to Japanese (MOS, JERS and ADEOS) and foreign (LANDSAT, SPOT and ERS) earth observation satellites. This paper provides an overview of the Master Data Storage Subsystem (MDSS) and NASDA's EOC data archiving policy. The MDSS archives processed data and is based on a SONY ID1 tape robotics system and FDDI networks.

* Present address: NASA Goddard Space Flight Center
Greenbelt, Maryland 20771
sobue@eos.nasa.gov
Tel : 301-286-0148
Fax : 301-286-0267

NASDA EOC Data Archive Policy

The EOC, NASDA's primary data center, receives, records, processes, archives, and distributes earth observation satellite data and related information from Japanese and foreign EO satellites. NASDA's EOC has maintained raw data using HDDT (High Density Digital Tape) and processed data using 9-track magnetic tape since NASDA's EOC was established. Maintaining the present archive of 50,000 magnetic tapes is cumbersome and expensive. Thus, NASDA's EOC is developing the MDSS utilizing ED1 digital tapes.

Master Data Storage Subsystem Overview

The MDSS (see Figure 1) consists of three SONY ID1 digital tape recorders, an ID1 digital tape library (file bank system), robotics, work stations, and an FDDI network. A midsize ID1 digital tape holds 36 Gigabytes which is 300 times more than an MT. The MDSS archives 736 ID1 midsize tapes which is equivalent to 220,800 magnetic tapes (see Figure 2). The archiving cost for ED1 tapes is around 0.75 dollar/GB which is 270 times cheaper than magnetic tape. Thus, NASDA's EOC adopted ID1 tapes to archive level 0 and processed data. The physical format for ID1 tapes is the SONY ID1 special format, the logical format of level 0 data is NASDA's special format and the logical format of processed data is HDF or CEOS superstructure.

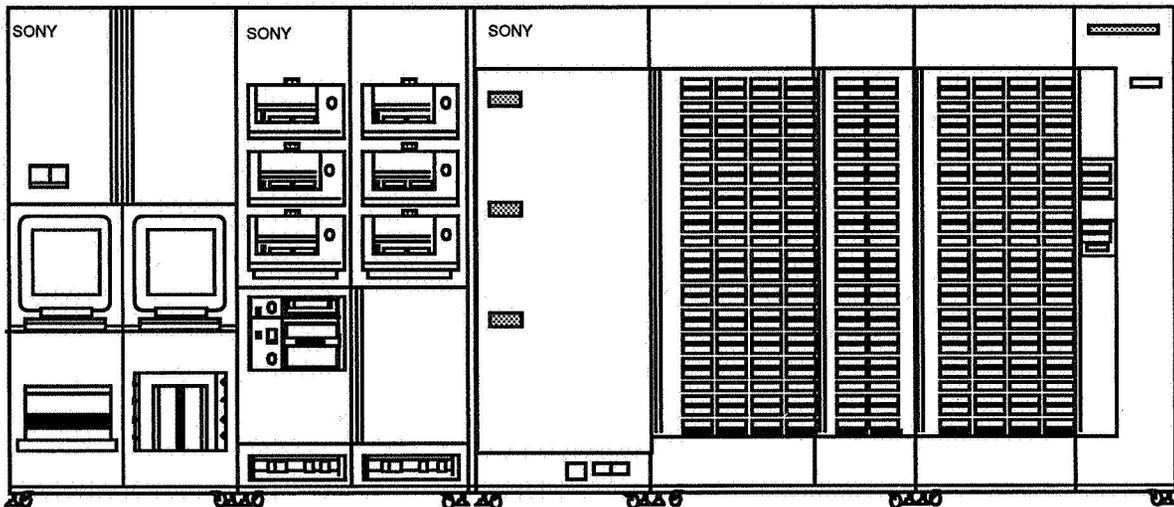


Figure 1. Master Data Storage Subsystem (MDSS)

