MaSSIVE™: The Mass Storage System IV Enterprise

J. L. SLOAN, B. T. O'LEAR, D. L. KITTS, AND B. L. IRWIN

Invited Paper

The Mass Storage System IV Enterprise is a fourth-generation mass storage system being designed at the National Center for Atmospheric Research. It will provide file system service to teraflops computing systems appropriate to Grand Challenge problems. A bitfile managed by MaSSIVE™ can be a complete, self-contained UNIX™ file system specific to a particular MaSSIVE™ client. MaSSIVE™ will stage whole file systems from archival storage onto online storage devices and then provide its clients with raw block access to the staged file system. It will comply with the IEEE Mass Storage System Reference Model, and will consist of co-operating processes distributed across a network of supermicrocomputers. The client interface to a file system will be implemented through a custom device driver that will permit I/O to MaSSIVE™ online storage devices. The device interface through a high bandwidth data fabric to the storage device will use common device controller-specific protocols. As file systems are unmounted by clients or clients disconnect from the data fabric, file systems may be migrated back to archive storage and removed from online storage. Prototypes for portions of MaSSIVE™ are under development.

I. Introduction

The Scientific Computing Division (SCD) at the National Center for Atmospheric Research (NCAR) in Boulder, Colorado, is developing a next-generation mass storage system (MSS) that will eventually replace the existing mainframe-based NCAR MSS. The design of the new system is driven by the challenge of the changing nature of high-performance computing. It meets this challenge by exploiting technologies emerging from the changing nature of high-speed and high-capacity storage.

The terms *high performance*, *high-capacity*, and *high-speed* imply different things in different contexts. This paper considers the high-end: gigaflops or teraflops of compute power, data transfer rates approaching or exceeding a gigabit per second, and storage capacities from terabytes to petabytes.

Manuscript received March 19, 1992; revised November 9, 1992.
The authors are with the National Center for Atmospheric Research, Scientific Computing Division, Boulder, CO 80307-3000.
IEEE Log Number 9208736.

The Data Problem

Supercomputers, data storage systems, high-speed networks, and output facilities form the backbone of the high performance computing environment. In all computing environments, performance is limited by the lowest common denominator. In high performance environments, idle supercomputer cycles translate into significant dollars lost. Hence, providers of such environments become experts in *bottleneckology*, the science of identifying and eliminating the lowest common denominator. To these people, the most scandalous four letter word has always been *data*: supercomputing stresses its own ability to consume it, to generate it, to move it, and to store it. Among the most stressful applications for supercomputing are the so called *Grand Challenge* problems [1].

Among the Grand Challenges is the problem of modeling global climate changes. It is currently infeasible to conduct controlled experiments with actual planetary climates. Programmatic models must be developed to simulate the climate as closely as possible in a time domain faster than our own. These models often have extensive requirements for data consumption, generation, and storage. It is more economical to store output data rather than regenerate it. This is not expected to change, despite improvements in computing technology. Because the true value of such output data may not be realized until well into the future, when it can be compared with observational data, it is often necessary to reliably and efficiently archive model output data for decades.

SCD has empirically observed that NCAR's global climate modeling user community archives about a half a million bytes (or four million bits) for every billion floating point operations performed, regardless of the computing technology used. For example, sustained performance of 0.5 gigaflops (billion floating point operations per second) on NCAR's CRAY Y-MP™ 8/864 supercomputer generates 0.25 megabytes of data per second. At that rate, 7.5 terabytes per year of model output data must be archived. A computer capable of sustaining one teraflops could generate

nearly fifteen petabytes ($15x10^{15}$) of data each year. It is precisely these teraflops computing engines which will be necessary to solve Grand Challenge-class problems such as global climate modeling.

Computers capable of teraflops performance are very near to being available to organizations charged with solving Grand Challenge problems. These computers are likely to overwhelm the current mass storage capabilities of those sites. The data problem is compounded by the need to provide a scalable mass storage system that presents as few single points of failure as is feasible, and that can be economically and incrementally expanded as new and additional high performance computers are added to the mass storage environment.

Data Plumbing

The selection of storage, networking and computing technologies in high performance environments is driven by frequently conflicting requirements: to leverage off industry standards where ever possible, to find and eliminate the lowest common denominator in performance, and to make the most cost effective use of limited resources. Corresponding to the science of bottleneckology is the engineering discipline of *data plumbing*: the matching of technologies in different arenas (data storage, networking, computing) whose characteristics fall within approximately the same realm of performance.

Data pipes are a way of graphically characterizing the rate at which supercomputers can generate data, networks and channels can move data, and storage devices can consume data. Data produced by a model running on a supercomputer can be thought of as flowing out through a pipe whose *cross sectional area* is proportional to the rate at which data is generated. The data is carried through a channel or network, which is represented by a pipe proportional to the technology's maximum (and frequently theoretical or at least very optimistic) point-to-point bandwidth. The data flows into a final pipe which represents the rate at which a storage system (including the controller and actual storage device) can consume and store the data. Bottlenecks, such as protocol overhead and slow network devices, are shown as constrictions in the data pipe.

The NCAR formula of four megabits archived per gigaflop executed can be used to normalize computing power into a common dimension (megabits per second) with channel and network bandwidths and disk transfer rates. These data rates can then be graphed as data pipes and compared.

