The Global File System[1]

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

Abstract

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to a level of performance and extensibility so that the previous disadvantages of *shared disk* architectures are no longer valid.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

Introduction

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

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

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

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

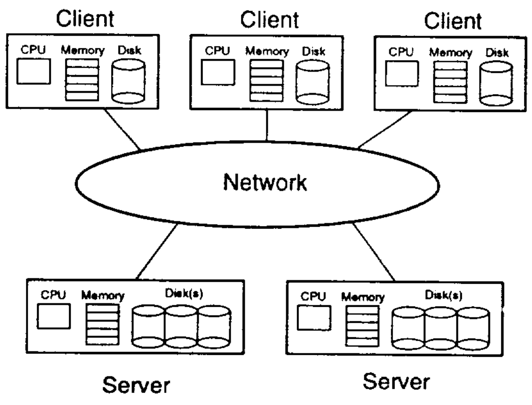

Figure 1: Message-Based

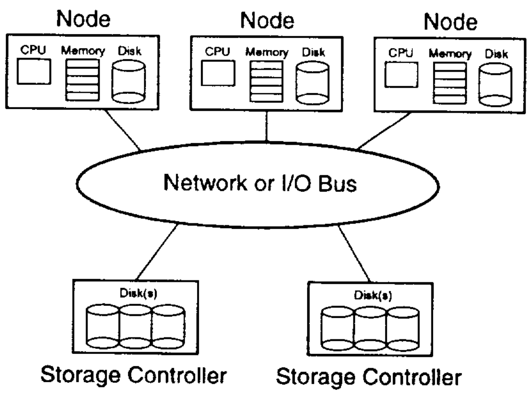

Figure 2: Shared Storage

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

Message-based Distributed File Systems

Sun Network File System

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

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

Sprite File System

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

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

Andrew and Coda File Systems

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

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

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