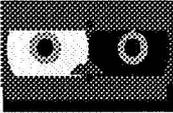
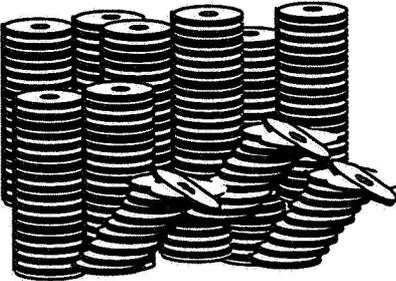
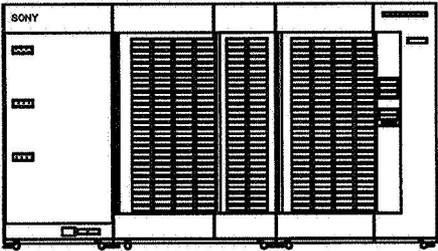
36 GB	
CCT (1/2 inch open reel type)	ID1 cassette (Formatted)
 <p style="text-align: right;">300</p>	<p style="text-align: center;">Midsized D1 Tape</p>  <p style="text-align: right;">1</p>
27 TB	
CCT Storehouse	Master Data Storage Subsys.
 <p style="text-align: right;">220,800 ? m²</p>	 <p style="text-align: right;">736 3.4m²</p>

Figure 2. Improved Storage Space Utilization

In response to user requests, within 10 minutes the MDSS can simultaneously transfer 3 different 100 MB files to EWS (SONY NEWS) buffer disks. After staging a 100 MB data file (one granule of data), the file is transferred from a NEWS to a client subsystem such as the Media Conversion Subsystem (MCS) or the Film Generation Subsystem (FGS) using the EOIS communications protocol. The MDSS is also capable of ingesting existing processed data from magnetic tapes, converting the format and writing the data to ID1 tapes.

Master Data Subsystem Development

NASDA's EOC began developing the MDSS in March, 1994 and will complete development by the end of 1996. The MDSS will archive data not only for existing satellites (JERS-1, MOS-1/1b, LANDSAT, SPOT and ERS) but also new Japanese EO satellites (ADEOS, the Advanced Earth Observing Satellite, will be launched in August, 1996). In addition, NASDA's EOC is also developing other EOIS subsystems such as the catalogue subsystem, browse data distribution subsystem, data distribution subsystem, media conversation subsystem and the ADEOS receiving, recording and processing subsystem for ADEOS and existing satellite operations.

Conclusion

Starting from the end of 1996 NASDA's EOC will archive all earth observation satellite data on ID1 digital tapes. NASDA's EOC is also studying advanced data base management techniques using object oriented technology to integrate the MDSS, which uses ID1 data recorders, with other newly developed EOIS subsystems such as the Browse Data Distribution Subsystem (BDS) which uses RAID disk and Magnetic Optical Jukebox technology, and the Catalogue Subsystem (CATS) which uses RAID disk storage.

Progress in Defining a Standard for File-Level Metadata

Joel Williams
Systems Engineering and Security
7474 Greenway Center Drive
Greenbelt MD 20770
joel.williams@ses-inc.com
Tel: 301-441-3694
Fax: 301-441-3697

522-82
83204

Ben Kobler
NASA/GSFC
Greenbelt MD 20771
ben.kobler@gsfc.nasa.gov
Tel: 301-614-5231
Fax: 301-614-5267

Introduction

In the following narrative, metadata required to locate a file on a tape or collection of tapes will be referred to as *file-level metadata*. This paper describes the rationale for and the history of the effort to define a standard for this metadata.

