

The MaSSIVE™ Project At NCAR

J. L. Sloan, B. T. O'Lear, D. L. Kitts, E. E. Harano
National Center for Atmospheric Research
Boulder, Colorado

Abstract

The Mass Storage System IV Enterprise is a fourth-generation mass-storage system being designed at the National Center for Atmospheric Research. 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 on-line storage devices and then provide its clients with raw block access to the staged file system. It will consist of cooperating processes distributed across a network of super-microcomputers. The client interface to a file system will be implemented through a device driver that will permit access to MaSSIVE on-line 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, the file systems may be migrated back to archive storage and removed from on-line storage.

Introduction

The Scientific Computing Division (SCD) at the National Center for Atmospheric Research (NCAR) in Boulder, Colorado, is in the design phase for the next-generation mass-storage system (MSS) that will eventually replace the existing mainframe-based NCAR MSS [1]. The name of the project is MaSSIVE[†], the Mass Storage System IV Enterprise.

Three different client interfaces are planned for MaSSIVE:

- High-speed access to complete bitfiles, in which the client issues I/O commands over a channel directly to the storage-device controller;
- Lower-speed FTP access to complete bitfiles over a network; and
- *File-system service*, in which clients issue I/O commands over a channel to the storage device controller as client applications make I/O requests.

The first two interfaces exist in the current NCAR MSS. This paper describes the third interface.

[†] 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.

Overview

The planned hardware architecture of MaSSIVE differs little from other distributed mass-storage systems. The software difference is that it will provide *file systems* to its clients, not just flat files. To the client, the file system will appear to be on a disk attached through a direct high-speed channel. Although MaSSIVE will not directly provide conventional data-access methods such as NFS, visualization, or database services, such facilities could easily be built on top of it. MaSSIVE will be implemented as a distributed collection of cooperating storage devices and super-microcomputer-class systems, and will support a multi-level storage hierarchy. A design goal is to encapsulate the bitfile mover portion of MaSSIVE in what appears to the client to be a device driver for a local disk.

Figure 1 shows an overview of MaSSIVE, including servers and bitfile clients.

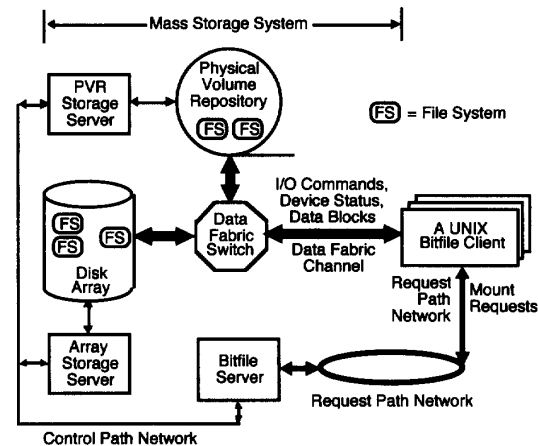


Figure 1. Servers and Bitfile Clients

File Systems and Logical Storage Volumes

Unlike existing record or file servers, MaSSIVE provides *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* [2] 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. File-system service offers significant capabilities beyond current mass-storage systems. 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, or even use a commercial database package, 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

MaSSIVE makes a careful distinction between a *network* and a *channel*. A network is a general purpose data communications path which connects computers to one another. Data that moves through a network is processed through many layers of software. A channel is a special purpose communications path which connects a computer to a storage device. Data that moves through a channel is handled at a much lower level, and hence with lower overhead, than with a network which uses heavy-weight protocols.

Because protocol stacks and transport mechanisms such as TCP/IP and Ethernet were designed to be general purpose, 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 desk top workstations, but it is very costly when it results in a super-computer 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 [3]. However, this was accomplished using optimized TCP/IP software and large (64KB) packets, between two dedicated CRAY Y-MP supercomputers. Getting a similar level of performance in the context of FTP or NFS is still problematic.

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. 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 storage devices 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 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 provide file-system service for a variety of file-system architectures.

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. This is the same strategy that has proved successful in the NCAR Local Data Network (LDN), an NSC HYPERchannel-based back-end network which services the third-generation NCAR MSS.

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. However, not all storage devices will offer the functional characteristics needed by MaSSIVE, such as scatter/gather and large block transfers. This is discussed in more detail below.