Figure 1 shows such a comparison, using channel and network technologies such as Ethernet™, FDDI and HIPPI, storage devices such as a disk using a SCSI-2 interface, a disk using an IPI-2 interface, and a RAID disk array, and computers such as an IBM™ RS6000 supermicrocomputer, a single processor of a CRAY Y-MP™ supercomputer, and a hypothetical massively parallel processor such as might be applied to Grand Challenge problems. Also shown are the data pipe cross sections for actual measured throughput between client and server computers using Network File System (NFS™) to pass data over a combination of Ethernet™ and HYPERchannel™ 50.

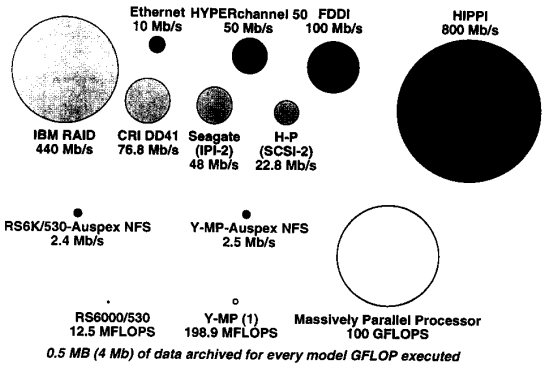


Fig. 1. Data plumbing.

Figure 1 suggests two conclusions. First, while technologies such as HYPERchannel™ 50 and IPI-2 disks may be adequate for current generation supercomputers, the emerging massively parallel processors may generate data at rates faster than these technologies can accommodate. Faster channel and storage technologies such as HIPPI and RAID will be required. Second, the overhead imposed by protocols such as NFS™ places a severe restriction in otherwise adequate network data pipes. This argues for a lower overhead mechanism for moving data between computers and storage systems.

II. Mass Storage: The Next Generation

To meet the impending requirements of Grand Challenge research, NCAR SCD has embarked on a long term project to design and implement a fourth generation mass storage system. The name of the project is MaSSIVE™, the Mass Storage System IV Enterprise.[2] In brief:

- MaSSIVE™ will provide three interfaces to its client computers: high-speed access to complete bitfiles in which the client issues I/O commands over an I/O channel to the storage device controller; lower speed access to complete bitfiles over a network via FTP; and *file-system service* in which clients issue I/O commands over an I/O channel to the storage device controller as client applications make I/O requests. File-system service is what sets MaSSIVE™ apart from existing servers and mass storage systems which provide only record or file service. The MaSSIVE™ file-system interface is the focus of this paper.
- The client interface to a file system provided by MaSSIVE™ will be through a transparent direct high-speed channel connection. The file system will appear to a client to be on a locally attached disk volume.

[2]NCAR has a tradition of naming major systems after *fourteeners*, mountains in Colorado whose peaks are over fourteen thousand feet high. Mount Massive has a height of 14 421 feet (4396 meters) and lies among the Sawatch mountains north of Mount Elbert, the highest peak in Colorado.

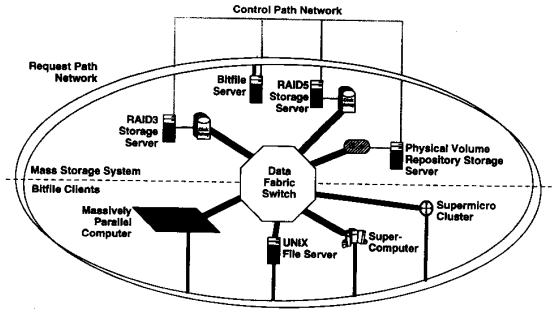


Fig. 2. Architecture overview.

- MaSSIVE™ will not directly provide conventional data access methods such as Network File System. However, a computer supplying such services for its own clients can do so from a file system provided by MaSSIVE™.
- MaSSIVE™ will comply with the IEEE Mass Storage System Reference Model, and will be implemented as a collection of cooperating storage devices and supermicro-class computer systems. Examples of a supermicro-class computer are the IBM™ RS6000 and the Sun Microsystems Sun-4.
- MaSSIVE™ will support a multilevel storage hierarchy which makes use of RAID technology and automated tape library systems.
- MaSSIVE™ will use high bandwidth communications technologies such as FDDI for conventional local area networking based on the Internet Protocol, and a HIPPI-compatible data fabric for high speed data access. (The term "data fabric" is used here in the generic sense of a switching platform interconnecting I/O channels between several computers and storage devices.)
- MaSSIVE™ will require no modifications to its clients' UNIX™ kernels beyond the addition of device drivers. The use of a disk-like device driver as the client interface to the mass storage system is more efficient than using conventional protocols such as NFS™.

Figure 2 shows a high level overview of the MaSSIVE™ architecture, including components of the mass storage system, clients, and the networking infrastructure.

III. Evolution, Not Revolution

MaSSIVE™ is not a revolution in mass storage system design. Rather, it is a rational evolution based on long term experience at NCAR satisfying the need for high-speed, large-scale data storage service to a broad community of demanding supercomputer users. Of particular relevance to MaSSIVE™ are the lessons learned during the development and use of the NCAR third-generation Mass Storage System, the NCAR Local Data Network, and the NCAR Text And Graphics System.