xFS Serverless Network File System

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

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

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

Shared Storage Distributed File Systems

Digital's VAXClusters VMS

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

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

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

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

Cray's Serverless File System

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

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

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

Message—based Versus Shared Storage

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

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

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

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

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

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

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

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

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

The Global File System

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

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

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

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

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

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

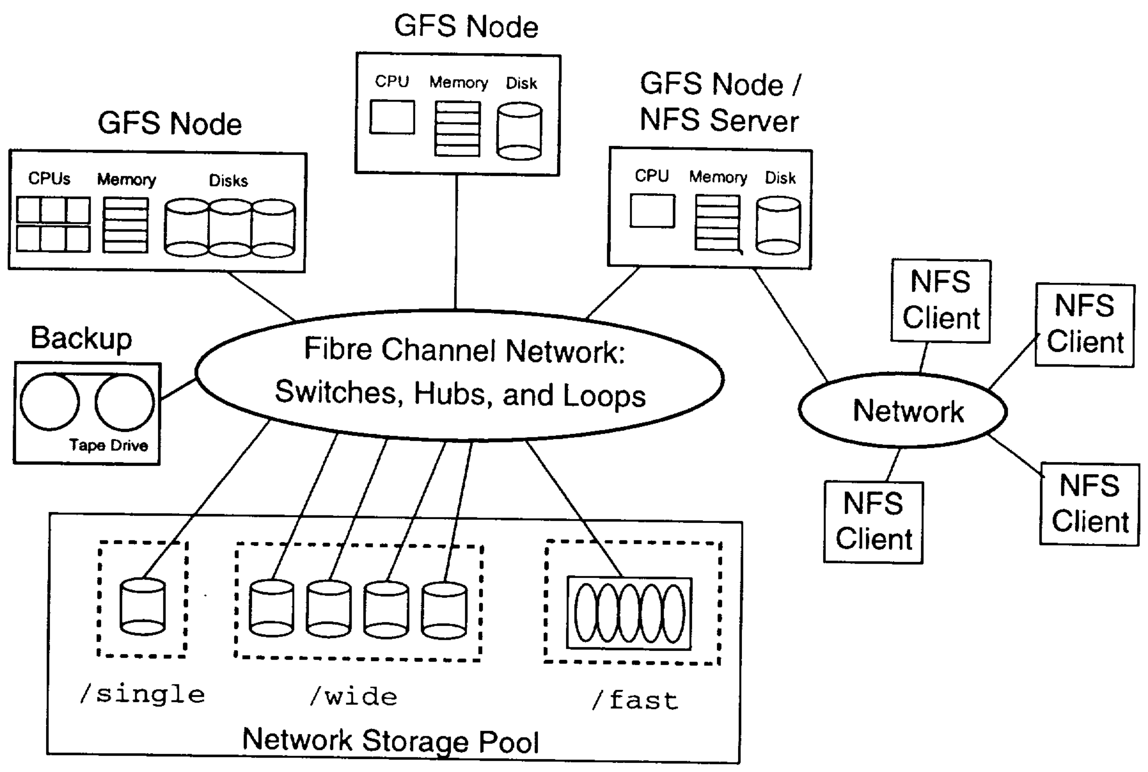

Figure 3: GFS Distributed Environment

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

File System Structure

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

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

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

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

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

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

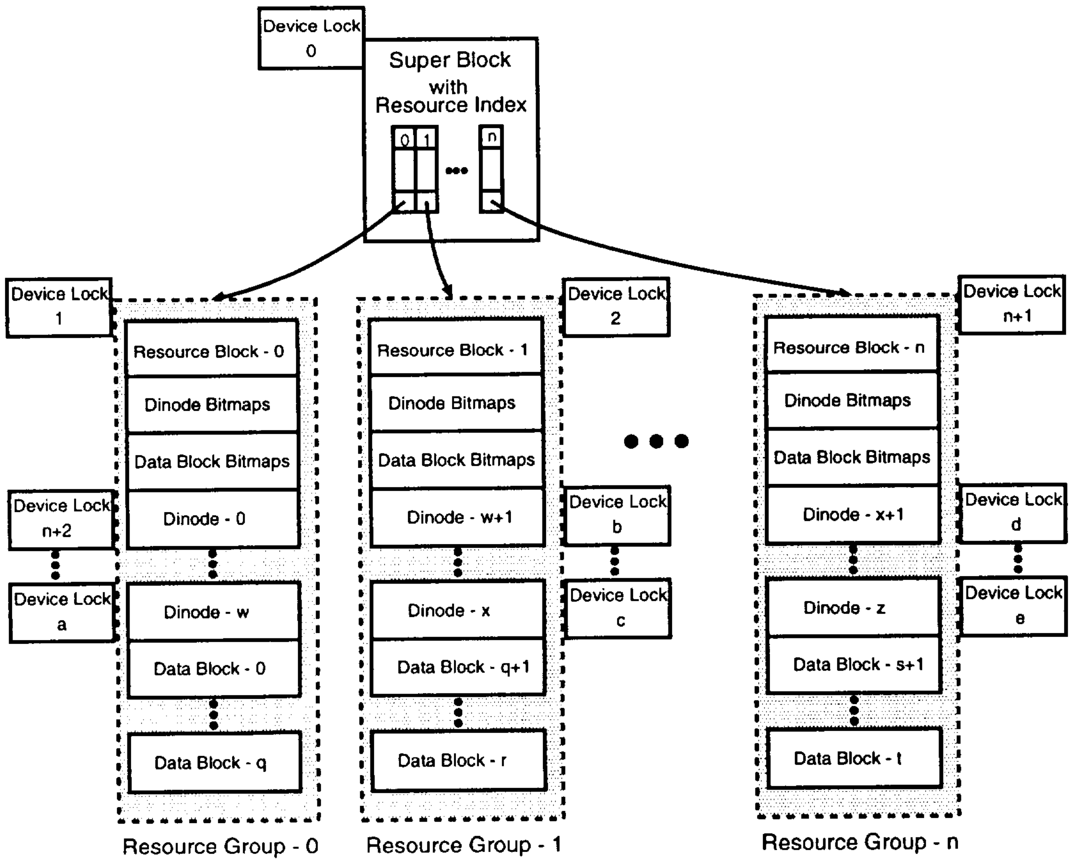

Figure 4: GFS Structure

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

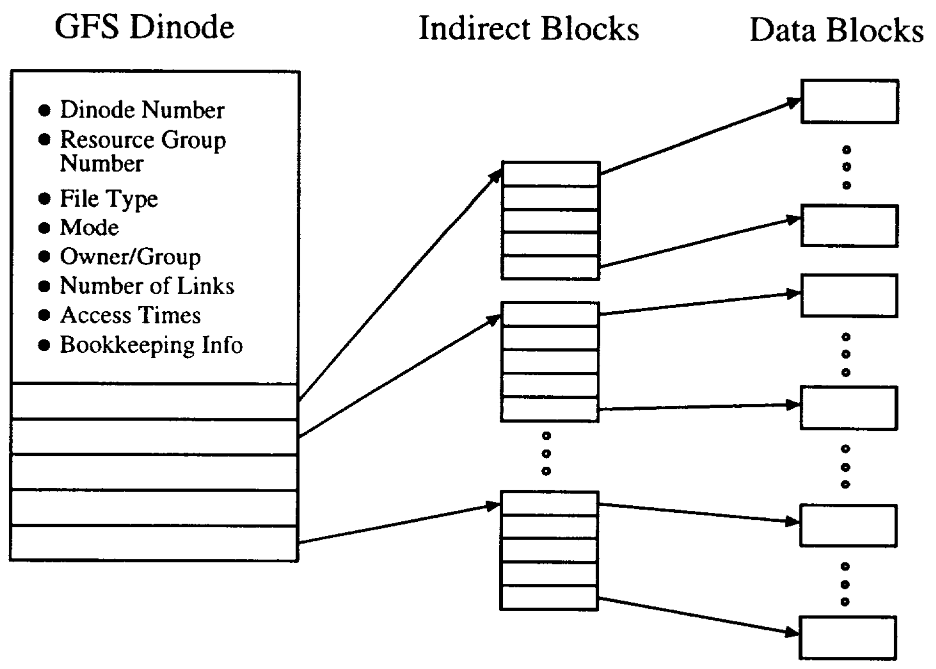

Figure 5: GFS Dinode

Device Locks

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

1. The device lock commands are independent of all other SCSI commands.
2. Devices supporting device locks have no awareness of the nature of data or resource that is locked for mutual exclusion.