Conventional Storage Services

The MaSSIVE file-system interface does not directly address providing conventional network attached storage, 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. Storage services built on top of MaSSIVE might include NFS, AFS (a.k.a. DFS), or FTP.

Storage Resources

Storage resources can be very broadly divided into three classes: on-line, near-on-line, and off-line, the latter two being examples of archival storage. Examples of on-line storage devices include RAID disk arrays, robotic tape libraries are near-on-line, and shelf-stored tapes are off-line. The MaSSIVE file-system interface will require a bitfile to be staged on-line from near-on-line or off-line before a client can access the bitfile as a file system. This is in contrast to the existing NCAR MSS flat file interfaces, which MaSSIVE will also incorporate, which permit a client to access the MSS-managed storage directly.

Communication Resources

MaSSIVE will use three different communication paths: a *request path*, a *control path*, and a *data fabric*.

The request path will connect clients to the bitfile server. Current plans call for the request path to be implemented as an FDDI ring.

The 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 communications bandwidth and packet volume are not necessary.

The 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 Interface

The file-system client interface to MaSSIVE will be through a UNIX device driver. This driver serves as the *bitfile mover* for the client. From the user's point of view, MaSSIVE file systems will be mounted similarly and accessed identically to a file system on a local disk.

A request to mount a file system from a bitfile will be made through a MaSSIVE *mount daemon* running on the client. Communication between the mount daemon and the MaSSIVE device driver will be through a combination of outstanding read requests and *ioctl* calls. The daemon serves as the interface between the client and the MaSSIVE bitfile server, communicating over the request path. The bitfile server in turn handshakes with the appropriate storage servers over the control path. The storage servers will cooperate to stage, if necessary, the requested bitfile from archival storage to on-line storage. Figure 2 shows a detail of the bitfile client.

How a bitfile that is used as a file system is uniquely identified is an open question. Traditional UNIX file systems can be identified by the UNIX device name for the storage device on which they reside, and by the absolute path name for the point in the file-system hierarchy upon which they are mounted. Neither is sufficient to uniquely identify a MaSSIVE file-system bitfile which, at different times, may be mounted on different clients and on different mount points, and which may be accessed through different UNIX device names (and which may be assigned different UNIX major and minor device numbers). A special MaSSIVE *mount* command, which includes as its arguments an MSS bitfile name, may be necessary.

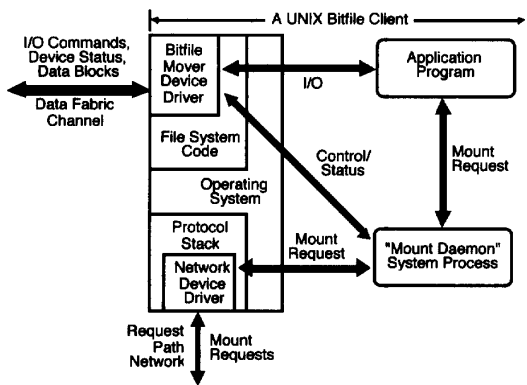


Figure 2. Bitfile Client Detail

The MaSSIVE UNIX device driver will be divided into three separate functional layers. The outermost layer is the interface to the UNIX kernel. It will implement 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 will contain a *personality module* specific to the target storage device. This layer will understand 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 will be the interface to the high speed data fabric. It encapsulates the I/O commands in a suitable manner for transmission to the storage device.

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.

Inter-Process Communication

The current NCAR MSS uses an Inter-Process Communication Subsystem (IPCS) based on message-passing through shared memory to tie together tasks (processes) in a single MVS job or in separate jobs [4]. The TAGS UNIX-based distributed system developed at NCAR uses message passing over Berkeley sockets [5]. The leading contender for MaSSIVE IPC is the Remote Procedure Call (RPC) mechanism that is part of the Open Software Foundation (OSF) Distributed Computing Environment (DCE) [6].

The modular design of the current NCAR MSS allows it to be incrementally ported process-by-process from MVS to UNIX. Part of the porting process will be to adopt a common IPC mechanism in both the old and new

portions of the MSS. MVS supports Sun Microsystems RPC protocol; if DCE is not available for MVS in time for the initial portions of the project, an interim IPC layer will be implemented to map from Sun RPC on MVS to DCE RPC on other platforms.

Security and Data Integrity

The issues of security and data integrity fall into two categories:

- Providing for user authentication for requests arriving over the request path, and
- Insuring 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 [7].

