Appears in *Proceedings of the ACM International Conference on Measurement and Modeling of Computer Systems (Sigmetrics '97)*, Seattle, Washington, June 15-18, 1997.

File Server Scaling with Network-Attached Secure Disks

Garth A. Gibson†, David F. Nagle*, Khalil Amiri*, Fay W. Chang†, Eugene M. Feinberg*, Howard Gobioff†, Chen Lee†, Berend Ozceri*, Erik Riedel*, David Rochberg†, Jim Zelenka†

*Department of Electrical and Computer Engineering
†School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213-3890
garth+nasd@cs.cmu.edu
http://www.cs.cmu.edu/Web/Groups/NASD/

Abstract

By providing direct data transfer between storage and client, network-attached storage devices have the potential to improve scalability for existing distributed file systems (by removing the server as a bottleneck) and bandwidth for new parallel and distributed file systems (through network striping and more efficient data paths). Together, these advantages influence a large enough fraction of the storage market to make commodity network-attached storage feasible. Realizing the technology's full potential requires careful consideration across a wide range of file system, networking and security issues. This paper contrasts two network-attached storage architectures—(1) Networked SCSI disks (NetSCSI) are network-attached storage devices with minimal changes from the familiar SCSI interface, while (2) Network-Attached Secure Disks (NASD) are drives that support independent client access to drive object services. To estimate the potential performance benefits of these architectures, we develop an analytic model and perform trace-driven replay experiments based on AFS and NFS traces. Our results suggest that NetSCSI can reduce file server load during a burst of NFS or AFS activity by about 30%. With the NASD architecture, server load (during burst activity) can be reduced by a factor of up to five for AFS and up to ten for NFS.

1 Introduction

Users are increasingly using distributed file systems to access data across local area networks; personal computers with hundred-plus MIPS processors are becoming increasingly affordable; and the sustained bandwidth of magnetic disk storage is expected to exceed 30 MB/s by the end of the decade. These trends place a pressing need on distributed file system architectures to provide clients with efficient, scalable, high-bandwidth access to stored data. This paper discusses a powerful approach to fulfilling this need. Network-attached storage provides high bandwidth by directly attaching storage to the network, avoiding file server store-and-forward operations and allowing data transfers to be striped over storage and switched-network links.

The principal contribution of this paper is to demonstrate the potential of network-attached storage devices for penetrating the markets defined by existing distributed file system clients, specifically the Network File System (NFS) and Andrew File System (AFS) distributed file system protocols. Our results suggest that network-attached storage devices can improve overall distributed file system cost-effectiveness by offloading disk access, storage management and network transfer and greatly reducing the amount of server work per byte accessed.

We begin by charting the range of network-attached storage devices that enable scalable, high-bandwidth storage systems. Specifically, we present a taxonomy of network-attached storage — server-attached disks (SAD), networked SCSI (NetSCSI) and network-attached secure disks (NASD) — and discuss the distributed file system functions offloaded to storage and the security models supportable by each.

With this taxonomy in place, we examine traces of requests on NFS and AFS file servers, measure the operation costs of commonly used SAD implementations of these file servers and develop a simple model of the change in manager costs for NFS and AFS in NetSCSI and NASD environments. Evaluating the impact on file server load analytically and in trace-driven replay experiments, we find that NASD promises much more efficient file server offloading in comparison to the simpler NetSCSI. With this potential benefit for existing distributed file server markets, we conclude that it is worthwhile to engage in detailed NASD implementation studies to demonstrate the efficiency, throughput and response time of distributed file systems using network-attached storage devices.

In Section 2, we discuss related work. Section 3 presents our taxonomy of network-attached storage architectures. In Section 4, we describe the NFS and AFS traces used in our analysis and replay experiments and report our measurements of the cost of each server operation in CPU cycles. Section 5 develops an analytic model to estimate the potential scaling offered by server-offloading in NetSCSI and NASD based on the collected traces and the measured costs of server operations. The trace-driven replay experiment and the results are the subject of Section 6. Finally, Section 7 presents our conclusions and discusses future directions.

This research was sponsored by DARPA/ITO through ARPA Order D306 under contract N00174-96-0002 and in part by an ONR graduate fellowship. The project team is indebted to generous contributions from the member companies of the Parallel Data Consortium: Hewlett-Packard, Symbios Logic Inc., Data General, Compaq, IBM Corporation, EMC Corporation, Seagate Technology, and Storage Technology Corporation. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of any supporting organization or the U.S. Government.

© 1997 by the Association for Computing Machinery, Inc. Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that new copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted.

To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request Permissions from Publications Dept, ACM Inc. Fax +1 (212) 869-0481, or <permissions@acm.org>.

2 Related Work

Distributed file systems provide remote access to shared file storage in a networked environment [Sandberg85, Howard88, Minshall94]. A principal measure of a distributed file system's cost is the computational power required from the servers to provide adequate performance for each client's work [Howard88, Nelson88]. While microprocessor performance is increasing dramatically and raw computational power would not normally be a concern, the work done by a file server is data- and interrupt-intensive and, with the poorer locality typical of operating systems, faster microprocessors will provide much less benefit than their cycle time trends promise [Ousterhout91, Anderson91, Chen93].

Typically, distributed file systems employ client caching to reduce this server load. For example, AFS clients use local disk to cache a subset of the global system's files. While client caching is essential for high performance, increasing file sizes, computation sizes, and workgroup sharing are all inducing more misses per cache block [Ousterhout85, Baker91]. At the same time, increased client cache sizes are making these misses more bursty.

When the post-client-cache server load is still too large, it can either be distributed over multiple servers or satisfied by a custom-designed high-end file server. Multiple-server distributed file systems attempt to balance load by partitioning the namespace and replicating static, commonly used files. This replication and partitioning is too often ad-hoc, leading to the "hotspot" problem familiar in multiple-disk mainframe systems [Kim86] and requiring frequent user-directed load balancing. Not surprisingly, custom-designed high-end file servers more reliably provide good performance, but can be an expensive solution [Hitz90, Drapeau94].

Experience with disk arrays suggests another solution. If data is striped over multiple independent disks of an array, then a high-concurrency workload will be balanced with high probability as long as individual accesses are small relative to the unit of interleaving [Linvy87, Patterson88, Chen90]. Similarly, striping file storage across multiple servers provides parallel transfer of large files and balancing of high concurrency workloads [Hartman93]; striping of metadata promises further load-balancing [Dahlin95].

Scalability prohibits the use of a single shared-media network; however, with the emergence of switched network fabrics based on high-speed point-to-point links, striped storage can scale bandwidth independent of other traffic in the same fabric [Arnould89, Siu95, Boden95]. Unfortunately, current implementations of Internet protocols demand significant processing power to deliver high bandwidth — we observe as much as 80% of a 233 MHz DEC Alpha consumed by UDP/IP receiving 135 Mbps over 155 Mbps ATM (even with adaptor support for packet reassembly). Improving this bandwidth depends on interface board designs [Steenkiste94, Cooper90], integrated layer processing for network protocols [Clark89], direct application access to the network interface [vonEiken92, Maeda93], copy avoiding buffering schemes [Druschel93, Brustoloni96], and routing support for high-performance best-effort traffic [Ma96, Traw95]. Perhaps most importantly, the protocol stacks resulting from these research efforts must be deployed widely. This deployment is critical because the comparable storage protocols, SCSI, and soon, Fibre Channel, provide cost-effective hardware implementations routinely included in client machines. For comparison, a 175 MHz DEC Alpha consumes less than 5% of its processing power fetching 100 Mbps from a 160 Mbps SCSI channel via the UNIX raw disk interface.

To exploit the economics of large systems resulting from the cobbling together of many client purchases, the xFS file system distributes code, metadata and data over all clients, eliminating the need for a centralized storage system [Dahlin95]. This scheme naturally matches increasing client performance with increasing server performance. Instead of reducing the server workload, however, it takes the required computational power from another, frequently idle, client. Complementing the advantages of filesystems such as xFS, the network-attached storage architectures presented in this paper significantly reduce the demand for server computation and eliminate file server machines from the storage data path, reducing the coupling between overall file system integrity and the security of individual client machines.

As distributed file system technology has improved, so have the storage technologies employed by these systems. Storage density increases, long a predictable 25% per year, have risen to 60% increases per year during the 90s. Data rates, which were constrained by storage interface definitions until the mid-80s, have increased by about 40% per year in the 90s [Grochowski96]. The acceptance, in all but the lowest cost market, of SCSI, whose interface exports the abstraction of a linear array of fixed-size blocks provided by an embedded controller [ANSI86], catalyzed rapid deployment of technology advances, resulting in an extremely competitive storage market.