The Mass Storage System

The existing NCAR Mass Storage System [6], [7] is implemented as a central bitfile storage system, where a *bitfile* is a sequential stream of bits, the meaning of which is uninterpreted by the mass storage system. An IBM™ 3090-110J mainframe serves as the Mass Storage Control Processor. The MSCP manages a 100 gigabyte IBM™ 3380 disk farm, a 1–terabyte StorageTek 4400 ACS robotic tape library, and a forty-three terabyte offline tape library containing 111 000 IBM™ 3490 tape cartridges. Now in its third generation, the NCAR MSS was conceived and designed to meet both current and near future storage needs at NCAR. The MSS was patterned after the 1985 version of the IEEE Reference Model for Mass Storage Systems. Major requirements of the system are to provide high speed access to large files (200 megabytes) and to archive data for future use, while handling the day-to-day storage of small files.

The Local Data Network

The MSS provides two different communication mechanisms depending on the speed at which a client can transmit and receive data. The NCAR Local Data Network is the *fastpath* between the MSS and mainframe-class clients that can handle very high data rates. The LDN separates control and data movement onto two different communication paths. The control path connects the client with the MSCP, over which requests to read or write bitfiles are passed. The MSCP establishes a channel connection between the client and the storage device controller over the data path, after which the client issues channel commands directly to the controller. Data moves at channel speeds, and the controller appears to be directly connected to the client. The LDN is implemented with a combination of HYPERchannel™ network adaptors and remote device management modules. The design of the LDN eliminates the intermediate server processor, common to conventional mass storage and network attached storage systems, which is often a bottleneck to high data throughput. Lower performance clients use more conventional network connections in which the MSCP provides the device control. The MSS was designed to be device independent. The only device dependent code is in the client LDN device drivers themselves. This allows the addition of new and different MSS storage devices, while having only to develop device drivers in order to incorporate them into the MSS.

The Text And Graphics System

The NCAR Text And Graphics System is a visualization output production engine which renders an average of 500 000 graphic images per month, using several high speed automated film cameras. It was implemented in C as a group of cooperating processes distributed across a network of UNIX™-based supermicro-class machines. The processes communicate via a message passing scheme which features information hiding and a variable and architecture neutral message format.

The successful MSS, LDN, and TAGS experiences led to the key concepts behind MaSSIVE™: the need for a direct channel connection between the client and the MSS storage device; the use of separate control and data paths; the use of a device driver as the client's interface to the data path; the use of cooperating processes distributed across several supermicrocomputers; and a high degree of software portability and interoperability.

IV. How MaSSIVE™ Differs

MaSSIVE™ will differ from other mass storage and network attached storage systems not in the hardware that it will use, but in the additional functionality it can provide to its client computers.

File Systems and Logical Storage Volumes

Unlike existing record or file servers, MaSSIVE™ can provide *file-system service*. A *file system* is a self-contained, self-defining hierarchical collection of files. For example, a UNIX™ file system consists of a tree structure in which a node that may contain branches or pointers to other files is a directory, and a node which may not is a flat file. Each file system is rooted in a single directory. Branches to other files are of the form of disk block addresses relative to the beginning of the file system. No matter how complex the combination of directories and flat files contained in a file system, no directory contains a branch to a block address outside of the file system. (*Log structured file systems* [8] are organized a bit differently, but MaSSIVE™ will work equally well with them.)

A *logical storage volume* is a directly block addressable extent of data of a fixed size which contains a file system. It is a separate entity from the *physical storage volume*, which is typically a disk drive, but could be some other storage medium. A logical volume could be stored on a single physical volume, it could be one of several logical volumes on a single physical volume, or it could be spread across multiple physical volumes. The structure of a file system stored on a logical storage volume, and the organization of the files within it, is defined to the native operating system solely by information contained in the file system itself.

UNIX™ creates a single system-wide file hierarchy by binding, or *mounting*, the root directory of each file system to an empty directory in the existing hierarchy. As users traverse the file hierarchy, they transparently cross logical storage volume boundaries as they move from file system to file system.

A device driver is an operating system's lowest level software interface to a physical hardware device. Access to a file system is, at the device driver level, reduced to reading and writing blocks on a logical storage volume. At this level, a logical storage volume can itself be thought of as a bitfile whose contents are directly accessible. The interpretation of the contents as a file system takes place at a higher level in the operating system. It is this abstraction of file systems to logical storage volumes to bitfiles that is the basis for MaSSIVE™.

MaSSIVE™ will provide seemingly local logical storage volumes containing complete native file systems to its UNIX™-based clients when the client mounts the file system. This is in contrast to record servers like Network File System (NFS™) [5] that provide data to the client in piecemeal fashion as the client's application performs I/O activities, or file servers such as the Andrew File System (AFS™) [9] which transfer complete files to the client.

Current mass storage systems may be file or record servers, or they may provide manual access to files upon user request. An example of such a mass storage system is UniTree™, originally developed at Lawrence Livermore National Laboratory [10], [11]. UniTree™ provides record access via NFS™ and nontransparent file access via the Internet File Transport Protocol (FTP) [12].