The Problem

Extremely large data systems, such as the Earth Observing System Data and Information System (EOSDIS), must rely on hierarchical File Storage Management Systems (FSMS) to stage files to disk as required for fast access, and to migrate files to tape for more economical storage when there is no requirement to keep them on disk. There is no standard format for such files when they are moved to tape, and so each FSMS uses a proprietary format. Files, particularly those which have been updated frequently, may be scattered over several tapes, and the information required to reconstruct the files is likely to be stored on disk separate from the tapes on which the files reside. Some file-level metadata information may be embedded as header information on each block on the tapes, so that any program reading the file would have to identify this header and understand that it is not really part of the file.

Changing from one FSMS to another would therefore most likely require the re-writing of all of the tape files written by the original system. For a large archive this would be extremely expensive.

Initial Analysis

This situation has been analyzed in the paper dated March 15, 1995, *An Assessment of Requirements, Standards, and Technology for Media-Based Data Interchange* by David Isaac and Dana Dismukes of the MITRE Corporation. The work was funded by the

Goddard Space Flight Center (GSFC) Earth Observing System Data and Information System (EOSDIS) Project. In the paper the following conclusions were made:

- Standards for media-based data interchange could save EOSDIS approximately \$2M per storage system migration by reducing the need for additional computing capacity to support copy operations.
- While there was no current standards activity addressing the problem, there was sufficient interest in the customer and vendor community to support such an activity.
- While the requirement to refresh media as it ages somewhat reduces the potential for cost savings from media-based data interchange, it does not eliminate it.

In order to avoid the copy operation of re-writing an extensive tape archive when transferring tapes from one FSMS to another, there must be a standard way of transferring the file-level metadata. This metadata needs to contain sufficient information to enable the receiving system to reconstruct the file system represented by the tapes. Transferring this metadata would enable the receiving system to incorporate the tapes with a minimum of effort.

In this context, we are not concerned with the semantics of the information contained in the files themselves. We are only concerned with the information required to identify the file (for example its name) and to associate it with one or more delimited bytestreams on one or more tapes. The bytestreams themselves would have to be ordered, as in the case of multi-reel files or striped files.

Initial Proposal for a Standard Tape Format

Encouraged by the MITRE study, the NASA GSFC EOSDIS project asked Joel Williams (then of the MITRE Corporation, currently of Systems Engineering and Security, Inc.) to develop a *Straw Man* standard and to gauge the reaction of the vendor and user community to this standards effort and to the proposed *Straw Man* standard itself.

The *Straw Man* standard was a tape format standard. The fundamental concept of the standard was to put a directory on each tape of the archive so that by reading the directory an application could determine where the files or file segments on the tape were located. The *Straw Man* was inspired by two proposed standards which include on-tape directories, the DD1 (ISO/IEC CD 14417) and the DD3 (ANSI X3.267) standards, and also by the EMASS practice of placing a directory on D2 tapes. In addition, during this same time period when the *Straw Man* was being developed, IBM announced its Magstar product, which has a directory at the beginning of the tape. Subsequently, Sony has announced a tape which has a directory on a chip on the tape cassette.

The *Straw Man* proposal was for a logical tape format, and would have been written at the application (FSMS) level. It could therefore apply to any tape technology, although it did require partitioning of the tape in order to be able to update the directory without having to re-write the entire tape.

There are two different types of on-tape directories: those that contain information about the file (such as its name, for instance) and those that primarily contain information allowing the fast positioning of the tape.

The DD3 standard, as outlined in Figure 1, is of the second type. It allows one to position the tape quickly, but the (file name, position) mapping must be done at a higher level.