The level of indirection introduced by SCSI has also led to transparent improvements in storage performance such as RAID; transparent failure recovery; real-time geometry-sensitive scheduling; buffer caching; read-ahead and write-behind; compression; dynamic mapping; and representation migration [Patterson88, Gibson92, Massiglia94, StorageTek94, Wilkes95, Ruemmler91, Varma95]. However, in order to overcome the speed, addressability and connectivity limitations of current SCSI implementations [Sachs94, ANSI95], the industry is turning to high-speed packetized interconnects such as Fibre Channel at up to 1 Gbps [Benner96]. The disk drive industry anticipates the marginal cost for on-disk Fibre Channel interfaces, relative to the common single-ended SCSI interface in use today, to be comparable to the marginal cost for high-performance differential SCSI (a difference similar to the cost of today's Ethernet adapters) while their host adapter costs are expected to be comparable to high-performance SCSI adapters [Anderson95].

The idea of simple, disk-like network-based storage servers whose functions are employed by higher-level distributed file systems, has been around for a long time [Birrel80, Katz92]. The Mass Storage System Reference Model (MSSRM), an early architecture for hierarchical storage subsystems, has advocated the separation of control and data paths for almost a decade [Miller88, IEEE94]. Using a high-bandwidth network that supports direct transfers for the data path is a natural consequence [Kronenberg86, Drapeau94, Long94, Lee95, Menascé96, VanMeter96]. The MSSRM has been implemented in the High Performance Storage System (HPSS) [Watson95] and augmented with socket-level striping of file transfers [Berdahl95, Wiltzius95], over the multiple network interfaces found on mainframes and supercomputers.[1]

[1]Following Van Meter's [VanMeter96] definition of network-attached peripherals, we consider only networks that are shared with general local area network traffic and not single-vendor systems whose interconnects are fast, isolated local area networks [Horst95, IEEE92].

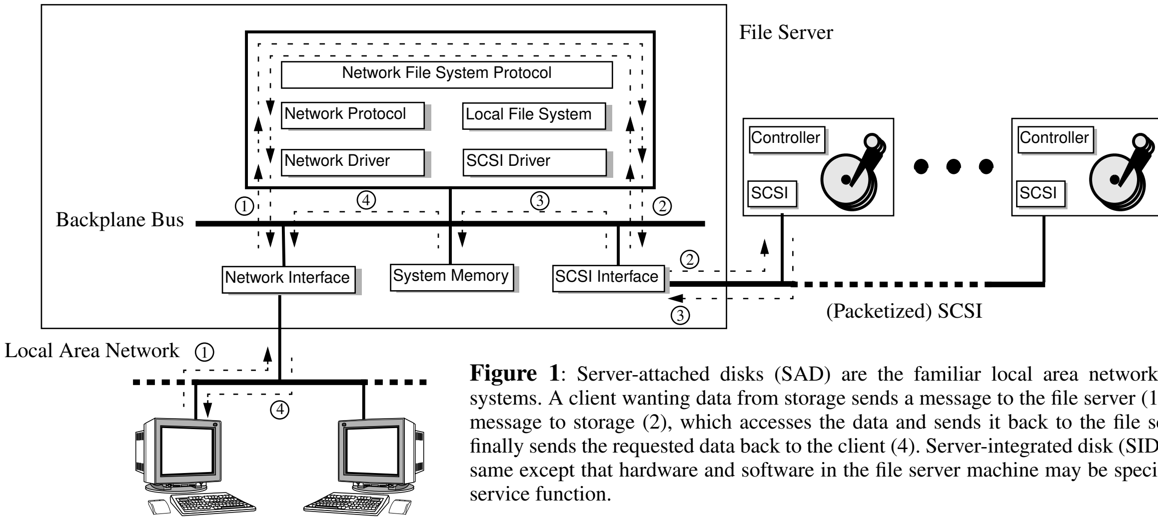

Figure 1: Server-attached disks (SAD) are the familiar local area network distributed file systems. A client wanting data from storage sends a message to the file server (1), which sends a message to storage (2), which accesses the data and sends it back to the file server (3), which finally sends the requested data back to the client (4). Server-integrated disk (SID) is logically the same except that hardware and software in the file server machine may be specialized to the file service function.

Striping data across multiple storage servers with independent ports into a scalable local area network has been advocated as a means of obtaining scalable storage bandwidth [Hartman93]. If the storage servers of this architecture are network-attached devices, rather than dedicated machines between the network and storage, efficiency is further improved by avoiding store-and-forward delays through the server.

Our notion of network-attached storage is consistent with these projects. However, our analysis focuses on the evolution of commodity storage devices rather than niche-market, very high-end systems, and on the interaction of network-attached storage with common distributed file systems. Because all prior work views the network-based storage as a function provided by an additional computer, instead of the storage devices itself, cost-effectiveness has never been within reach. Our goal is to chart the way network-attached storage is likely to appear in storage products, estimate its scalability implications, and characterize the security and file system design issues in its implementation.

3 Taxonomy of Network-Attached Storage

Simply attaching storage to a network underspecifies network-attached storage's role in distributed file systems' architectures. In the following subsections, we present a taxonomy for the functional composition of network-attached storage. Case 0, the base case, is the familiar local area network with storage privately connected to file server machines — we call this *server-attached disks*. Case 1 represents a wide variety of current products, *server-integrated disks*, that specialize hardware and software into an integrated file server product. In Case 2, the obvious network-attached disk design, *network SCSI*, minimizes modifications to the drive command interface, hardware and software. Finally, Case 3, *network-attached secure disks*, leverages the rapidly increasing processor capability of disk-embedded controllers to restructure the drive command interface.

3.1 Case 0: Server-Attached Disks (SAD)

This is the system familiar to office and campus local area networks as illustrated in Figure 1. Clients and servers share a network and storage is attached directly to general-purpose workstations that provide distributed file services.

3.2 Case 1: Server Integrated Disks (SID)

Since file server machines often do little other than service distributed file system requests, it makes sense to construct specialized systems that perform only file system functions and not general-purpose computation. This architecture is not fundamentally different from SAD. Data must still move through the server machine before it reaches the network, but specialized servers can move this data more efficiently than general-purpose machines. Since high performance distributed file service benefits the productivity of most users, this architecture occupies an important market niche [Hitz90, Hitz94]. However, this approach binds storage to a particular distributed file system, its semantics, and its performance characteristics. For example, most server-integrated disks provide NFS file service, whose inherent performance has long been criticized [Howard88]. Furthermore, this approach is undesirable because it does not enable distributed file system and storage technology to evolve independently. Server striping, for instance, is not easily supported by any of the currently popular distributed file systems. Binding the storage interface to a particular distributed file system hampers the integration of such new features [Birrell80].

3.3 Case 2: Network SCSI (NetSCSI)

The other end of the spectrum is to retain as much as possible of SCSI, the current dominant mid- and high-level storage device protocol. This is the natural evolution path for storage devices; Seagate's Barracuda FC is already providing packetized SCSI through Fibre Channel network ports to directly attached hosts [Seagate96]. NetSCSI is a network-attached storage architecture that makes minimal changes to the hardware and software of SCSI disks. File manager software translates client requests into commands to disks, but rather than returning data to the file manager to be forwarded, the NetSCSI disks send data directly to clients, similar to the support for third-party transfers already supported by SCSI [Drapeau94]. The efficient data transfer engines typical of fast drives ensure that the drive's sustained bandwidth is available to clients. Further, by eliminating the file manager from the data path, its workload per active client decreases. However, the use of third-party transfer changes the drive's role in the overall security of a distributed file system. While it is not unusual for distributed file systems to employ a security protocol between clients and

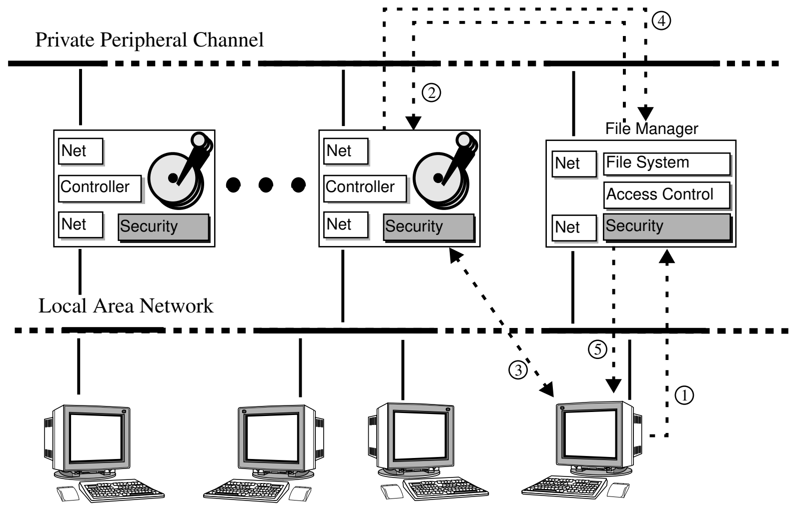

Figure 2: Network SCSI (NetSCSI) is a network-attached disk architecture designed for minimal changes to the disk's command interface. However, because the network port on these disks may be connected to a hostile, broader network, preserving the integrity of on-disk file system structure requires a second port to a private (file manager-owned) network or cryptographic support for a virtual private channel to the file manager. If a client wants data from a NetSCSI disk, it sends a message (1) to the distributed file system's file manager which processes the request in the usual way, sending a message over the private network to the NetSCSI disk (2). The disk accesses data, transfers it directly to the client (3), and sends its completion status to the file manager over the private network (4). Finally, the file manager completes the request with a status message to the client (5).

servers (e.g. Kerberos authentication), disk drives do not yet participate in this protocol.