File-system service offers significant capabilities beyond current mass storage systems. Since conventional mass storage systems provide files and not file systems, users have had little alternative but to organize their model output data into large monolithic flat files. This has produced a cumbersome data organization problem, resulting in extremely inefficient access to desired subsets of the data. A mass storage system which provided an entire file system would allow an application program (such as a climate model) running on a client to organize its output in, for example, a hierarchical file and directory structure, yet permit the mass storage system to treat the entire hierarchical structure as a single entity for the purposes of management and archival. This also alleviates some of the problems mass storage systems have suffered in the past in managing a name space containing millions of bitfiles, by managing instead a smaller name space in which each individual bitfile (from the MSS point of view) is a file system (from the user point of view).

Channels and Networks

When a UNIX™ application reads data from a local disk, the I/O request passes through the file system code in the operating system where it is translated into a read for a specific data block on a logical storage volume. From there it is passed to a *disk device driver*, the operating system's lowest-level software interface to the physical storage device. The device driver translates the read request into a physical I/O command appropriate to the device. The command is passed to the storage device *controller* through a *channel*, a special purpose data path that links a processor with a controller. The controller handles the specifics of actually dealing with the physical storage device.

When a UNIX™ application reads data from a file residing on a remote Network File System server, the path that the request takes is quite different. NFS™ is a remote storage service implemented as a layer on top of the Internet Protocol stack. It is typically used over local area networks based on physical transport mechanisms such as Ethernet™ or FDDI. The I/O request still passes through a portion of the file system code, then it takes a detour into the protocol stack code. It is then processed through several layers of software before it reaches a network device

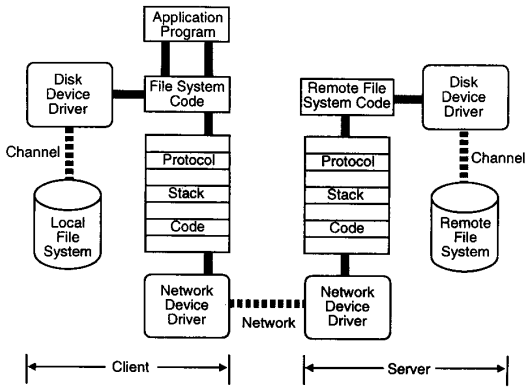


Fig 3. Channels versus networks.

driver. In the driver, the network datagram containing the request is further encapsulated into a packet suitable for the physical transport mechanism. The packet travels over the network to the server, where it is received by that machine's *network device driver*. The decapsulated datagram travels back through the protocol stack in the server, where the original I/O request is finally extracted. Then, as before, this request is passed through the disk device driver, down a channel, and to the physical storage device controller for actual execution. The resulting data block must make the same tortuous return trip. This is illustrated in Fig. 3.

Because protocol stacks and transport mechanisms were designed to be general purpose, and because networks in general have more points of failure than channels (due mainly to their wider geographic dispersion and less restricted physical access), they necessarily provide capabilities—and impose overhead—beyond what is required for any one specific use. The expense of this overhead may not be great on relatively low performance computers such as desktop workstations, but it is very costly when it results in a supercomputer going idle because the application is waiting for I/O to complete. Worse, many protocols and transport mechanisms enforce a relatively small maximum datagram or packet size (eight kilobytes and 1500 bytes are typical), which prevents the overhead from being reduced by increasing the amount of data transferred with a single request. Remarkably good performance has been demonstrated by Cray Research using the TCP/IP protocol suite over a HIPPI channel [13]. However, this was accomplished using optimized TCP/IP software and large (64 KB) packets, between two dedicated CRAY Y-MP™ supercomputers. Getting a similar level of performance in the context of FTP or NFS™ is still problematic.

This is why MaSSIVE™ makes a careful distinction between a network and a channel. Data that moves through a channel is handled at a much lower level, and hence with lower overhead, than with a network which uses heavyweight protocols.

MaSSIVE™ clients access file systems through what appears to be a disk device driver. The I/O requests are passed down a channel to the storage device controller. The fact that I/O requests and data move through a high-performance data fabric, which is handling the switching of channel connections between several computers and controllers simultaneously, is transparent to the client operating system.

Architecture Neutrality

The physical location of data on a record server is transparent to the client's application. However, the file system format for the server and the client must be the same or must be translated in real time. Similarly, the location of data on a file server is also transparent to the client. The proposed MaSSIVE™ system takes this capability one step further by storing entire file systems as uninterpreted streams of bits, or *bitfiles*. The interpretation of the contents of the file system (e.g., the directory structure and file naming conventions), is completely up to the client operating system. MaSSIVE™ can therefore provide file system service for a variety of file system architectures. The downside is that clients whose file-system architectures differ cannot serially access the same file system from MaSSIVE™.

No Processor Bottleneck

A record server processor must respond to each remote I/O request from a client. A file server must respond each time a client opens a file that is not already cached on the client. In both cases, the server is a potential bottleneck between the supercomputer client and its data. MaSSIVE™ eliminates the processor bottleneck by providing a direct channel connection between the client and its data.

Third Party Storage Devices

While MaSSIVE™ is not intended to completely replace local disks on clients, it can eliminate the need for large local disk capacity by providing direct access to more cost-effective disk architectures available from a potentially large number of vendors.