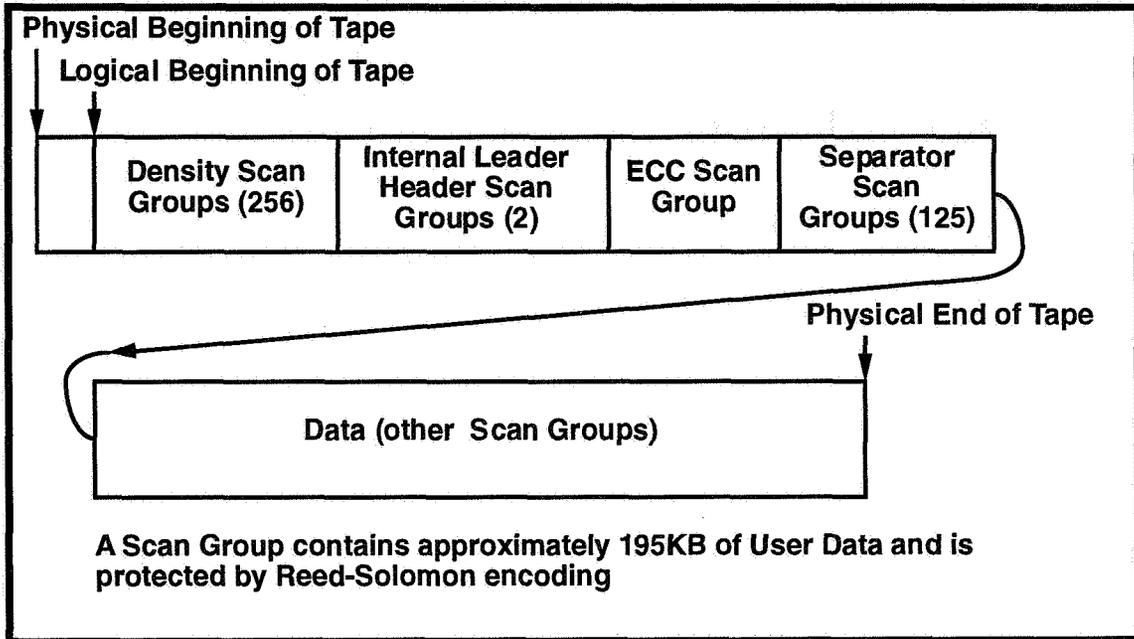


Figure 1

Each of these scan groups contains approximately 195KB of user-written data, including end-of-record markers, but exclusive of error correcting codes. The Internal Leader Header scan groups are reserved for directory information, and whenever the tape is mounted, they are read into the drive memory, then modified before the tape is dismounted. It contains information that allows for fast positioning of the tape, and additional information such as

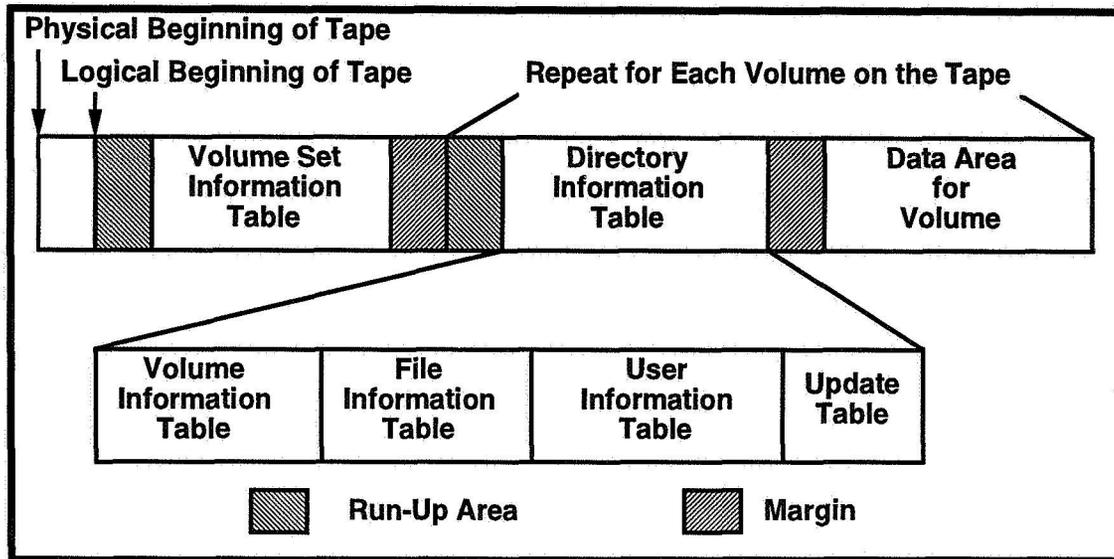


Figure 2

the volume id, the number of mounts, the time and date of the last five mounts, and the tape manufacturer.

