Proceedings of the 8th Conference on Architectural Support for Programming Languages and Operating Systems, 1998.

A Cost-Effective, High-Bandwidth Storage Architecture

Garth A. Gibson*, David F. Nagle†, Khalil Amiri†, Jeff Butler†, Fay W. Chang*,
Howard Gobioff*, Charles Hardin†, Erik Riedel†, David Rochberg*, Jim Zelenka*

School of Computer Science*
Department of Electrical and Computer Engineering†
Carnegie Mellon University, Pittsburgh, PA 15213
garth+asplos98@cs.cmu.edu

ABSTRACT

This paper describes the Network-Attached Secure Disk (NASD) storage architecture, prototype implementations of NASD drives, array management for our architecture, and three filesystems built on our prototype. NASD provides scalable storage bandwidth without the cost of servers used primarily for transferring data from peripheral networks (e.g. SCSI) to client networks (e.g. ethernet). Increasing dataset sizes, new attachment technologies, the convergence of peripheral and interprocessor switched networks, and the increased availability of on-drive transistors motivate and enable this new architecture. NASD is based on four main principles: direct transfer to clients, secure interfaces via cryptographic support, asynchronous non-critical-path oversight, and variably-sized data objects. Measurements of our prototype system show that these services can be cost-effectively integrated into a next generation disk drive ASIC. End-to-end measurements of our prototype drive and filesystems suggest that NASD can support conventional distributed filesystems without performance degradation. More importantly, we show scalable bandwidth for NASD-specialized filesystems. Using a parallel data mining application, NASD drives deliver a linear scaling of 6.2 MB/s per client-drive pair, tested with up to eight pairs in our lab.

Keywords

D.4.3 File systems management, D.4.7 Distributed systems, B.4 Input/Output and Data Communications.

1. INTRODUCTION

Demands for storage bandwidth continue to grow due to rapidly increasing client performance, richer data types such as video, and data-intensive applications such as data mining. For storage subsystems to deliver scalable bandwidth, that is, linearly increasing application bandwidth with increasing numbers of storage devices and client processors, the data must be striped over many disks and network links [Patterson88]. With 1998 technology, most office, engineering, and data processing shops have sufficient numbers of disks and scalable switched networking, but they access storage through storage controller and distributed fileserver bottlenecks. These bottlenecks arise because a single "server" computer receives data from the storage (peripheral) network and forwards it to the client (local area) network while adding functions such as concurrency control and metadata consistency. A variety of research projects have explored techniques for scaling the number of machines used to enforce the semantics of such controllers or fileservers [Cabrera91, Hartman93, Cao93, Drapeau94, Anderson96, Lee96, Thekkath97]. As Section 3 shows, scaling the number of machines devoted to store-and-forward copying of data from storage to client networks is expensive.

This paper makes a case for a new scalable bandwidth storage architecture, Network-Attached Secure Disks (NASD), which separates management and filesystem semantics from store-and-forward copying. By evolving the interface for commodity storage devices (SCSI-4 perhaps), we eliminate the server resources required solely for data movement. As with earlier generations of SCSI, the NASD interface is simple, efficient and flexible enough to support a wide range of filesystem semantics across multiple generations of technology. To demonstrate how a NASD architecture can deliver scalable bandwidth, we describe a prototype implementation of NASD, a storage manager for NASD arrays, and a simple parallel filesystem that delivers scalable bandwidth to a parallel data-mining application. Figure 1 illustrates the components of a NASD system and indicates the sections describing each.

We continue in Section 2 with a discussion of storage system architectures and related research. Section 3 presents enabling technologies for NASD. Section 4 presents an overview of our NASD interface, its implementation and performance. Section 5 discusses ports of NFS and AFS filesystems to a NASD-based system, our implementation of a NASD array management system, a simple parallel filesystem, and an I/O-intensive data mining application that exploits the bandwidth of our prototype. This section reports the scalability of our prototype compared to the performance of a fast single NFS server. Section 6 discusses active disks, the logical extension of NASD to execute application code. Section 7 concludes with a discussion of ongoing research.