Conventional Storage Services

The MaSSIVE™ file-system interface does not directly address providing file or record access or other related services such as database access, nor does it address the issues of file or file system sharing and the concomitant problems of translation of different file system architectures and data formats and concurrent access control. However, a MaSSIVE™ client can supply such services using its own file system provided by MaSSIVE™, through the file, record and database access, file-sharing, and file locking mechanisms native to its own operating system.

V. How MaSSIVE™ Will Work

Reference Model

MaSSIVE™ will comply with Version 4 of the IEEE Mass Storage System Reference Model [14], and will

follow the Version 5 standard as it emerges from the IEEE Storage Systems Standards Working Group. The unit of storage defined by the IEEE Model is the *bitfile*, an uninterpreted string of bits. The bitfile is independent of structure imposed by any particular operating system. MaSSIVE™ clients (referred to as *bitfile clients* [14] in the Model) will provide their own interpretation of a bitfile as a native file system.

Storage Resources

Storages resources can be divided into three classes: *online*, *near-online*, and *offline*. Online storage consists of direct access storage devices such as disks. Near-online storage includes robotic tape libraries. Offline storage consists of shelf-stored tapes, which must be mounted manually, and their associated tape drives. The quality of service offered by each storage class defines a multilevel storage hierarchy. Client computers of the current NCAR Mass Storage System can access bitfiles from each storage class; for example, a client computer can read a bitfile directly from a tape in a robotic tape library. Other mass storage systems copy, or *stage*, a bitfile from offline or near-online storage to online storage before it can be accessed by a client. Bitfiles which are infrequently used are copied, or *migrated*, from online storage to slower, less expensive, offline or near-online storage. For this reason, near-online and offline storage are sometimes referred to as *archival* storage. MaSSIVE™ will require a bitfile to be staged online before a client can access it as a file system.

MaSSIVE™ associates a supermicrocomputer with each storage device. This supermicro is called a *storage server* [14] in the Model. It provides intelligent services to its associated storage device. Storage servers respond to requests for access to their storage devices, they talk among themselves to coordinate the migration of bitfiles between one another, and they allocate and manage space on their storage devices.

MaSSIVE™ will incorporate four storage resources:

- a Level Three RAID [2] configured for very high speed transfers of large data blocks (complete bitfile transfers);
- a Level Five RAID [2] configured to transfer many data block requests of small to medium size (file system access);
- a robotic tape library, called the *physical volume repository* [14] in the Model, with the capability to store terabytes of archived data (including entire file systems stored as bitfiles); and
- an offline tape library, referred to as a *deep archive*, used to store very infrequently used data in the most cost effective manner.

Communication Resources

MaSSIVE™ will be implemented using three different communication paths.

- A *request path* will connect clients to the *bitfile server* [14]. The bitfile server is the clients' interface to the rest of the mass storage system. The request path will be implemented as a FDDI ring.
- A *control path* will connect the bitfile server and the storage servers to one another, and will not be accessible to clients. The control path will be implemented as an Ethernet™ local area network. Ethernet™ will be used because several commercial storage subsystems already provide Ethernet™ control interfaces, and high bandwidth and large packets are not required.
- A high speed *data fabric* will provide direct channel connection between the physical storage devices and the clients. The data fabric will be implemented using a high speed switch featuring HIPPI-compatible channels. Eventually, the data fabric will also make use of the emerging gigabit per second Fibre Channel Standard.

Client Interfaces

Three client interfaces are planned for MaSSIVE™.

- Clients on the data fabric may copy complete bitfiles to and from the mass storage system over the data fabric. These clients issue I/O commands over the data fabric directly to the storage device controller until the entire file is transferred. This function is provided by the current NCAR mass storage system.
- Clients which are not on the data fabric, but which have a TCP/IP network path to the mass storage system (e.g., workstations at remote locations), may copy complete bitfiles to and from the mass storage system via FTP. This function is also provided by the current NCAR MSS.
- Clients which are on the data fabric may access bitfiles as native file systems. These clients issue I/O commands over the data fabric directly to the storage device controller as long as the file system is in use by the client.

The file-system interface provides additional functionality over the current NCAR MSS. This interface will be implemented through a UNIX™ device driver. This driver serves as the *bitfile mover* [14] for the client.

Before a MaSSIVE™ file system can be used by a client, the client must first bind it to the existing directory hierarchy in a manner similar to mounting a local file system. A user command makes a request to the MaSSIVE™ *mount daemon* which runs on each client system. The daemon serves as the interface between the client and the MaSSIVE™ bitfile server, communicating over the request path. The request to the bitfile server includes the name of the bitfile in the MSS that contains the file system, and the name of the empty directory in the client's directory hierarchy on which the new file system is to be mounted. If the file system is on archive storage, the bitfile server will negotiate with the appropriate storage servers over the control path to stage the file system online. Once the file system is online, a connection is established over the data fabric between the MaSSIVE™ device driver in the client and the storage device controller. Finally, the daemon issues

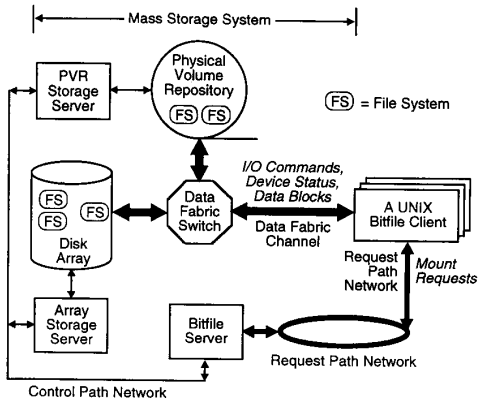


Fig. 4. Bitfile client and servers.