We identify four levels of security within the NetSCSI model: (1) accident-avoidance with a second private network between file manager and disk, both locked in a physically secure room; (2) data transfer authentication with clients and drives equipped with a strong cryptographic hash function; (3) data transfer privacy with both clients and drives using encryption and; (4) secure key management with a secure coprocessor.

Figure 2 shows the simplest security enhancement to NetSCSI: a second network port on each disk. Since SCSI disks execute every command they receive without an explicit authorization check, without a second port even well-meaning clients can generate erroneous commands and accidentally damage parts of the file system. The drive's second network port provides protection from accidents while allowing SCSI command interpreters to continue following their normal execution model. This is the architecture employed in the SIOF and HPSS projects at LLNL [Wiltzius95, Watson95]. Assuming that file manager and NetSCSI disks are locked in a secure room, this mechanism is acceptable for the trusted network security model of NFS [Sandberg85].

Because file data still travels over the potentially hostile general network, NetSCSI disks are likely to demand greater security than simple accident avoidance. Cryptographic protocols can strengthen the security of NetSCSI. A strong cryptographic hash function, such as SHA [NIST94], computed at the drive and at the client would allow data transfer authentication (i.e., the correct data was received only if the sender and receiver compute the same hash on the data).

For some applications, data transfer authentication is insufficient, and communication privacy is required. To provide privacy, a NetSCSI drive must be able to encrypt and decrypt data. NetSCSI drives can use cryptographic protocols to construct private virtual channels over the untrusted network. However, since keys will be stored in devices vulnerable to physical attack, the servers must still be stored in physically secure environments. If we go one step further and equip NetSCSI disks with secure coprocessors [Tygar95], then keys can be protected and all data can be encrypted when outside the secure coprocessor, allowing the disks to be used in a variety of physically open environments. There are now a variety of secure coprocessors [NIST94a, Weingart87, White87, National96] available, some of which promise cryptographic accelerators sufficient to support single-disk bandwidths.

3.4 Case 3: Network-attached Secure Disks (NASD)

With network-attached secure disks, we relax the constraint of minimal change from the existing SCSI interface and implementation. Instead we focus on selecting a command interface that reduces the number of client-storage interactions that must be relayed through the file manager, offloading more of the file manager's work without integrating file system policy into the disk.

Common, data-intensive operations, such as reads and writes, go straight to the disk, while less-common ones, including namespace and access control manipulations, go to the file manager. As opposed to NetSCSI, where a significant part of the processing for security is performed on the file manager, NASD drives perform most of the processing to enforce the security policy. Specifically, the cryptographic functions and the enforcement of manager decisions are implemented at the drive, while policy decisions are made in the file manager. Because clients directly request access to data in their files, a NASD drive must have sufficient metadata to map and authorize the request to disk sectors. Authorization, in the form of a time-limited capability applicable to the file's map and contents, should be provided by the file manager to protect higher-level file systems' control over storage access policy. The storage mapping metadata, however, could be provided dynamically [VanMeter96a] by the file manager or could be maintained by the drive. While the latter approach asks distributed file system authors to surrender detailed control over the layout of the files they create, it enables smart drives to better exploit detailed knowledge of their own resources to optimize data layout, read-ahead, and cache management [deJonge93, Patterson95, Golding95]. This is precisely the type of value-added opportunity that nimble storage vendors can exploit for market and customer advantage. With mapping metadata at the drive controlling the layout of files, a NASD drive exports a namespace of file-like objects. Because control of naming is more appropriate to the higher-level file system, pathnames are not understood at the drive, and pathname resolution is split between the file manager and client. While a single drive object will suffice to represent a simple client file, multiple objects may be logically linked by the file system into one client file. Such an interface provides support for banks of

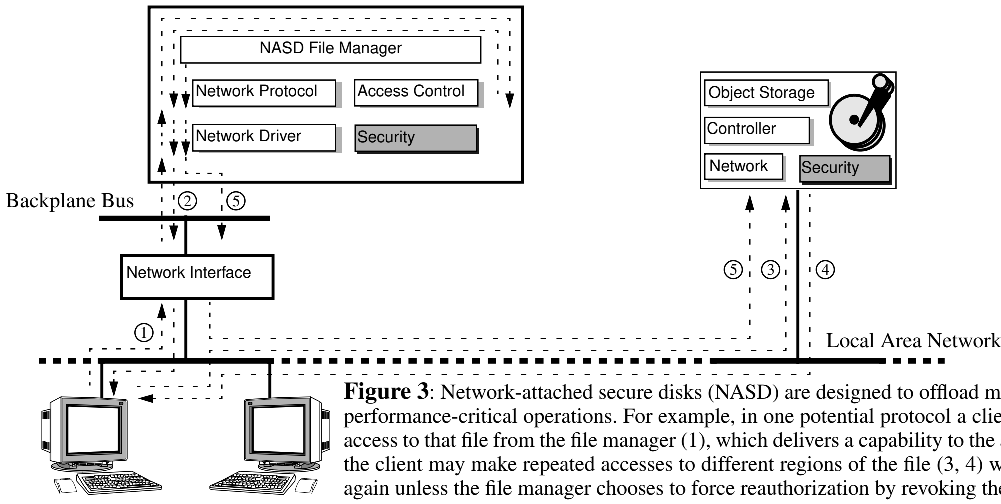

Figure 3: Network-attached secure disks (NASD) are designed to offload more of the file system's simple and performance-critical operations. For example, in one potential protocol a client, prior to reading a file, requests access to that file from the file manager (1), which delivers a capability to the authorized client (2). So equipped, the client may make repeated accesses to different regions of the file (3, 4) without contacting the file manager again unless the file manager chooses to force reauthorization by revoking the capability (5).

striped files [Hartman93], Macintosh-style resource forks, or logically-contiguous chunks of complex files [deJong93].

As an example of a possible NASD access sequence, consider a file read operation depicted in Figure 3. Before issuing its first read of a file, the client authenticates itself with the file manager and requests access to the file. If access is granted, the client receives the network location of the NASD drive containing the object and a time-limited capability to access the object and for establishing a secure communications channel with the drive. After this point, the client may directly request access to data on NASD drives, using the appropriate capability [Gobioff96].

In addition to offloading file read operations from the distributed file manager, later sections will show that NASD should also offload file writes and attributes reads to the drive. High-level file system policies, such as access control and cache consistency, however, remain the purview of the file manager. These policies are enforced by NASD drives according to the capabilities controlled by the file manager.

3.5 Summary

This taxonomy, summarized in Table 1, splits into two classes — SAD and SID offer a specific distributed file system while NetSCSI and NASD offer enhanced storage interfaces. The difference between SID and NASD merits further consideration. Many of the optimizations we propose for NASD, such as shortened data paths and specialized protocol processing, can also be implemented in a SID architecture. However, SID binds storage to a particular distributed file system, requires higher-level (or multiple-SID) file management to offer network striped files and, by not evolving the drive interface, inhibits the independent development of drive technology. For the rest of this paper, we focus on SAD, NetSCSI, and NASD and present a coarse-grained estimate of the potential benefit of network-attached storage. The results suggest that by exploiting the processing power available in next generation storage devices, computation required from the file manager machines can be dramatically reduced, enabling the per-byte cost of distributed file service to be reduced.

	Case 0	Case 1	Case 2	Case 3
FM per byte	X	X		
FM per operation			X	
FM on open/close				X
specialization		X	X	X

Table 1. Comparison of network-attached storage architectures. SAD and SID require the file manager (file server) to handle each byte of data, but SID allows specialization of the hardware and software to file service. NetSCSI allows direct transfers to clients, but requires file manager interaction on each operation to manage metadata.