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

Lock States

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

Clocks

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

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

Device Failures

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

Node Failures

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

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

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

Deadlocks and Starvation

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

Consistency and Caching

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

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

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

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

Preliminary Results

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

Test Configuration

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

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

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

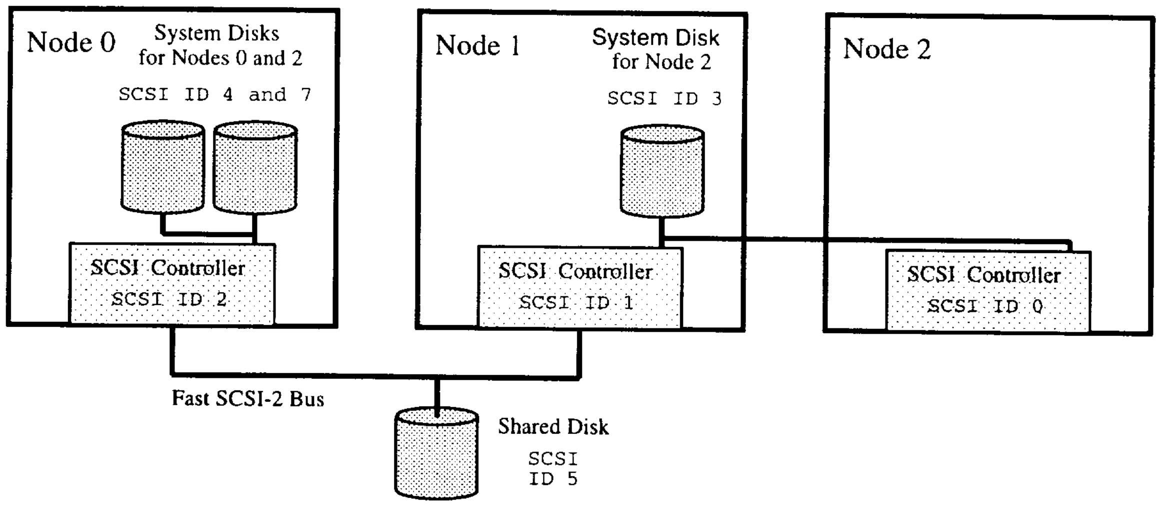

Figure 6: GFS Test Environment

Benchmarks

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

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

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

Table 1: Benchmark Parameters

Parallel SCSI Performance

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

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

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

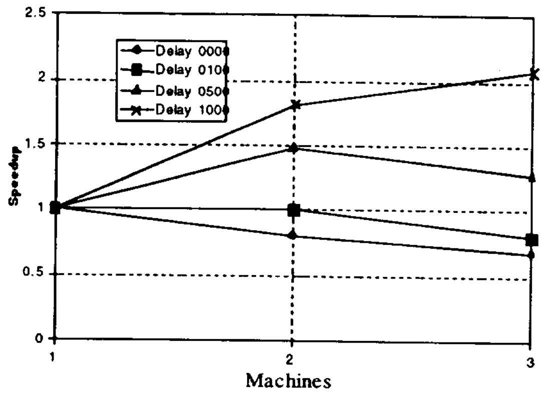

Figure 7: GFS Speedup for 1 MB files Created by 128 KB Requests

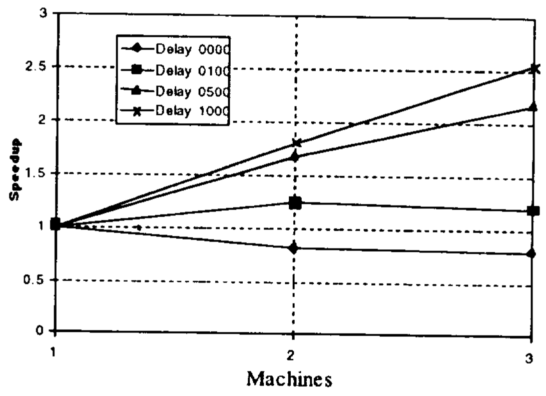

Figure 8: GFS Speedup for 1 MB files Read by 128 KB Requests

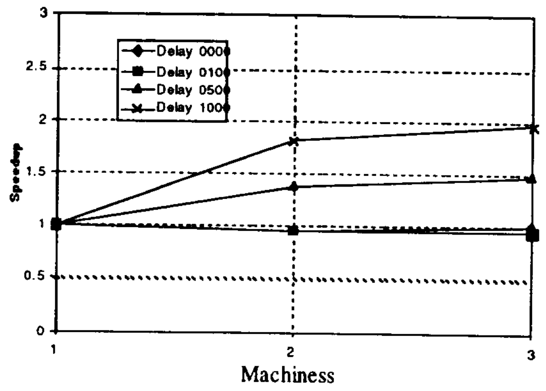

Figure 9: GFS Speedup for 16 MB files Created by 1 MB Requests

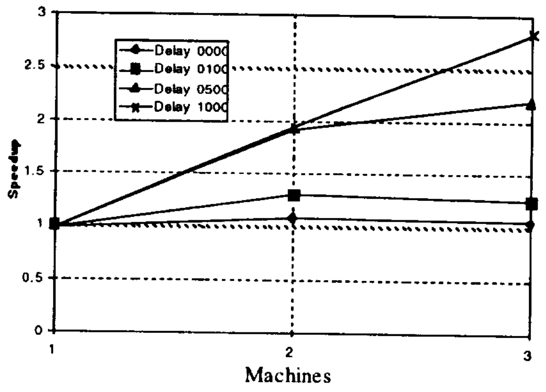

Figure 10: GFS Speedup for 16 MB files Read by 1 MB Requests

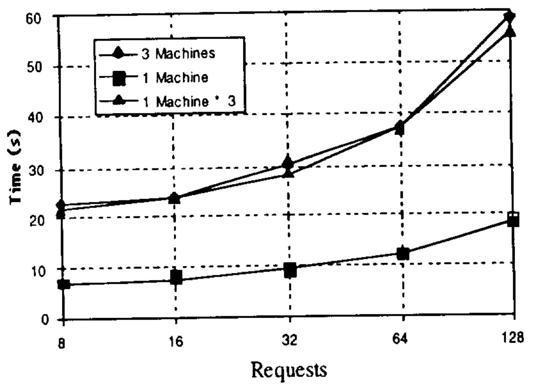