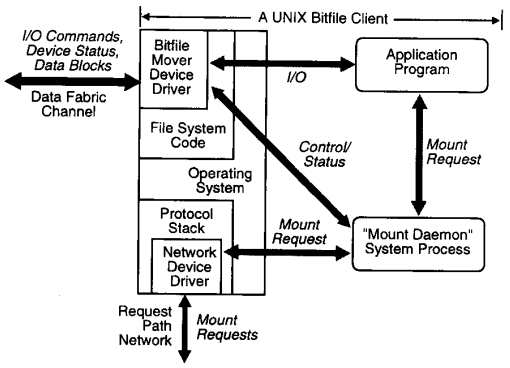


Fig. 5. Bitfile client detail.

a mount system call to the client's operating system to access the bitfile and complete the mounting procedure. This is illustrated in Fig. 4. The detail of the bitfile client is shown in Fig. 5.

The MaSSIVE™ UNIX™ device driver is divided into three separate functional layers. The outermost layer is the interface to the UNIX™ kernel. It implements a standard disk driver interface with entry points for the usual I/O functions such as *open, read, write, ioctl* (I/O control), and *close*. The intermediate layer contains a *personality module* specific to the target storage device. This layer understands how to translate a higher level driver request into a particular I/O command set, such as FIPS-60, IPI-3, IBM™ 3393, etc. The innermost layer is the interface to the high speed data fabric. It encapsulates the I/O commands in a suitable manner for transmission to the storage device controller. This is illustrated in Fig. 6.

A problem in current mass storage systems which MaSSIVE™ does *not* address is that of store-and-forwarding data. Typically, a device driver reads data into a buffer inside the operating system. Then the data is copied into the buffer inside the application program. Additional buffering may even occur inside the device driver itself. Data being written passes through a similar chain of store-and-forwarding. This data movement between buffers

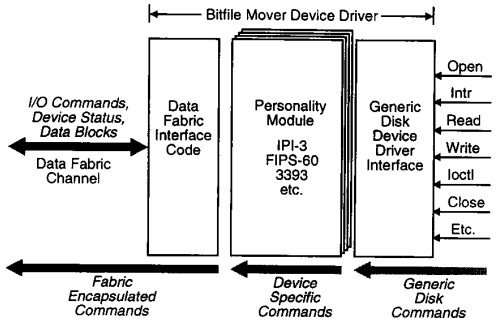


Fig. 6. Bitfile mover device driver.

imposes significant overhead. If the client operating system provides a mechanism to directly access the application program buffer, bypassing the intermediate buffering, the MaSSIVE™ device driver should make use of it.

Utilities will be provided for MaSSIVE™ to create empty bitfiles of a fixed size to be used as client logical storage volumes. System utilities native to the client system will be used to initialize a MaSSIVE™ volume with a file-system structure, and to verify and repair the integrity of the structure. On a typical UNIX™ system, these utilities would be *newfs* (new file system) and *fsck* (file system check). The native utilities will work in exactly the same manner as they would on a local disk.

Security and Integrity

The issues of security and data integrity fall into two categories: providing for user authentication for requests arriving over the request path, and ensuring data integrity when clients access shared storage devices over the data path. Since MaSSIVE™ provides physical device sharing, but not file, file system, or logical device sharing, it avoids many of the security pitfalls associated with distributed file systems and traditional mass storage systems. A discussion of distributed security in mass storage systems can be found in [15].

Basic physical security is provided by segregating MaSSIVE™ clients onto a request path that is a separate network from the control path, and by the connectivity requirements implied by the data fabric. User security at the client end is provided by mechanisms native to the client's operating system. The control path contains only trusted hosts.

For performing user authentication of requests on the request path, we envision using available authentication mechanisms, such as those provided by the Kerberos user authentication software from MIT's Project Athena [16].

Ensuring data integrity ultimately requires hardware support. MaSSIVE™ will allow a client to access a shared high performance disk array as if that device were directly channel attached. At the lowest level of disk access, access validation to the device is provided via the storage server. To provide the optimal connection and prevent bottlenecks, data should move between the client and the

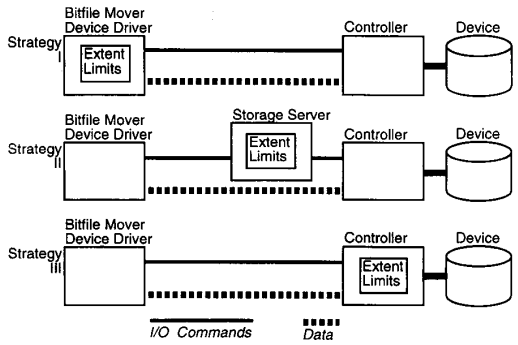


Fig. 7. Extent checking strategies.

storage device through the data fabric, without data flowing through the validating storage server. To do this, the bitfile mover/device controller must have information passed to it by the storage server which specifies the data extent that the client is permitted access on the storage device. Access validation can be divided into three strategies. Each strategy can be characterized by three factors: the efficiency with which it performs I/O, the degree to which it secures the integrity of clients' file systems on a shared storage device, and the ease with which it can be implemented. This is illustrated in Fig. 7.