4 Analysis of File System Workload

To develop an understanding of performance parameters critical to network-attached storage, we performed a series of measurements to (1) characterize the behavior and cost of AFS and NFS distributed file server functionality; and (2) identify and subset busy periods during which server load is limiting.

4.1 Trace Data

Our data is taken from NFS and AFS file system traces summarized in Table 2. The NFS trace [Dahlin94] records the activity of an Auspex file server supporting 231 client machines over a one week period at the University of California at Berkeley[2]. The AFS trace records the activity of our laboratory's Sparcstation 20 AFS server supporting 250 client machines over a one month period[3]

	NFS trace	AFS trace
Number of client machines	231	250
Total number of requests	6,676,479	1,615,540
Read data transferred (GB)	8.1	2.9
Write data transferred (GB)	2.0	1.6
Trace period	9/20/93-9/24/93 40 hours	9/9/96-10/3/96 435 hours[3]

Table 2. Description of the traces used in the experiments. The NFS trace was collected in a study performed at the University of California at Berkeley. The AFS trace was collected by logging requests at the AFS file server in our laboratory.

[2]Some attribute reads were removed from the NFS trace by the Berkeley researchers based on a heuristic for eliminating excessive cache consistency traffic. Because this change is pessimistic to our proposed architecture, we choose to continue to use these traces, already familiar to the community, rather than collect new traces.

[3]The trace covers three periods of activity - 9/9-10, 9/13-15, and 9/20-10/3.

Table 3(a) - NFS Trace Operations

Trace Record	NFS Operations	Description	Percent	Quantity (millions)	% of Cycles
AttrRead	getattr	Get metadata information	42.5	2.84	11.8
AttrWrite	setattr	Update metadata information	0.3	0.02	0.3
BlockRead	read	Get data from server	20.4	1.36	31.6
BlockWrite	write	Send data to server	4.2	0.28	19.3
DirRead	lookup, readdir	Convert filename to filehandle, get directory entries	31.4	2.10	35.5
DirReadWrite	create, mkdir, rename, etc.	Create new files/directories, rename, etc.	1.0	0.07	1.0
DeleteWrite	unlink, rmdir	Remove file/directory	0.2	0.01	0.4

Table 3(b) - NFS Cost Measurements

Data Size (bytes)	Read Cycles (thousands)	Write Cycles (thousands)
1	54	117
1K	61	—
2K	68	—
4K	78	148
8,000	100	199

Table 3(c)

Operation	Cycles (thousands)
getattr	33
setattr	64
lookup	50
readdir (1 entry)	63
readdir (40 entries)	105

Table 3(d)

Operation	Cycles (thousands)
create	81
unlink (last link)	135

Table 3: Distribution and average costs of NFS operations. Cycle counts were taken on an ATOMized DEC 3000/400 (133 MHz, 64 MB of memory, Digital UNIX 3.2c) kernel, including NFSv3 server functionality. ATOM overhead was calculated and removed. The server's caches were warmed, and trials that produced misses in the buffer cache were discarded. The write and create operations were measured using a RAM-based file system.

Both the NFS and AFS traces document each client request with an arrival timestamp, a unique client host id, and an indication of the request type. The AFS trace records the exact type of primitive AFS file system request and also includes a response timestamp. The NFS trace only records the general class of the issued request which leaves some ambiguity in determining exactly which primitive NFS requests were issued (e.g., a request recorded as a directory read may have been either a lookup or readdir request).

The original NFS trace is dominated by overnight backup activity. Since users are mostly insensitive to backup performance, and this is not a major concern of this study, we exclude this activity by only including requests timestamped Monday through Friday between 9am and 5pm in our data set. The AFS trace does not include any backup activity because AFS backups are handled by a separate task on the file server machine.

4.2 Cost of NFS and AFS Operations

Our trace data captures the types and relative frequencies of client requests but does not include the amount of CPU work performed by the file server in handling each request. To estimate this cost, we measured NFS and AFS server code paths on Digital Equipment Alpha workstations. Specifically, we used the ATOM binary annotation tool [Srivastava94] and the Alpha's on-chip cycles counters to identify the code paths traversed and measure the work required for each type of primitive file system operation.

To minimize measurement overhead and improve accuracy, cost measurements were taken in two steps. First, we used ATOM to annotate the entry and exit points of each procedure and issued specific requests, producing a dynamic call graph for each primitive operation. Then we re-annotated the server routines, starting at packet arrival and ending at response packet dispatch, limiting annotation to the critical components of each operation's code path. For each operation, file system requests were repeatedly applied to the selectively annotated server, generating traces that recorded code-path execution times. Measurements were repeated for a range of request sizes where appropriate and all the measurements are summarized in Table 3(b,c,d) and Table 4(b,c,d).

4.3 Relative Importance of NFS and AFS Operations

Table 3(a) and Table 4(a) report the frequency distribution of various server operations for the traces. Each table describes the types of primitive operations and reports their frequencies and the total number of occurrences in the trace. This data shows that attribute read requests (AttrRead, FetchStatus, BulkStatus) are the most frequently executed operations. While frequency statistics emphasize attribute operations, the cycle count data indicate that data movement can place a significantly larger per request burden on the server CPU.

To assess the relative importance of various primitive operations in the total workload applied to a file server, we estimate the total amount of work performed by a server per request type during the execution of each trace. Specifically, we estimate the total server workload per operation type by multiplying the per-type count of occurrences by the measured average per-type cycle count. Since the NFS trace groups certain operation types together, as indicated by Column 2 of Table 3(a) we use a representative member of each group to perform our NFS calculations. The relative importance of operation types can be deduced from the percentage of the server load attributable to each type, as shown in the last column of Table 3(a) and Table 4(a). These calculations show that data-moving operations contribute 51% of the NFS workload and 36% of the AFS workload. Because these fractions are far short of 100%, the performance gained by directly moving data between clients and disks may be limited [Drapeau94]. As the next subsection shows, this limits the benefit of NetSCSI for offloading file manager workload and motivates the design of a NASD drive interface.

Table 4(a) - AFS Trace Operations

AFS Operation	Description	Percent	Quantity (thousands)	% of Cycles
FetchStatus	Get metadata information	65.1	1052.2	39.3
BulkStatus	Perform a group of FetchStatus operations	5.8	93.4	10.9
StoreStatus	Update metadata information	2.5	40.4	2.2
FetchData	Get data from server	13.9	224.0	27.9
StoreData	Send data to server	3.8	61.5	8.5
CreateFile	Create a new file	1.7	27.0	2.4
Rename	Rename a file/directory	0.6	10.4	0.9
RemoveFile	Remove a file	1.5	25.0	2.4
Others	ACL manipulation, symbolic links, directory creation/deletion, lock management, etc.	5.0	81.6	5.4

Table 4(b) - AFS Cost Measurements

	Cycles according to Size of Operation (thousands)										
Operation	**0**	**1**	**512**	**1K**	**2K**	**4K**	**8K**	**16K**	**32K**	**64K**	**1M**
FetchData	—	179	192	191	204	270	330	439	788	1,544	—
StoreData	259	—	291	303	363	371	410	578	750	1,242	16,752
RemoveFile	—	331	396	396	410	411	412	414	429	452	1,053

Table 4(c)

BulkStatus Size (directory entries)	Cycles (thousands)
1	151
3	178
10	324
20	578
25	662

Table 4(d)

Operation	Cycles (thousands)
FetchStatus	128
StoreStatus	189
CreateFile	307
Rename	285
Others	227

Table 4: Distribution and average costs of AFS operations. Cycle counts were taken on a DEC 3000/500 (150MHz, 128 MB of memory, Digital UNIX 3.2c) running an ATOMized AFS version 3.4 server. ATOM tracing overhead was negligible compared to other system-level effects on the server. The server's caches were warmed and trials that produced misses in the local file system cache were discarded. The number of cycles for "Others" was estimated as the average of the four size-independent operations that were measured individually (FetchStatus, StoreStatus, CreateFile, Rename).