The Magstar directory provides similar functionality, and also includes extensive information concerning any errors which may have happened when the tape has been accessed.

The DD1 proposed standard is outlined in Figure 2. It contains all of the file-level metadata which would be required to locate a file on the tape, given the file name.

The following blocks are defined:

- The Volume Set Information Table. This is at the beginning of the tape, and contains information on the number of volumes contained on the tape, and identifies the cassettes in a volume set. This information supports files which are striped across different tapes. There is only one Volume Set Information Table on each tape.
- The Directory Information Table. There is one of these for each logical volume on the tape, and it consists of the following four blocks:
 - The Volume Information Table, which describes the volume
 - The File Information Table, which contains information used for positioning the tape to files in the Data Area
 - The User Information Tables, which contain information on each file in the following Data Area, such as the file name, version number, creation date, etc.
 - The Update Table, which is used to ensure that the directory has been updated properly.

- The Data Area, which contains the files in the volume described in the Directory Information Table.

The *Straw Man* proposal looks very similar to the DD1 proposal, and is outlined in Figure 3.

In addition to information such as the file name and its location, the directory contains the following information:

- Pointer to the next file segment if the file is continued
- Pointer to the first file segment if the file does not begin in this location
- Pointers to the other stripes if the file is striped

In this way multi-reel files and striped files are supported.