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 OSF DCE Kerberos user authentication software, originally from MIT's Project Athena.

Insuring 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 storage device through the data fabric, without data or control 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.

Strategy I: A client is allowed direct access to the complete contents of the device and is limited only by restrictions in the client software. This strategy is currently used in the third-generation NCAR Mass Storage System. It is efficient, insecure (since the client may not be a trusted host), and easy to implement.

Strategy II: A super-microcomputer is assigned the task of being a storage server that allocates and controls the extent of access for a client through third-party control of the storage device controller. This server obtains the extent information at the time the MaSSIVE file system is mounted. Data is moved directly between the client and the controller, but it is necessary that the client I/O request first be received by the storage server for processing before it then sends the modified command to the on-line storage device controller. Clients are identified based upon the source hardware address in the I/O request packet. This strategy is the one that will be developed during the initial phase of MaSSIVE. It is inefficient (since I/O requests must pass through an intermediate processor), secure (since the intermediate processor is trusted), and easy to implement.

Strategy III: A super-microcomputer is assigned the task of being a storage server that controls the extent of access a client has to a storage device controller. This extent is passed to the controller along with a packet of information that defines the access path the client will be using. This information is returned to the client, who is then allowed to access the device with no intervention from the storage server. Like Strategy II, client identification is based upon the hardware source address in the I/O request packet. This is the preferred method of communication for the device controller. It will require close coordination with a device vendor. This strategy is the final development of the paradigm to be used in the final version of MaSSIVE. It is efficient (since extent checking is handled at the controller level), secure (since the storage server and its associated controller are trusted), and difficult to implement (since it requires cooperation by the controller manufacturer).

Concurrency Control

It is not part of the proposed MaSSIVE 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 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

Since MaSSIVE is in the very early design stage, many issues have not yet been tied down. Some of the more vexing concerns are discussed below.

Platform Constraints

UNIX is a popular operating system on which to build large software projects. At least two such efforts, Andrew at Carnegie-Mellon University [8, 9] and Athena at MIT [10], have generated articles in the literature discussing pitfalls of using UNIX as a development and production platform. A common thread running through such articles is the issue of *scalability*. Kernel resources such as process or file tables are too easily exhausted, and frequently there are severe performance consequences to expanding those resources. Typically, such problems in scalability are only discovered after a substantial investment of time and money has already been made in the development project, and their resolution can have a serious retroactive impact on the design.

For example, a benchmark program developed at NCAR has shown that on many UNIX systems, the per-process context switch time is not constant as the number of processes increase, lending little credibility to context switch times published by the vendor [11]. The increase in context switch time can be dramatic and even non-linear. Efforts are underway to identify hidden pitfalls like this while evaluating possible development and production platforms.

Input/Output Strategy

Several client computers on the MaSSIVE data fabric will share the same storage resource. Contention for that resource will force one client to wait at least until the other client completes its current I/O request. The waiting time is determined not only by the I/O bandwidth and latency of the client computer and the storage device, but also by the bandwidth, latency, and connection rate of the data fabric. Furthermore, the difference in I/O behavior on UNIX-based supercomputers between interactive users versus large application programs suggests that optimization to a single I/O strategy may be difficult.

A trace analysis of file-system usage on *non-supercomputer* UNIX systems [12] has shown that most files are both short (under four kilobytes) and short-lived (under three minutes), although large files account for a significant fraction of the data transferred; also, accesses tend to be highly sequential, and file-system activity occurs in bursts. We assume that this behavior describes file system use by interactive users, even on supercomputers.

A similar analysis of the I/O behavior of supercomputer applications [13] has shown that small files contribute in only a minor way to the total supercomputer I/O load, which is dominated by transfers of large (multi-megabyte) machine-generated files. I/O is "bursty" but cyclic. A simulation of application I/O activity also shows the value of caching data blocks for read-ahead and write-behind (asynchronous writes).

The bandwidth of the data fabric can be more effectively utilized by transferring larger blocks of data per transfer. However, there may be little locality of reference when servicing many independent I/O requests from different processes on a single client computer, particularly for interactive users. Storage device controllers that implement a scatter/gather function, so that data from independent disk blocks destined to a single host can be transferred in a single bundle, may be a performance win. Read-ahead and write-behind caches on the client end will also contribute to efficient large-block transfers, with the usual reliability penalty for asynchronous writes.