4.4 Busy Client-Minutes

A distributed file system scales if an increase in aggregate client demand, and the corresponding increase in storage capacity and bandwidth, does not result in a decrease in client-observed performance. In a previous study [Riedel96], we examined the correlation between hourly averages of client response times, network round-trip times and server load. Users may be satisfied with their response times when servers are idle, but experience periods of dramatically longer response times which correlate with periods of high server load. Since client dissatisfaction is strongly determined by prolonged periods of considerably higher than average response time, this study focuses on server performance during such bursts of high load. For such a burst to have client impact, it must persist for a sufficiently long time. In this paper we have chosen to examine load during one minute intervals — long enough for interactive users to identify a slowdown, but not so long that poor performance during bursts is hidden by overall averages. Our previous study also observed that periods of high server load may exhibit a different distribution of request types — data movement is more prevalent. In order to capture the distribution of operations during these critical bursty periods, we restrict the rest of our analysis to

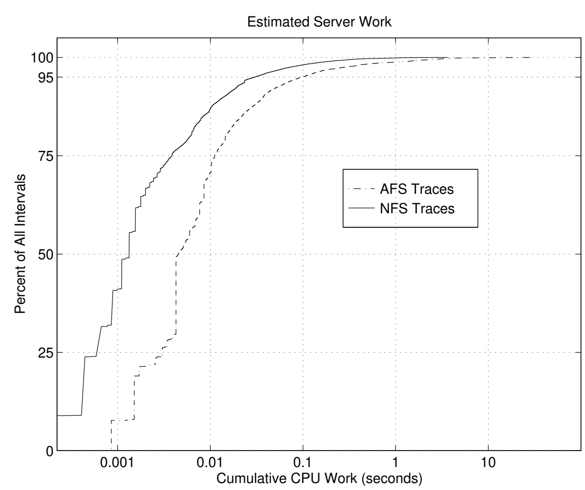

Figure 4: Cumulative distribution of estimated server work for NFS and AFS intervals. The graph shows that 98% of NFS client-minutes and 95% of AFS client-minutes require less than 0.1 seconds of estimated server work.

	NFS		AFS	
	Number	% of total	Number	% of total
Busy client-minutes	4,636	2	2,809	5
Client machines	135	58	78	31
Requests	3,730,031	56	1,199,419	74
Read data (GB)	4.8	59	2.8	96
Write data (GB)	1.7	84	1.3	84

Table 5: Statistics for the top 2% of NFS client-minutes, and top 5% of AFS client-minutes, as measured by estimated work.

the busiest one-minute intervals as measured by the amount of work detailed in Section 4.3.

Based on this metric and the data in Figure 4, we chose to restrict analysis to *client-minutes* (single minutes of a single client's activity) that consume more than 0.1 seconds of server CPU (top 2% of NFS and top 5% of AFS client-minutes). Table 5 summarizes these busy client-minutes.

5 Analytic Model

When a distributed file system is ported to NetSCSI or NASD environments, the disposition of client requests is adjusted according to the goals described in Section 3. The principal benefit we expect for an existing file system such as NFS or AFS is a more cost-effective scaling of throughput by a reduction in the file manager load. In this section, we develop a simple estimate of this scaling. Following the work estimates of Section 4.3, where total file manager work is estimated as the sum of operation costs weighted by the frequency of each operation, we derive estimates of the NetSCSI and NASD file manager work done by NFS and AFS operations by approximating these costs with SAD operations which accomplish similar amounts of work, as reported in Table 6. These estimates are only coarse approximations, but provide a reasonable estimate of the potential benefit of network-attached storage over SAD in terms of file manager scaling.

In the NetSCSI model, the only change from SAD is that the read/write datapath avoids the file manager. However, each read or write request must still be authorized and translated to NetSCSI block addresses. As we see in Table 7, this severely limits the scalability of NetSCSI even though we optimistically model the manager cost of read or write as a simple attribute read in SAD. Specifically, this model estimates file manager work with NetSCSI to be at least two-thirds and three-quarters as much as with SAD during busy NFS and AFS client-minutes.

In our NASD model, all read operations, including attribute and directory reads, are sent directly to the NASD drive. We further assume that NFS clients in NASD systems replace directory lookup operations with NASD (directory) object reads and execute the lookup locally. The data in file writes are also sent directly to the NASD drive. However, in order to support AFS consistency semantics, data writes generate an additional request to the file manager. This request, which would allow the AFS file manager to perform the appropriate consistency maintenance (e.g. breaking callbacks), is estimated to require the same work as an attribute read request. NFS, which has a weaker consistency model, does not require this additional request. For attribute and directory writes, we assume that clients must send their requests to the file manager. To estimate the file manager's pre-authorization and capability setup work prior to any access, we introduced a NASD open request which we emulate with an attribute read operation. Since NASD capabilities are valid for a limited time (twenty-four hours in this model), unless revoked by a change in access rights (an operation that is extremely rare in our traces), transforming the traces in this way adds one additional operation when a file is first referenced on a given day. Finally, remove operations, whose deallocation work is done by the NASD drive, require file manager work comparable to the removal of an empty file.

For AFS, Table 7 shows that NASD systems may reduce file manager workload during busy client-minutes by a factor of two over NetSCSI systems and a factor of three over SAD systems. For NFS, where directory and attribute reads dominate the workload, file managers using NASD drives may benefit from a factor of fourteen decrease in file manager load over SAD systems.

6 Replay Experiment

The analytic model neglects several factors. Particularly concerning is its inability to account for system-level activity (e.g. page faults, scheduler activity, thread overhead, queueing effects) that could significantly impact the behavior and performance of NetSCSI and NASD systems. Given our goal of justifying further implementation studies, we chose to explore system overheads and interactions by replaying the traces, modified according to Table 6 to coarsely model the work of a NetSCSI or NASD server, against existing SAD implementations. This experiment allows us to measure expected file manager load under SAD, NetSCSI and NASD,

NFS trace	SAD	NetSCSI	NASD
AttrRead	getattr	getattr	—
AttrWrite	setattr	setattr	setattr
BlockRead	read(size)	getattr	—
BlockWrite	write(size)	getattr	—
DirRead	lookup	lookup	—
DirRW	create	create	create
DeleteWrite	remove(size)	remove(size)	remove(0 byte)
NasdOpen	—	—	getattr

AFS trace	SAD	NetSCSI	NASD
FetchStatus	FetchStatus	FetchStatus	—
StoreStatus	StoreStatus	StoreStatus	StoreStatus
FetchData	FetchData(size)	FetchStatus	—
StoreData	StoreData(size)	FetchStatus	FetchStatus
RemoveFile	RemoveFile(size)	RemoveFile(size)	RemoveFile(0 byte)
BulkStatus	BulkStatus	BulkStatus	—
NasdOpen	—	—	FetchStatus

Table 6. Description of what the operations in the filesystem traces translate to in the SAD, NetSCSI and NASD models. The tables list the operations as recorded in the trace and the corresponding RPC request issued by a client during replay for each of SAD, NetSCSI and NASD and used in the analytic calculations to estimate server load in each model. The operations not listed for AFS are the same across SAD, NetSCSI and NASD. The last row in each table corresponds to the NASD open operation, which we added to the traces.

NFS Operation	Count in top 2% by work (thousands)	SAD		NetSCSI		NASD	
		Cycles (billions)	%	Cycles (billions)	%*	Cycles (billions)	%*
Attr Read	792.7	26.4	11.8%	26.4	11.8%	0.0	0.0%
Attr Write	10.0	0.6	0.3%	0.6	0.3%	0.6	0.3%
Block Read	803.2	70.4	31.6%	26.8	12.0%	0.0	0.0%
Block Write	228.4	43.2	19.4%	7.6	3.4%	0.0	0.0%
Dir Read	1577.2	79.1	35.5%	79.1	35.5%	0.0	0.0%
Dir RW	28.7	2.3	1.0%	2.3	1.0%	2.3	1.0%
Delete Write	7.0	0.9	0.4%	0.9	0.4%	0.9	0.4%
Open	95.2	0.0	0.0%	0.0	0.0%	12.2	5.5%
Total	**3542.4**	**223.1**	**100.0%**	**143.9**	**64.5%**	**16.1**	**7.2%**

AFS Operation	Count in top 5% by work (thousands)	SAD		NetSCSI		NASD	
		Cycles (billions)	%	Cycles (billions)	%*	Cycles (billions)	%*
FetchStatus	770.5	98.6	37.9%	98.6	37.9%	0.0	0.0%
BulkStatus	91.3	36.6	14.1%	36.6	14.1%	0.0	0.0%
StoreStatus	16.2	3.1	1.2%	3.1	1.2%	3.1	1.2%
FetchData	193.7	83.7	32.1%	24.8	9.5%	0.0	0.0%
StoreData	23.1	15.1	5.8%	3.0	1.1%	3.0	1.1%
CreateFile	12.1	3.7	1.4%	3.7	1.4%	3.7	1.4%
Rename	6.4	1.8	0.7%	1.8	0.7%	1.8	0.7%
RemoveFile	14.6	4.8	1.9%	4.8	1.9%	4.8	1.9%
Others	57.3	13.0	5.0%	13.0	5.0%	13.0	5.0%
Open	480.8	0.0	0.0%	0.0	0.0%	61.5	23.6%
Total	**1665.9**	**260.5**	**100.0%**	**189.4**	**72.7%**	**90.9**	**34.9%**

Table 7: Estimated work performed by the NFS and AFS file managers to handle requests issued during busy client-minutes. This table reports the estimates of our analytic model comparing the relative scalability of file managers in SAD, NetSCSI, and NASD environments. "%" in the NetSCSI and NASD columns represents the percentage difference between each particular NetSCSI or NASD operation cycle count and the SAD total cycle count.

capturing system-level activities not accounted for in the analytic model and more accurately estimating the increase in scalability possible with NetSCSI- and NASD-based systems.