Strategy I: A client sends I/O commands directly to the storage device's controller. The client is allowed direct access to the complete contents of the storage device and is limited only by restrictions in the client software. This strategy is currently used in the NCAR Mass Storage System. It is efficient, insecure (since the client may not be a trusted host), and easy to implement.

Strategy II: A supermicrocomputer used as a storage server is placed between the client and the controller. Each I/O command from the client is first received by the storage server for validation before it is forwarded to the controller. Data still moves directly between the client and the controller. This strategy is the one that will be used during the initial phase of MaSSIVE^{TM}. It is inefficient (since I/O requests must pass through an intermediate processor), secure (since the intermediate processor is trusted), and easy to implement provided the controller accepts third-party control (that is, I/O commands can be received from a source different from where the data is received or sent).

Strategy III: The function of the storage server in Strategy II is integrated into the controller. The controller itself validates the I/O commands which it receives from a client computer. This strategy is to be used in the final version of MaSSIVE^{TM}. It is efficient (since extent checking is handled at the controller level), secure (since the controller is trusted), and difficult to implement (since it requires co-operation by the controller manufacturer).

Concurrency Control

It is not part of the proposed MaSSIVE^{TM} design to allow concurrent read/write access to a file system to multiple clients. It is also not part of the proposed design that MaSSIVE^{TM} support any file system or data format translation. Concurrency control within a single client, including file locking protocols and other synchronization mechanisms, will be provided by the client's native operating system.

Open Questions

There are many open questions and concerns regarding specific design and implementation issues in MaSSIVE^{TM}. A few of them are mentioned here.

- How will the switching overhead for the data fabric impact UNIX^{TM} block-level I/O performance?
- How will the global view of bitfile directories be distributed and managed among the storage servers?
- What is the most suitable strategy for the name server to manage the bitfile name space?
- What are the kernel limitations and performance bottlenecks inherent in various UNIX^{TM} platforms which may affect the MaSSIVE^{TM} client device driver?
- What is the ability of manufacturers of suitable storage devices to support HIPPI-compatible channels, and to allow third party control of them?
- Will the emerging Fibree Channel (draft) Standard [17] be ratified and will commercial products based on FCS become available soon enough to be used instead of HIPPI in the early phases of MaSSIVE^{TM}?
- Will RAID-class storage devices permit scatter/gather I/O operations coupled with very large data transfers, so that the bandwidth of the I/O channel can be effectively utilized?
- What is the most suitable strategy for the management of migration of file systems (versus files) in the multilevel storage hierarchy?

VI. Where It Stands Today

The concepts behind the Mass Storage System IV Enterprise were initially developed as part of the planning effort by the Mass Storage Group at NCAR, headed by David Kitts. They were later expanded in cooperation with Cray Research, Inc., IBM^{TM} Corporation, Network Systems Corporation, and Storage Technology Corporation, in response to the DARPA Broad Area Announcement 91–17, "Research in Advanced Software Technology and Algorithms."

The implementation of MaSSIVE^{TM} is divided into several steps.

The MVS-based production MSS has recently been given TCP/IP capabilities and FDDI connectivity, and an FTP interface to the MSS has been implemented. The existing HYPERchannel^{TM} Local Data Network is being augmented by a HIPPI-based data fabric built around a Network Systems Corporation PS32 crossbar switch. Part of the MSS import/export function, which moves data between external media and the mass storage system, is being migrated from the IBM^{TM} 3090 MSCP to a UNIX^{TM}-based IBM^{TM}

RS6000 RISC system. This effort is in progress, and will result in a UNIX™-based bitfile mover.

The next step is to develop a UNIX™-based storage server, and use it to integrate a disk array into the production MSS. The resulting hybrid system will leverage off the existing MVS-based bitfile server.

Saving the most difficult tasks to last, the final steps will include building UNIX™-based replacements for the remaining MVS portions of the MSS. A second disk array will be integrated into the system. The HYPERchannel™ network will be completely phased out. Finally, the archive storage media will begin the migration from IBM™ 3490-compatible cartridge tapes to media with larger capacity. One possible choice is helical scan tape in D-3 format, which holds fifteen to thirty gigabytes per tape, compared to 400 megabytes per 3490 tape, and has a form factor identical to that of the 3490 cartridge. This could reduce NCAR's tape inventory from more than 111 000 3490 tapes to fewer than 1500 D-3 tapes. It takes a long time to read all the data from 111 000 tapes; this migration will span several years. The duration of the migration process, as well as the need for data longevity, places severe requirements on the choice of media. The reliability and durability of tapes and drives using helical scan technology is a major concern.

ACKNOWLEDGMENT

The National Center for Atmospheric Research is operated by the University Corporation for Atmospheric Research under the sponsorship of the National Science Foundation. Any opinions, findings, conclusions, or recommendations expressed in this paper are those of the authors and do not necessarily represent the views of the National Science Foundation.