Figure 11: GFS Creates of 16 MB files with no delay

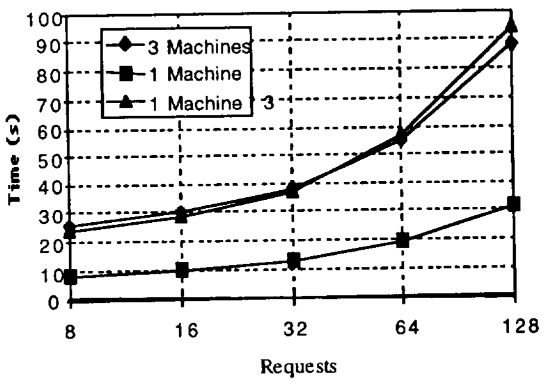

Figure 12: GFS Creates of 16 MB files with 100 ms delay

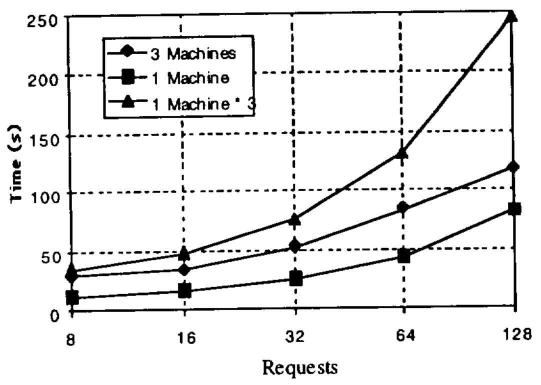

Figure 13: GFS Creates of 16 MB files with 500 ms delay

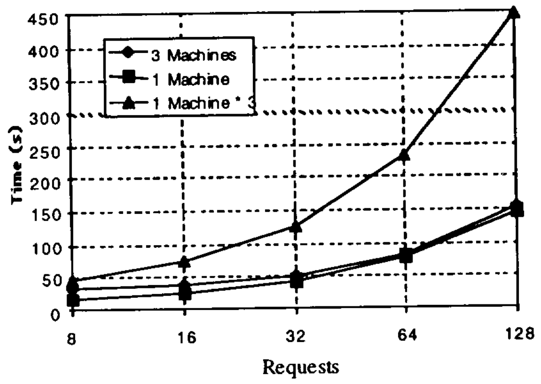

Figure 14: GFS Creates of 16 MB files with 1000 ms delay

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

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

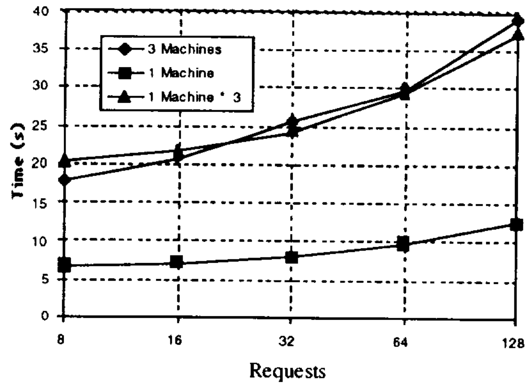

Figure 15: GFS Reads from 16 MB files with no delay

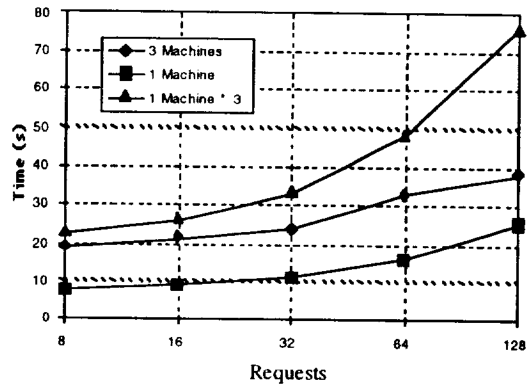

Figure 16: GFS Reads from 16 MB files with 100 ms delay

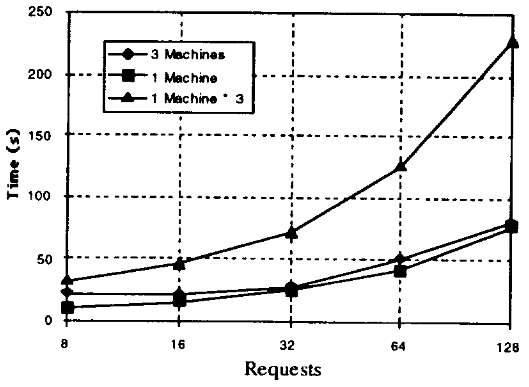

Figure 17: GFS Reads from 16 MB files with 500 ms delay

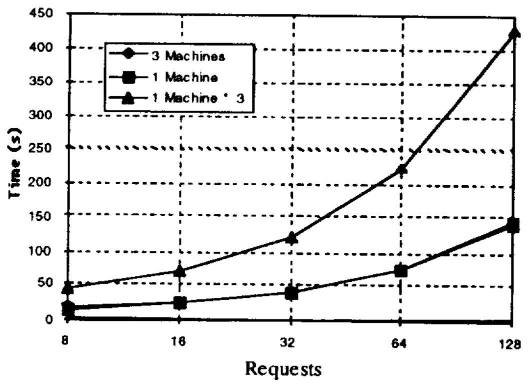

Figure 18: GFS Reads from 16 MB files with 1000 ms delay

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

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

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