6.1 Experiment

The replay environment, as illustrated in Figure 5, is composed of a single file manager (i.e. an NFS or AFS server) and several host workstations. We refer to these host workstations, used to replay (modified) trace requests to the file manager, as *replay hosts*. Each replay host merges several client-minute traces which it then replays using an open-loop request-issue model where multiple threads replay each of the requests according to the issue timestamps. When the total load applied is well under the server's capability, the timing of operations approximates the original traces and the replay completes in about one minute. However, as the number of client-minutes grows, the work required of the server exceeds its capability and responses may be delayed so long that all client threads are blocked when a timestamp requires a request to be replayed. When such deadlines are missed often, the system degrades to a closed-loop experiment and the runtime can be significantly longer than one minute. During replay, file manager CPU load is measured by recording time spent in the kernel idle loop, and subtracting this from the total duration of the replay.

To measure file server load as a function of increasing client demand, we varied the number of client-minutes replayed simultaneously. To replay *m* client-minutes, we randomly select *m* client-minutes from the pool of busy client-minutes described in Section 4.4. As indicated by the long tails of the distributions in Figure 4, different randomly selected sets of *m* client-minutes may have

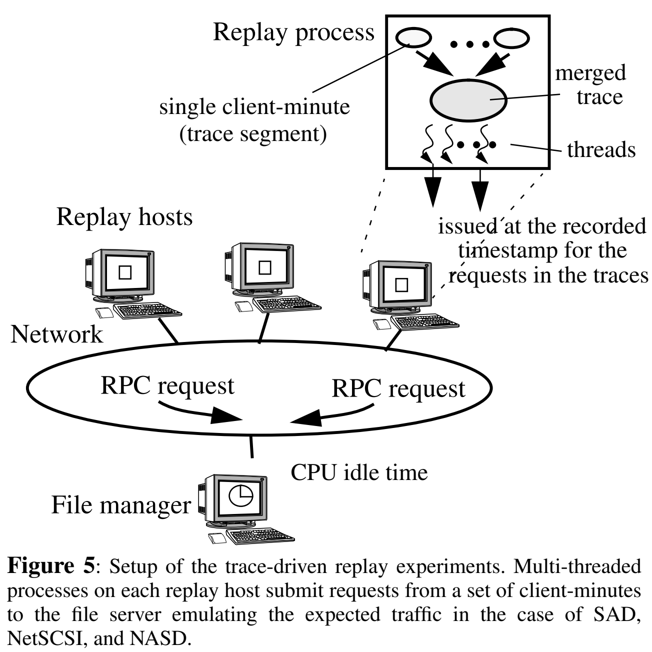

Figure 5: Setup of the trace-driven replay experiments. Multi-threaded processes on each replay host submit requests from a set of client-minutes to the file server emulating the expected traffic in the case of SAD, NetSCSI, and NASD.

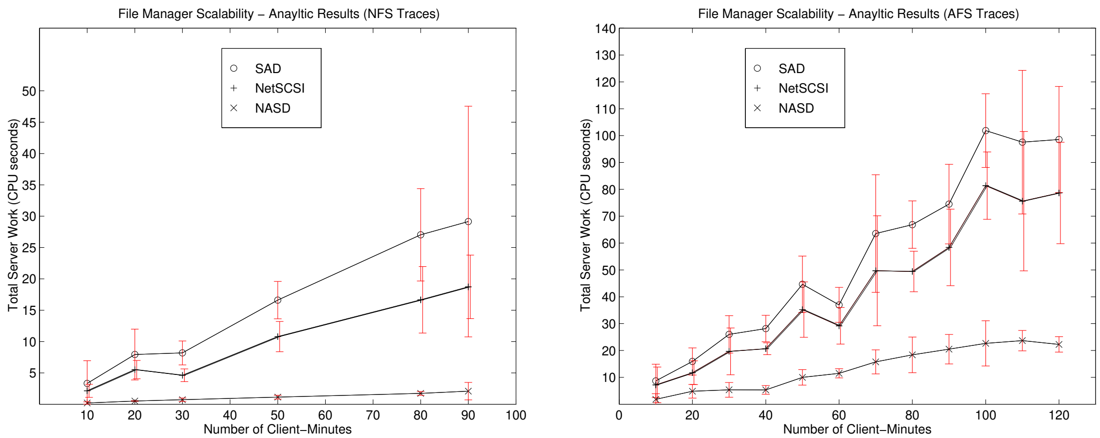

Figure 6: Mean and 90% confidence intervals according to the analytic model described in Section 5 applied to the five randomly selected samples of *m* client-minutes constructed for the replay experiment.

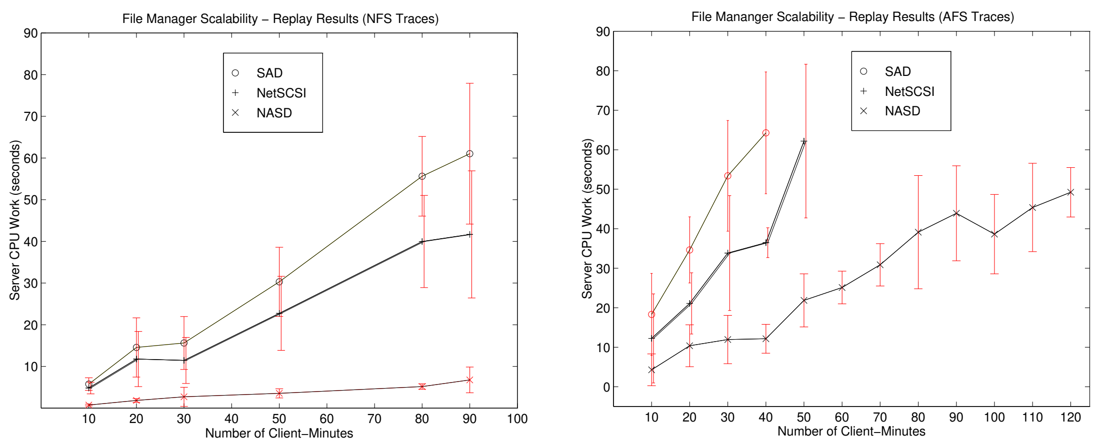

Figure 7. Mean and 90% confidence intervals of measured load for NFS and AFS replay experiments for five samples of *m* client-minutes.

The file manager was a DEC 3000/500 (150 MHz, 128 MB of memory, Digital UNIX 3.2g) with five fast wide differential SCSI busses, each with four HP C2247 disks. To balance I/O, we stripe data across the twenty disks using a 64KB stripe unit. The replay hosts were ten DEC AlphaStation 255, 3000/400, and 3000/600 workstations interconnected by a switched OC-3 ATM network (NFS replay used up to eight additional machines connected via Ethernet).

Prior to replaying a set of client-minutes, we build a filesystem which allows the clients to access files touched in those client-minutes (the original file system hierarchy was not collected with the traces). We create the correct number of files, sizing each file heuristically as the largest offset accessed in the trace. In NFS replay, each replay host was responsible for *c* (=5) client-minutes. Therefore, the number of replay hosts, *h,* varied with the number of client-minutes replayed. In AFS replay, the client-minutes were always evenly divided amongst *h* (=10) replay hosts.

widely varying load. Therefore, we construct *p* (=5) samples for each set of *m* client-minutes, and report the mean and 90% confidence intervals. For comparison, Figure 6 reports the mean and 90% confidence intervals in estimated file manager work according to the analytic model of Section 5 applied to the client-minutes selected for replay.

6.2 Results

Comparing Figure 6 and Figure 7, we see that replay experiences significantly more CPU work — work that was overlooked by the analytic model. At 90 client-minutes, the analytic NFS/NASD model predicts less than 30 CPU seconds while the replay model consumes over 60 CPU seconds. In spite of these differences, the replay results and the analytic data display a strong correlation in the relative performance of SAD, NetSCSI and NASD. For example, at 90 NFS client-minutes, both show a 40% difference in load between SAD and NetSCSI, and a 90% difference between SAD and NASD, with similar correspondence in the AFS case. The similarities between the results of Section 5 and 6 suggest that, provided implementations on NetSCSI and NASD have operation costs similar to those in Table 6, NetSCSI provides limited benefit to existing distributed file systems (a factor of about 1.5 improvement). In contrast, NASD promises substantially lower file manager costs per client (factors of up to ten for NFS and up to five for AFS).