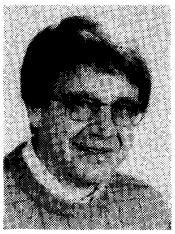
The authors are grateful for the contributions made to the design of MaSSIVE™ by SCD staff members Joe Choy, Gene Harano, John Merrill, and Craig Ruff.

REFERENCES

[1] Committee on Physical, Mathematical, and Engineering Sciences, *Grand Challenges 1993: High Performance Computing and Communications*, National Science Foundation, Computer and Information Science and Engineering Directorate, 1993.
[2] D. Patterson *et al.*, "A case for redundant arrays of inexpensive disks (RAID)," in *Proc. 1988 ACM-SIGMOD Conf. Management of Data*, Chicago IL, 1988.
[3] F. Ross, "An overview of FDDI: The fiber distributed data interface," *IEEE J. Selected Areas Commun.*, vol. 7, iss. 7, Sept. 1989.
[4] B. Morris *et al.*, ed., *High-Performance Parallel Interface Mechanical, Electrical, and Signalling Protocol Specification*, American National Standards Institute, X3.183–1991
[5] Sun Microsystems, Inc., NFS™: Network File System Protocol Specification, RFC1094, 1989
[6] M. Nelson *et al.*, "The NCAR Mass Storage System," in *Proc. Eighth IEEE Symp. Mass Storage Systems*, 1987.
[7] B. Collins *et al.*, "Profiles in mass storage: A tale of two systems," in *Proc. Ninth IEEE Symp. Mass Storage Systems*, 1991.
[8] J. Ousterhout *et al.*, "Beating the I/O bottleneck: A case for log-structured file systems," UCB/CSD 88/467, Computer Science Division (EECS), University of California, Berkeley, Berkeley, CA, 1988
[9] J. Howard, "An overview of the Andrew File System," in *USENIX Winter Conference Proc.*, 1988.
[10] Distributed Computing Solutions, The UniTree™ Solution for Mass Storage, San Diego, CA.
[11] C. Hogan *et al.*, "The Livermore Distributed Storage System: Requirements and overview," in *Proc. Tenth IEEE Symp. Mass Storage Systems*, 1990.
[12] J. Postel *et al.*, File Transfer Protocol, RFC959, 1985.
[13] D. Borman, "TCP/IP performance at Cray Research," in *Proc. Twenty-Third Internet Engineering Task Force, Corporation for National Research Initiatives*, 1992.
[14] S. Coleman *et al.*, eds., Mass Storage Systems Reference Model, Version 4, Institute for Electrical and Electronics Engineers, May 1990.
[15] A. Hanushevsky, Distributed Security in the IEEE Storage Reference Model, IEEE P1244 Name Server Subcommittee, Aug. 1992.
[16] J. Steiner *et al.*, "Kerberos: An authentication service for open network systems," in *Usenix Conf. Proc.*, Dallas, TX, Feb. 1988.
[17] Fibre Channel—Physical Layer, American National Standards Institute, X3T9.3 91/034.

John Sloan received the B.S. and M.S. degrees in computer science from Wright State University in Dayton, Ohio.

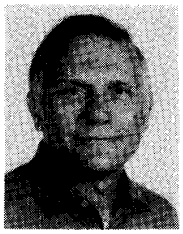
He is a systems programmer in the High Performance Systems and Networking Section of the Scientific Computing Division at the National Center for Atmospheric Research. His interests include concurrent programming, performance measurement, and scalability.



Bernard O'Lear received the B.S. degree in mathematics from Regis College Denver, CO.

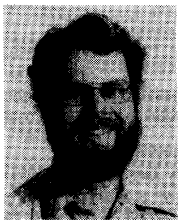
Currently he is Manager of the High-Performance Systems and Networking Section of the Scientific Computing Division at the National Center for Atmospheric Research. Since joining NCAR in 1964, he has been involved in a number of scientific and systems software projects. He has participated in the procurement and installation of several large-scale computer and data storage systems.

During 1989 and 1990, he was a member of the Congressional Office of Technology Assessment Advisory Panel on Information, Technology, and Research. He is currently a member of the Advisory Panel for the National Center for Supercomputing Applications. He was a member of the Technical Review Committee of the National Science Foundation Task Force on Supercomputing during 1984. In 1988, he received the IEEE Computer Society's Certification of Appreciation and the IEEE Computer Society's Meritorious Service Award for his work on the Mass Storage Systems Technical Committee.



David Kitts received the B.A. degree in mathematics from the University of California at Los Angeles in 1958.

Currently, he is head of the NCAR SCD Mass Storage Group. Since joining NCAR in 1961, he has been involved in the development of operating systems for several high performance computers and was the group head of the NCAR FORTRAN compiler project. His involvement with the Mass Storage Group began with the current third generation project where he was responsible for the development of a direct connect channel attachment to archive storage devices from a high performance computer. Additionally, he was co-developer of the overall project. He is currently responsible for developing the architecture for the future mass storage project at NCAR.



Basil Irwin has 13 years of experience in the NCAR Scientific Computing Division involving high speed networking design, implementation and operation. He is the chief designer and implementor of NCAR's HYPERchannel™-based MASnet file transfer network, which connects over a dozen computers including two CRAY Y-MP™ supercomputers, an IBM™ MVS 3090 system, Sun systems, and IBM™ RISC 6000 systems.