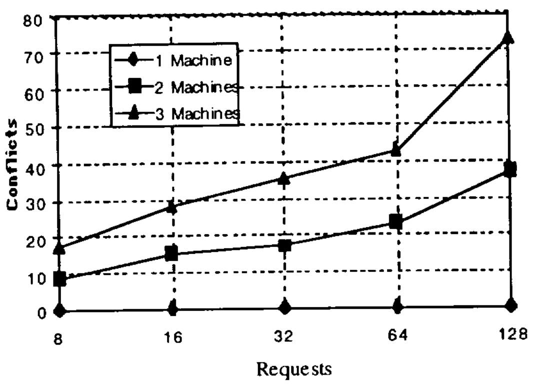

Figure 19: GFS Creates of 16 MB files with no delay

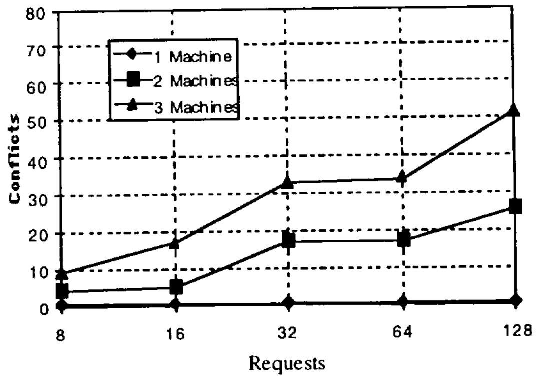

Figure 20: GFS Reads from 16 MB files with no delay

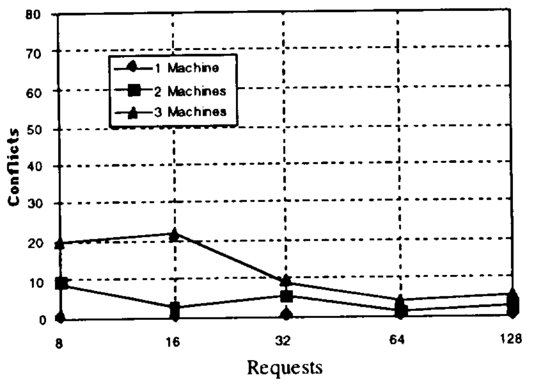

Figure 21: GFS Creates of 16 MB files with 1000 ms delay

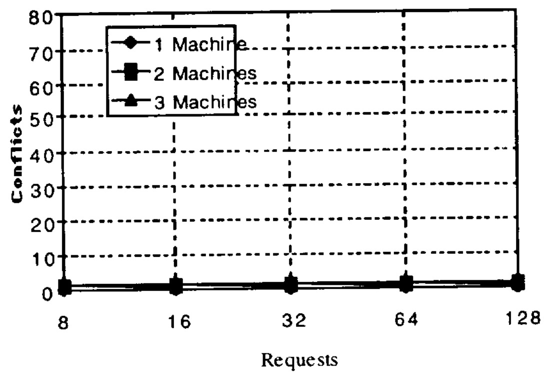

Figure 22: GFS Reads from 16 MB files with 1000 ms delay

NFS Comparison

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

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

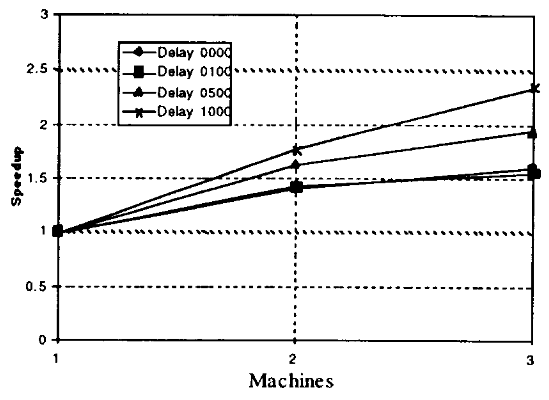

Figure 23: NFS Speedup for 1 MB files Created by 128 KB requests

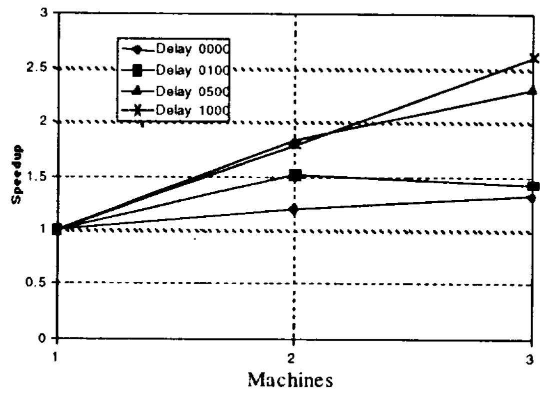

Figure 24: NFS Speedup for 1 MB files Read by 128 KB requests

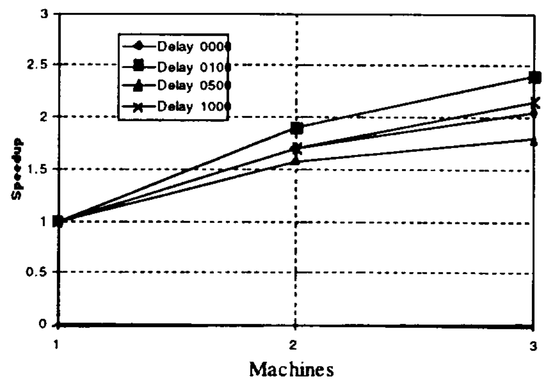

Figure 25: NFS Speedup for 16 MB files Created by 1 MB requests

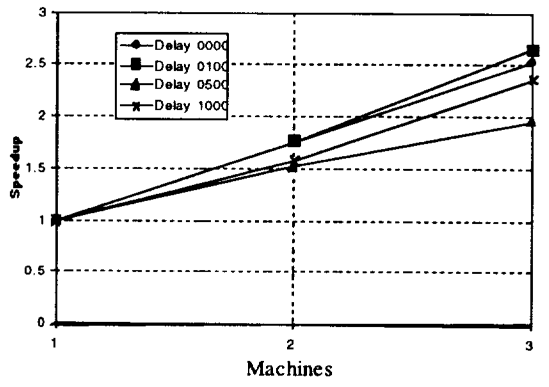

Figure 26: NFS Speedup for 16 MB files Read by 1 MB requests

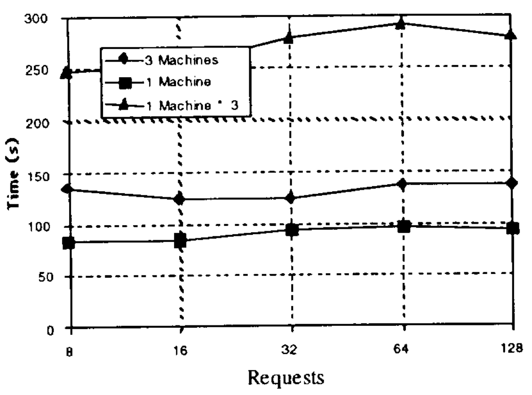

Figure 27: NFS Creates of 16 MB files with no delay