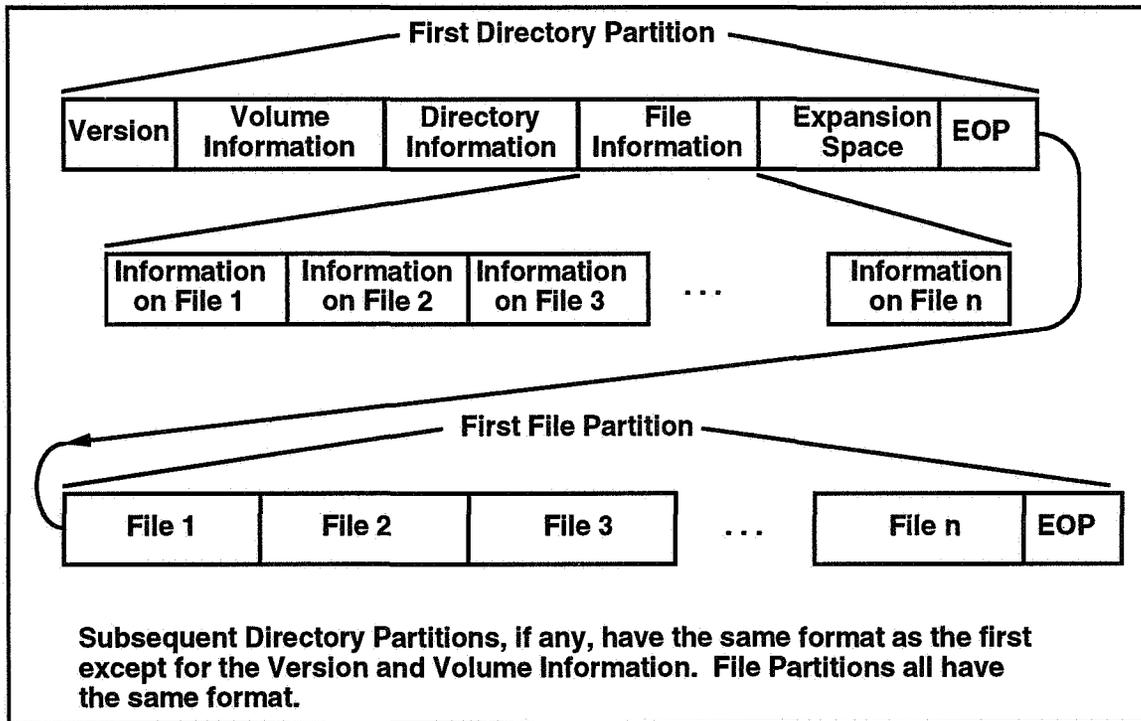


Figure 3

Presentation of the *Straw Man* and Reactions to it

This *Straw Man* proposal was first circulated at the Fourteenth IEEE Symposium on Mass Storage Systems at Monterey, California in September, 1995. Subsequently, it was briefed to THIC, the ISO/CCSDS Archiving Workshop at GSFC, the ANSI X3B5 Committee, the AIIM Optical Tape Study Group, and to individuals at the National

Security Agency. Several changes were made to the original proposal to lower the overhead of having to re-write the directory whenever the tape was updated.

The decision was made to form a Study Group under the auspices of the Association for Image and Information Management (AIIM). The group's name is the *File-Level Metadata for Portability of Sequential Storage (FMP) Study Group*, and the first meeting of the group was on April 1 at the AIIM International Convention in Chicago Illinois. The group is chaired by Fernando Podio of the National Institute of Standards and Technology (NIST).¹

The First FMP Meeting

The first FMP meeting was held April 1-2 in Chicago. The following organizations were represented:

- Ampex
- Applicon
- Datatape
- EMASS
- Fermilab
- NASA GSFC
- LDS Church
- HPSS
- Kofax Image Products
- Lawrence Livermore National Lab
- Legacy Data Systems
- Library of Congress
- Los Alamos National lab
- Lots Technology
- LSC Inc.
- Micro Design International
- MITRE
- National Media Lab
- NIST
- Research Libraries Group
- Systems Engineering and Security
- Storage Technology
- Terabank Systems

The *Straw Man* proposal was presented at this meeting, and various other presentations were made. The consensus of the group was that an on-tape directory containing file-level metadata was impractical for performance reasons. There was general agreement,

¹ For further information about the FMP study group, contact Fernando Podio at fernando.podio@nist.gov (Fernando Podio)

however, that there needed to be a standard for the export of file-level metadata, and that it was advantageous to work toward that standard under the auspices of AIIM.

In this regard, the group agreed to a statement of work as follows:

The AIIM FMP SG will document an interchange format for file-level metadata for data stored on sequential storage media. This approach does not concern the data format on the physical media or drive.

Figure 4 graphically depicts how this interchange standard would work.

The original system would of course maintain its own metadata in some form, which could remain proprietary. This metadata would enable it to manage the tapes under its domain. When it came time to migrate these tapes to a new system, the original system's metadata would be exported to the public standard. The new, receiving system would read this standard metadata and convert it to its own representation, which might also be proprietary. In this way, the new system would be able to take over the management of the tapes without re-writing them.

The major challenge in developing this standard for file-level metadata export is to develop something that is broad enough to cover current and anticipated practice. Cooperation from the vendor community will be important in meeting this goal.

