Shared File Systems and Fibre Channel

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

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

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

1. Introduction

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

2. A Short History of Distributed File Systems

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

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

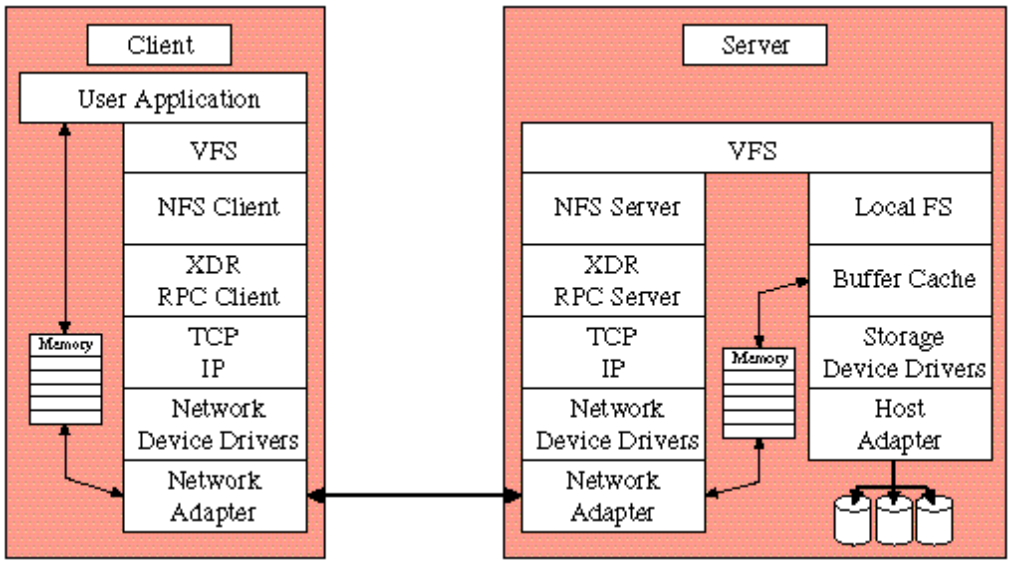

Figure 1: NFS Protocol Stack on Client and Server.

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

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

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

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

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

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

2.1 Network File System (NFS)

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

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

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

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

2.2 The Remote File Sharing system (RFS)

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

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

Crash recovery in RFS is relatively straightforward. A virtual connection is set up when a mount operation is first performed between a client and server. This virtual connection is

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

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

2.3 The Andrew File System

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

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

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

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

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

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

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

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

3.1 Interconnect Topologies

There are three basic topologies supported in Fibre Channel:

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

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

3.2 Physical Characteristics

Though the physical Fibre Channel interface was originally designed to support single- and multi-mode optical connections, the standard has been broadened to include support for copper coax and twisted pair lines run over shorter distances. Single-mode fiber optic connections can be run up to 10 kilometers; cheaper multi-mode connections can be run up to 1 km. Each port includes both a transmitter and receiver: in point-to-point and Fabric topologies the remote connections for the transmitter and receiver end at the same port,

while in the arbitrated loop topology these connections end up at different ports. Full-speed connections run at 1,062.5 Mbps (Megabits per second) which, after protocol overheads are factored in, gives a potential data transfer rate of 100 MBps (megabytes per second). The standard includes a well-defined path to "double-" and "quadruple-speed" which can achieve data transfer rates of 200 MBps and 400 MBps, respectively.

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

3.3 Protocol Layers

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

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

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

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

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

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

4. Shared File Systems

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

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

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

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

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

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

Shared file systems can be classified using the following characteristics:

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

A shared file system is *symmetric* if any client can perform any file system operation without interacting with another client. In *asymmetric* shared file systems a client must first make a request through a *file manager* executing on another client. The file manager

typically manages the file system metadata, checks file access permissions, and provides the client with information necessary to access the data directly on disk via the storage area network. Asymmetric shared file systems are sometimes said to use *third-party transfers* because the file manager acts as a third-party in the transfer between the client and storage device. Once approved the data transfer between client and device is direct.

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

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

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

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

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

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

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

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

4.1 DEC Vaxcluster file system (1984)

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

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

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

4.2 Cray's Shared File System (1994)

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

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

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

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

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

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

4.4 Global File System (GFS)

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

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

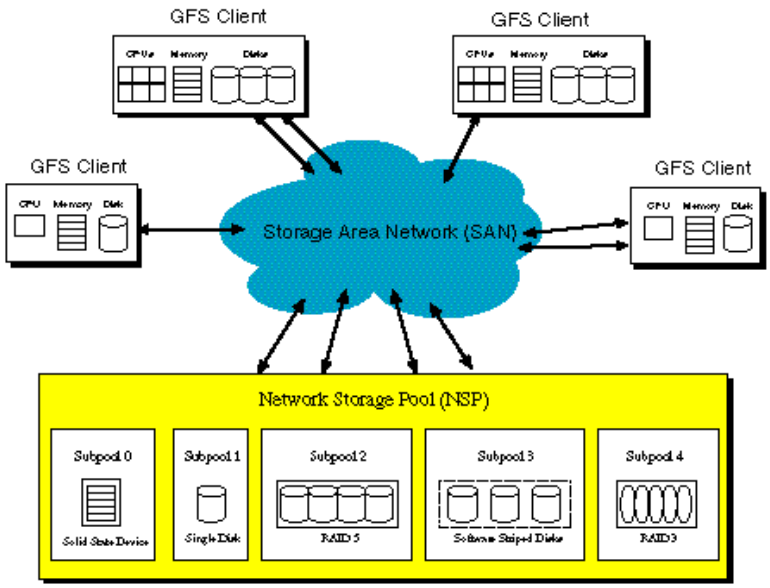

Figure 2: Global File System Distributed Environment.

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

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

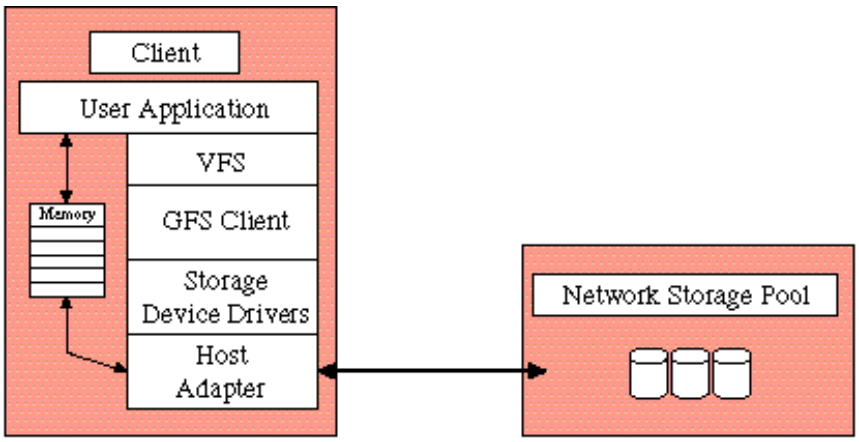

Figure 3: Global File System Protocol "Stack".

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

5. Summary and Conclusions

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

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

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

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

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

Acknowledgements

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

References

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

---

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

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

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

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

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