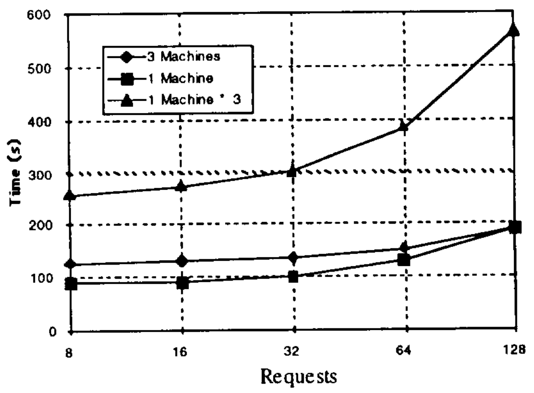

Figure 28: NFS Creates of 16 MB files with 1000 ms delay

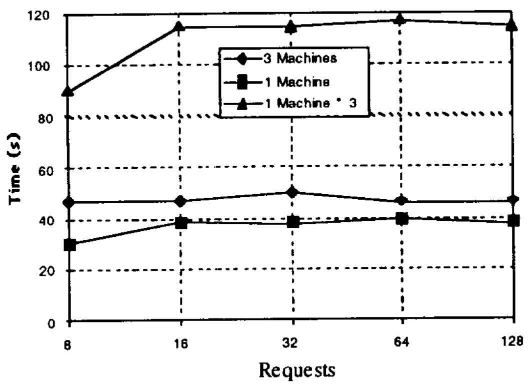

Figure 29: NFS Reads from 16 MB files with no delay

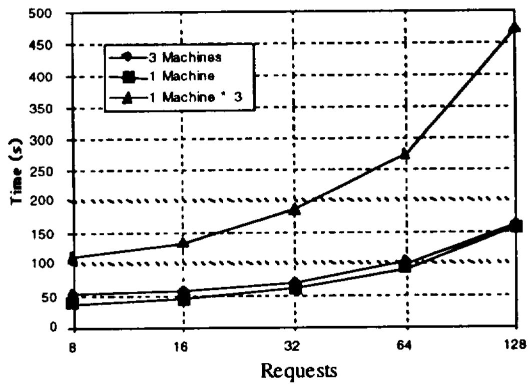

Figure 30: NFS Reads from 16 MB files with 1000 ms delay

Future Work

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

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

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

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

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

Conclusion

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

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

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

Acknowledgments

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

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

References

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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