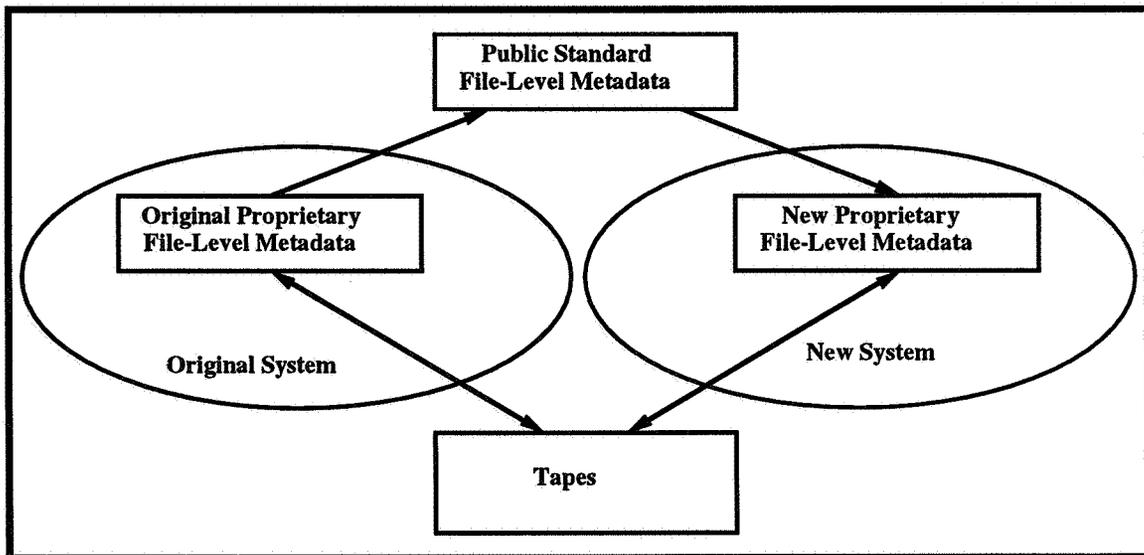


Figure 4

The Second FMP Meeting

The second meeting of the FMP Study Group occurred June 17-18 at the AIIM headquarters in Silver Spring, Maryland. The following organizations were represented:

BDM
Datatape
Department of Defense
EMASS
NASA GSFC
Hewlett Packard
HPSS
IBM
Lawrence Livermore National Lab
Lots Technology
LSC Inc.
IIT Research
National Media Lab
NSA
NASA Langley
NIST
Norsam
Systems Engineering and Security
Storage Technology
Terabank Systems

Discussions at this meeting centered on determining the data elements which constitute the file-level metadata required to be exported. These elements, it was determined, fall into three categories. The lists below characterize and contain examples from each category which were discussed at the meeting.

- Data elements having to do with the tapes
 - Tape ID
 - Tape Universally Unique Identifier (UUID)
 - Tape model or type
 - Statistics about errors on the tape
 - Compression information
 - Exporting FSMS and vendor name, operating system version, hardware identification

- Data elements having to do with the files on the tapes
 - File Name (including version information, if any)
 - File Universally Unique Identifier (UUID)
 - Method of tape addressing

- Location of file segments, including *magic cookie* information if it exists
 - Striping information
 - Information identifying multiple copies of the file
 - Account IDs for billing purposes
 - File Family
 - Tape set
 - Volume group
- Data elements having to do with the file system represented by the tapes
 - Directory and file structure
 - Hard and soft links
 - Principal names and groupings for security purposes

The next meeting of the FMP Study Group is October 1-2 at the AIIM Headquarters, 1100 Wayne Ave. in downtown Silver Spring, Maryland.

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 September 1996	3. REPORT TYPE AND DATES COVERED Conference Publication	
4. TITLE AND SUBTITLE Fifth Goddard Conference on Mass Storage Systems and Technologies - Volume I			5. 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. PERFORMING ORGANIZATION REPORT NUMBER <i>Rept-96B00117-Vol-1</i>	
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: I	
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 312	
			16. PRICE CODE	
17. SECURITY CLASSIFICATION OF REPORT Unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT Unclassified	20. LIMITATION OF ABSTRACT UL	

National Aeronautics and
Space Administration

Goddard Space Flight Center
Greenbelt, Maryland 20771

Official Business
Penalty for Private Use, \$300

SPECIAL FORTH-CLASS RATE
POSTAGE & FEES PAID
NASA
PERMIT No. G27



POSTERMASTER: If Undeliverable (Section 158,
Postal Manual) Do Not Return