From Figure 7, it appears that each data point required less than 60 seconds of file manager CPU, the amount available in these one minute replay experiments. However, when CPU saturation of the manager slows the generation of trace events, it causes the replay to run longer than 60 seconds. For NFS, replay overrun occurs with 80 or more client-minutes in SAD and NetSCSI and does not occur in NASD. For AFS, this overrun occurs with 25 or more client-minutes in SAD, 50 or more client-minutes in NetSCSI and 120 or more client-minutes in NASD. AFS suffers more from this effect because its file manager is user-level with only one kernel thread — the entire file manager blocks on every disk access (NFS is in-kernel and the file manager has sixteen threads at its disposal).

6.3 Cache Effects

A limitation of the results in Figure 7 relates to our handling of file manager cache state. The use of samples constructed from random, busy one-minute intervals makes it difficult to determine what constitutes a realistic initial state for the data and metadata caches. Because a cache miss induces more file manager work than a cache hit, biases which increase misses also increase work. Further, because the file manager work in SAD, NetSCSI and NASD differs, with far fewer cache accesses done by NetSCSI and NASD, it is reasonable to expect that SAD file manager work is over-estimated more by excess misses than the other cases.

For this reason we should have run all workloads with warm caches, biasing in favor of SAD file managers. Unfortunately, our ability to control cache contents carefully was best when using cold caches.[4] Therefore, to bound the bias against SAD, we ran a simple experiment. For NFS's 10 and 20 client-minute workloads, the entire set of data accessed during those client-minutes fits in the file manager's buffer cache. This allowed us to perform the NFS replay with an initially cold data cache, then repeat the same replay without flushing the contents of the cache (thus starting from an optimally-warmed cache, containing all the data which will be accessed in the second run). In this case, the CPU load on the file manager in SAD decreased by 10-18%, which we take as the upper bound on the effect of warm versus cold data caches on the CPU load of a SAD file manager. To restate, Figure 7 may falsely penalize SAD performance by up to 18% because of cold data caches during replay.

7 Conclusion and Future Directions

Network-attached storage, by enabling direct transfers between client and storage, can substantially increase distributed file system scalability while simultaneously enabling striped storage to satisfy the bursty, high-bandwidth demands of the increasingly high-performance clients populating local area networks. This promises benefits in a wide enough range of storage markets and makes commodity network-attached storage feasible.

In this paper we have presented a simple classification of storage architectures for distributed file systems with four models. The traditional, server-attached disk (SAD) model is our base case. Server-integrated (SID) disk systems, including specialized NFS server products, are architecturally identical, but have hardware and software designed specifically for file service. We do not emphasize this model because it binds storage products to a particular choice of distributed file system.

The remaining two storage models exploit the potential of network-attached storage. Network SCSI (NetSCSI) drives are very similar to current SCSI disks in that all file requests go through the file manager, but the resulting data transfers go directly between client and drive. This may reduce file manager workload during busy periods by about 30%. Different security models can be provided using NetSCSI depending on the cryptographic support provided in the drive.

[4] Even here our control was incomplete; NFS used a cold data cache, but a warm metadata cache. AFS uses both cold data and metadata caches.

Network-attached secure disks (NASD) support storage semantics at a level between that of block-level protocols like SCSI and distributed file systems like NFS and AFS. The partitioning of file system functionality between NASD drive and file manager is optimized to reduce file manager load while maintaining system flexibility. To operate securely in the face of this partition, NASD drives rely on cryptographic support for security and authorization. Our studies show that, by offloading data read and write and attribute and directory read operations, distributed file system server load during busy periods may be reduced by a factor of fourteen (NFS) and three (AFS) in the analytic model, and up to ten (NFS) and five (AFS) in the replay experiments.

Our analysis focuses on describing the distinct methods of organizing storage architecture and estimating the potential improvement each promises for existing distributed file systems. With the positive results given here, our future directions are clear. We plan to demonstrate that distributed file systems can be implemented around network-attached storage, preserving powerful security models and yielding considerable scalability and client performance advantages. Along this path, many open questions remain. Our NASD model, in particular, expects a disk drive to be capable of computation not normally associated with cost-sensitive commodity peripherals; drive micro-architectures and software structures must be developed and demonstrated.

Further, NASD's out-of-datapath file manager does not naturally provide the server caching found in traditional systems which store-and-forward data through the server. We must evaluate the penalty of distributing the caches among storage. This penalty may be mitigated if storage objects are striped over drives because striping inherently eliminates hotspots [Livny87]. On the other hand, server caching is significantly less important to performance than client caching and becomes less important still with cooperative caching in idle clients [Dahlin94, Feeley95] and aggressive prefetching by clients [Patterson95, Cao95].

Finally, in the NASD models presented, we assume that clients "open" files by contacting the distributed file system server one file at a time to set up the state needed for direct transfers to and from storage and allow the file manager to handle consistency. A clear improvement, similar to the effect of client caching in AFS, might be provided by pre-authorization or group-authorization schemes.

8 Acknowledgments

We would like to thank Michael Dahlin from Berkeley for providing us their NFS traces and helping us understand some of the details. We thank Doug Tygar for his comments. We would also like to thank all of the members of the Parallel Data Lab who allowed us to trace their AFS accesses over the course of several months and provided much helpful feedback on the ideas and experiments presented here. Finally, we thank the anonymous reviewers for their helpful comments.

9 Bibliography

[Anderson91]Anderson, T.E. et al., "The Interaction of Architecture and Operating System Design," 4th ASPLOS, Sept. 1991.

[Anderson95]Anderson, D. (Seagate Technology), Personal communication, 1995.

[ANSI86]ANSI, "Small Computer System Interface (SCSI) Specification", ANSI X3.131-1986, 1986.

[ANSI95]ANSI, "SCSI-3 Fast-20 Parallel Interface", X3T10/1047D Working Group, Revision 6.

[Arnould89]Arnould, E.A. et al., "The Design of Nectar: A Network Backplane for Heterogeneous Multicomputers", 3rd ASPLOS, April 1989, pp. 205-216.

[Baker91]Baker, M.G. et al., "Measurements of a Distributed File System", 13th SOSP, Oct. 1991, pp. 198-212.

[Benner96]Benner, A.F., "Fibre Channel: Gigabit Communications and I/O for Computer Networks", McGraw Hill, New York, 1996.

[Berdahl95]Berdahl, L., Draft of "Parallel Transport Protocol Proposal", Lawrence Livermore National Labs, January 3, 1995.

[Birrell80]Birrell, A.D. and Needham, R.M., "A Universal File Server", IEEE Transactions on Software Engineering SE-6,5, Sept. 1980.

[Boden95]Boden, N.J. et al, "Myrinet: A Gigabit-per-Second Local Area Network", IEEE Micro, Feb. 1995.

[Brustoloni96] Brustoloni, G. and Steenkiste, P., "Effects of Buffering Semantics on I/O Performance," 2nd OSDI, Oct. 1996.

[Cao95]Cao, P. et al., "A Study of Integrated Prefetching and Caching Strategies," SIGMETICS 95, May 1995.

[Chen90]Chen, P.M. et al., "An evaluation of Redundant Arrays of Disks using an Amdahl 5890," SIGMETRICS 90, 1990.

[Chen93]Chen, J.B. and Bershad, B., "The Impact of Operating System Structure on Memory System Performance," 14th SOSP, Dec. 1993, pp. 120-133.

[Clark89]Clark, D.D. et al., "An Analysis of TCP Processing Overhead," IEEE Communications 27,6 (June 89), pp. 23-36.

[Cooper90]Cooper, E., et al., "Host Interface Design for ATM LANs", 16th Conference on Local Computer Networks, Oct. 1991, pp. 247-258.

[Dahlin94]Dahlin, M. et al., "Cooperative Caching: Using Remote Client Memory to Improve File System Performance," First OSDI, pp. 267-280, Nov. 1994.

[Dahlin95]Dahlin, M.D. et al., "A Quantitative Analysis of Cache Policies for Scalable Network File Systems", 15th SOSP, Dec. 1995.

[deJong93]deJonge, W., Kaashoek, M.F. and Hsieh, W.C., "The Logical Disk: A New Approach to Improving File Systems," 14th SOSP, Dec. 1993.

[Drapeau94]Drapeau, A.L. et al., "RAID-II: A High-Bandwidth Network File Server", 21st ISCA, 1994, pp. 234-244.