The high speed switch upon which the MaSSIVE data fabric will be built offers a data latency and connection rate measured in tens to hundreds of microseconds, and is claimed to transfer eight-kilobyte data blocks faster than 10,000 blocks per second. It remains to be seen if this rate will be fast enough to service the aggregate disk I/O needs of several gigaflop supercomputers.

Data Fabric Technology

There is some concern over the choice of HIPPI as the prototype transport mechanism. There is a strong desire to use technologies endorsed as standards by organizations like IEEE and ANSI. Also, HIPPI products are available from vendors now. However, during the life span of the MaSSIVE software development project, the Fibre Channel Standard (FCS) will probably be formally endorsed, and products meeting this standard will become available. The fiber optic-based FCS is likely to become the standard for high-performance I/O channels.

Fortunately, the data fabric switch can be configured to use multiple transport mechanisms simultaneously. The production configuration for MaSSIVE will probably include a mixture of HIPPI, FCS, and other technologies.

Current Status

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 NCAR's vendor partners in MaSSIVE: Cray Research, Inc., IBM Corporation, Network Systems Corporation, and Storage Technology Corporation.

The implementation of MaSSIVE is divided into several incremental 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 Local Data Network is being augmented by a HIPPI-based data fabric built around a Network Systems Corporation PS32 crossbar switch. At the same time, a bitfile mover using HIPPI over the PS32 is being implemented on a UNIX-based platform to serve as a client prototype.

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 step will include building UNIX-based replacements for the remaining MVS portions of the MSS. The bitfile mover will be extended, and a second disk array will be integrated into the system. The HYPERchannel network will be completely phased out. At the same time, the Storage Technology ACS 4400 robotic tape silo will be upgraded to use a higher-performance robot and higher-capacity magnetic media.

Since MaSSIVE is still in the early stages of design and implementation, and several implementation strategies and technologies are being evaluated and prototyped, comments and criticisms are encouraged. The authors can be contacted via electronic mail at *massive@ncar.ucar.edu*.

Acknowledgments

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 MaSSIVE by SCD staff members Joe Choy, Basil Irwin, John Merrill, and Craig Ruff.

MaSSIVE is a trademark of the National Center for Atmospheric Research.

Unix is a registered trademark of USL, Inc.

References

- [1] M. Nelson et al., "The NCAR Mass Storage System", *Proc. Eighth IEEE Symposium on Mass Storage Systems*, 1987.
- [2] 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.
- [3] D. Borman, "TCP/IP Performance at Cray Research", *Proceedings of the Twenty-Third Internet Engineering Task Force*, Corporation for National Research Initiatives, 1992.
- [4] B. Collins et al., "Profiles in Mass Storage: A Tale of Two Systems", *Proc. Ninth IEEE Symposium on Mass Storage Systems*, 1991.
- [5] J. Sloan, "Parcels with TAGS", TN-377+IA, National Center for Atmospheric Research, Boulder Colorado, 1992.
- [6] Open Software Foundation, "DCE Features and Functionality Technical Seminar", *OSF Distributed Computing Environment*, 1989.
- [7] A. Hanushevsky, *Distributed Security in the IEEE Storage Reference Model*, IEEE P1244 Name Server Subcommittee, August 1992.
- [8] J. Morris, "'Make or Take' Decisions in Andrew", *Proceedings of the USENIX Technical Conference*, Texas, February 1988.
- [9] J. Howard et al., "Scale and Performance in a Distributed File System", *ACM Transactions on Computer Systems*, Vol. 6 No. 1, February 1988.
- [10] G. Treese, "Berkeley UNIX on 1000 Workstations: Athena Changes to 4.3BSD", *Proceedings of the USENIX Technical Conference*, Winter 1988.
- [11] J. Sloan, "Context Switch Time Scalability: Benchmark Results", TN-378+STR, National Center for Atmospheric Research, Boulder Colorado, 1992.
- [12] J. Ousterhout et al., "A Trace-Driven Analysis of the UNIX 4.2 BSD File System", *Proceedings of the Tenth ACM Symposium on Operating Systems Principles*, December 1985.
- [13] E. Miller, "Input/Output Behavior of Supercomputer Applications", UCB/CSD 91/616, University of California, January 1991.