[Druschel93]Druschel, P. and Peterson, L.L., "Fbufs: A High-Bandwidth Cross-Domain Transfer Facility", 14th SOSP, Dec. 1993, pp. 189-202.

[Feeley95]Feeley, M. J. et al., "Implementing global memory management in a workstation cluster," 15th SOSP, Dec. 1995.

[Gibson92]Gibson, G., "Redundant Disk Arrays: Reliable, Parallel Secondary Storage," MIT Press, 1992.

[Gobioff96]Gobioff, H. et al., "Security for Network-Attached Storage Devices," CMU-CS-96-179, 1996.

[Golding95]Golding, R., et al., "Attribute-managed storage," Workshop on Modeling and Specification of I/O, San Antonio, TX, Oct. 1995.

[Grochowski96]Grochowski, E.G., Hoyt, R.F., "Future Trends in Hard Disk Drives," IEEE Transactions on Magnetics 32, 3 (May 1996), pp. 1850-1854.

[Hartman93]Hartman, J.H. and Ousterhout, J.K., "The Zebra Striped Network File System", 14th SOSP, Dec. 1993.

[Hitz90]Hitz, D. et al., "Using UNIX as One Component of a Lightweight Distributed Kernel for Multiprocessor File Servers", Winter 1990 USENIX, pp. 285-295.

[Hitz94]Hitz, D., Lau, J. and Malcolm, M. "File Systems Design for an NFS File Server Appliance", Winter 1994 USENIX, Jan. 1994.

[Horst95]Horst, R.W., "TNet: A Reliable System Area Network", IEEE Micro, Feb. 1995.

[Howard88]Howard, J.H. et al., "Scale and Performance in a Distributed File System", ACM TOCS 6, 1, Feb. 1988, pp. 51-81.

[IEEE92]IEEE, "Scalable Coherent Interconnect", Standard 1596-1992, 1992.

[IEEE94]IEEE P1244. "Reference Model for Open Storage Systems Interconnection-Mass Storage System Reference Model Version 5", Sept. 1995.

[Katz92]Katz, R.H., "High-Performance Network- and Channel-Attached Storage", Proceedings of the IEEE 80,8, Aug. 1992.

[Kim86]Kim, M.Y., "Synchronized disk interleaving", IEEE Transactions on Computers C-35, 11, Nov. 1986.

[Kronenberg86]Kronenberg, N.P. et al., "VAXClusters: A closely-coupled distributed system", ACM TOCS 4,2, May 1986, pp. 130-146.

[Lee95]Lee, E.K., "Highly-Available, Scalable Network Storage", Spring 1995 COMPCON, Mar. 1995.

[Livny87]Livny, M., "Multi-disk management algorithms", SIGMETRICS 87, May 1987.

[Long94]Long, D.D.E., Montague, B.R., and Cabrera, L., "Swift/RAID: A Distributed RAID System," Computing Systems 7,3, Summer 1994.

[Ma96]Ma, Q., Steenkiste, P., and Zhang, H., "Routing High-Bandwidth Traffic in Max-Min Fair Share Networks," SIGCOMM 96, Aug. 1996.

[Maeda93]Maeda, C., and Bershad, B., "Protocol Service Decomposition for High-Performance Networking", 14th SOSP, Dec. 1993, pp. 244-255.

[Massiglia94]Massiglia, P., ed., "The RAIDbook", RAID Advisory Board, 1994.

[Menascé96]Menascé, D.A., et al., "An Analytic Model of Hierarchical Mass Storage Systems with Network-Attached Storage Devices," SIGMETRICS 96, May 1996.

[Miller88]Miller, S.W., "A Reference Model for Mass Storage Systems", Advances in Computers 27, 1988, pp. 157-210.

[Minshall94]Minshall, G., Major, D., and Powell, K., "An Overview of the NetWare Operating System", Winter 1994 USENIX, 1994.

[National96]National Semiconductor. "The PersonaCard 100 Data Security Card," http://www.ipsecure.com/htm/persona.html.

[Nelson88]Nelson, M.N., Welch, B.B. and Ousterhout, J.K., "Caching in the Sprite Network File System", ACM TOCS 6,1, Feb. 1988, pp. 134-154.

[NIST94]National Institute of Standards and Technology, "Digital Signature Standard." NIST FIPS Pub 186.

[NIST94a]National Institute of Standards and Technology, "Security Requirements for Cryptographic Modules", NIST FIPS 140-1.

[Ousterhout85]Ousterhout, J.K. et al., "A Trace-Driven Analysis of the UNIX 4.2 BSD File System", 10th SOSP, Dec. 1985.

[Ousterhout91]Ousterhout, J.K., "Why Aren't Operating Systems Getting Faster As Fast As Hardware?", Summer 1991 USENIX, June 1991, pp. 247-256.

[Patterson88]Patterson, D.A., Gibson, G. and Katz, R.H., "A Case for Redundant Arrays of Inexpensive Disks (RAID)", SIGMOD 88, June 1988, pp. 109-116.

[Patterson95]Patterson, R.H. et al., "Informed Prefetching and Caching", 15th SOSP, 1995.

[Rambus92]Rambus Inc., "Rambus Architectural Overview", 1992. http://www.rambus.com.

[Riedel96]Riedel, E. and Gibson, G. "Understanding Customer Dissatisfaction With Underutilized Distributed File Servers", 5th Goddard Conference on Mass Storage Systems and Technologies, College Park, MD, Sept. 1996.

[Ruemmler91]Ruemmler, C. and Wilkes, J., "Disk Shuffling", Hewlett-Packard Laboratories Concurrent Systems Project Tech Report HPL-CSP-91-30.

[Sachs94]Sachs, M.W., Leff, A., and Sevigny, D., "LAN and I/O Convergence: A Survey of the Issues", IEEE Computer, Dec. 1994. pp. 24-32.

[Sandberg85]Sandberg, R. et al., "Design and Implementation of the Sun Network Filesystem", Summer 1985 USENIX, June 1985, pp. 119-130.

[Seagate96]Seagate Corporation, "Barracuda Family Product Brief (ST19171)", 1996.

[Siu95]Siu, K.-Y. and Jain, R. "A brief overview of ATM: Protocol layers, LAN emulation and traffic management", ACM SIGCOMM, Vol 25,2, Dec. 1994, pp. 69-79.

[Srivastava94]Srivastava, A., and Eustace, A., "ATOM: A system for building customized program analysis tools", WRL Technical Report TN-41, 1994.

[StorageTek94]Storage Technology Corporation, "Iceberg 9200 Storage System: Introduction", STK Part Number 307406101, Storage Technology Corporation, 1994.

[Steenkiste94]Steenkiste, P., "A Systematic Approach to Host Interface Design for High-Speed Networks", IEEE Computer, Mar. 1994.

[Traw95]Traw, C.B.S. and Smith, J.M., "Striping Within the Network Subsystem", IEEE Network, Jul./Aug. 1995.

[Tygar95]Tygar, J.D., and Yee, B.S., "Secure Coprocessors in Electronic Commerce Applications," 1995 USENIX Electronic Commerce Workshop, 1995, New York.

[VanMeter96]Van Meter, R., "A Brief Survey Of Current Work on Network Attached Peripherals (Extended Abstract)", Operating Systems Review 30,1, Jan. 1996.

[VanMeter96a]Van Meter, R., Holtz, S., and Finn G., "Derived Virtual Devices: A Secure Distributed File System Mechanism", 5th Goddard Conference on Mass Storage Systems and Technologies", College Park, MD, Sept. 1996.

[Varma95]Varma, A. and Jacobson, Q., "Destage Algorithms for Disk Arrays with Non-volatile Caches", 22nd ISCA, 1995.

[vonEicken92]von Eicken, T. et al., "Active Messages: A Mechanism for Integrated Communication and Computation", 19th ISCA, May 1992, pp. 256-266.

[Watson95]Watson, R.W., and Coyne, R.A., "The Parallel I/O Architecture of the High-Performance Storage System (HPSS)," 14th IEEE Symposium on Mass Storage Systems, Sept. 1995, pp. 27-44.

[Weingart87]Weingart, S.H., "Physical Security of the µABYSS System", IEEE Computer Society Conference on Security and Privacy, 1987, pp. 52-58.

[White87]White, S.R. and Comerford, L., "ABYSS: A Trusted Architecture for Software Protection", IEEE Computer Society Conference on Security and Privacy, 1987, pp. 38-51.

[Wilkes95]Wilkes, J. et al., "The HP AutoRAID Hierarchical Storage System", 15th SOSP, Dec. 1995.

[Wiltzius95]Wiltzius, D. et al., "Network-attached peripherals for HPSS/SIOF", http://www.llnl.gov/liv_comp/siof/siof_nap.