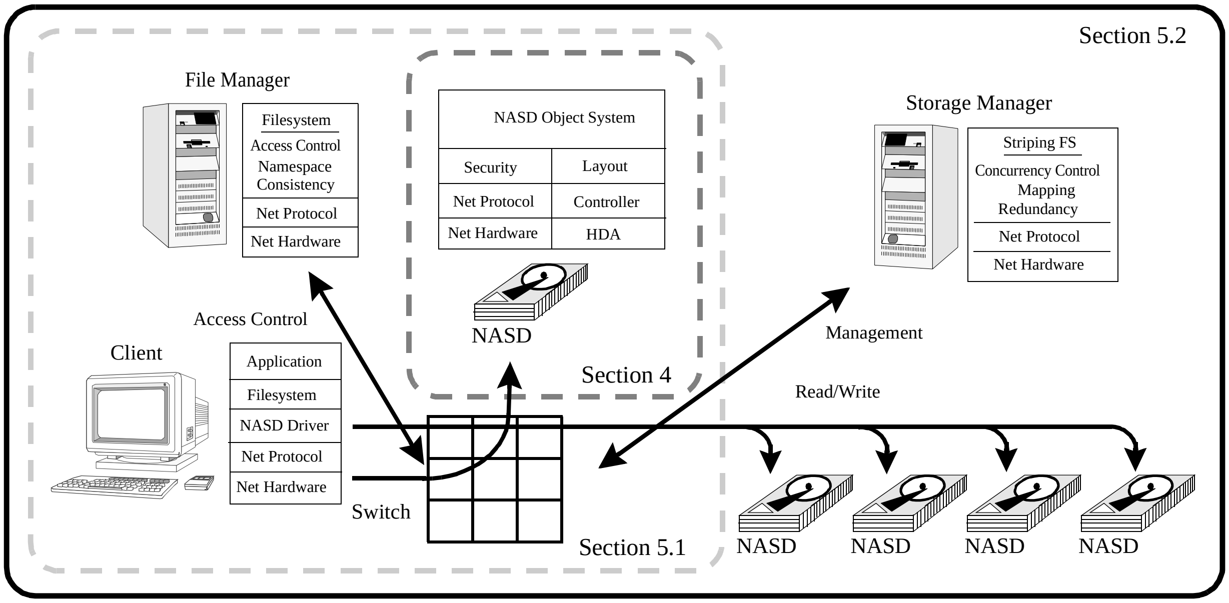


Figure 1: An overview of a scalable bandwidth NASD system. The major components are annotated with the layering of their logical components. The innermost box shows a basic NASD drive as described in Section 4. The larger box contains the essentials for a NASD-based filesystem, which adds a file manager and client as detailed in Section 5.1. Finally, the outer box adds a storage manager to coordinate drives on which parallel filesystem is built as discussed in Section 5.2.

2. BACKGROUND AND RELATED WORK

Figure 2 illustrates the principal alternative storage architectures: (1) a local filesystem, (2) a distributed filesystem (DFS) built directly on disks, (3) a distributed filesystem built on a storage subsystem, (4) a network-DMA distributed filesystem, (5) a distributed filesystem using smart object-based disks (NASD) and (6) a distributed filesystem using a second level of objects for storage management.

The simplest organization (1) aggregates the application, file management (naming, directories, access control, concurrency control) and low-level storage management. Disk data makes one trip over a simple peripheral area network such as SCSI or Fibrechannel and disks offer a fixed-size block abstraction. Stand-alone computer systems use this widely understood organization.

To share data more effectively among many computers, an intermediate server machine is introduced (2). If the server offers a simple file access interface to clients, the organization is known as a distributed filesystem. If the server processes data on behalf of the clients, this organization is a distributed database. In organization (2), data makes a second network trip to the client and the server machine can become a bottleneck, particularly since it usually serves large numbers of disks to better amortize its cost.

The limitations of using a single central fileserver are widely recognized. Companies such as Auspex and Network Appliance have attempted to improve file server performance, specifically the number of clients supported, through the use of special purpose server hardware and highly optimized software [Hitz90, Hitz94]. Although not the topic of this paper, the NASD architecture can improve the client-load-bearing capability of traditional filesystems by off-loading simple data-intensive processing to NASD drives [Gibson97a].

To transparently improve storage bandwidth and reliability many systems interpose another computer, such as a RAID controller [Patterson88]. This organization (3) adds another peripheral network transfer and store-and-forward stage for data to traverse.

Provided that the distributed filesystem is reorganized to logically “DMA” data rather than copy it through its server, a fourth organization (4) reduces the number of network transits for data to two. This organization has been examined extensively [Drapeau94, Long94] and is in use in the HPSS implementation of the Mass Storage Reference Model [Watson95, Miller88]. Organization (4) also applies to systems where clients are trusted to maintain filesystem metadata integrity and implement disk striping and redundancy [Hartman93, Anderson96]. In this case, client caching of metadata can reduce the number of network transfers for control messages and data to two. Moreover, disks can be attached to client machines which are presumed to be independently paid for and generally idle. This eliminates additional store-and-forward cost, if clients are idle, without eliminating the copy itself.

As described in Section 4, the NASD architecture (5) embeds the disk management functions into the device and offers a variable-length object storage interface. In this organization, file managers enable repeated client accesses to specific storage objects by granting a cachable capability.

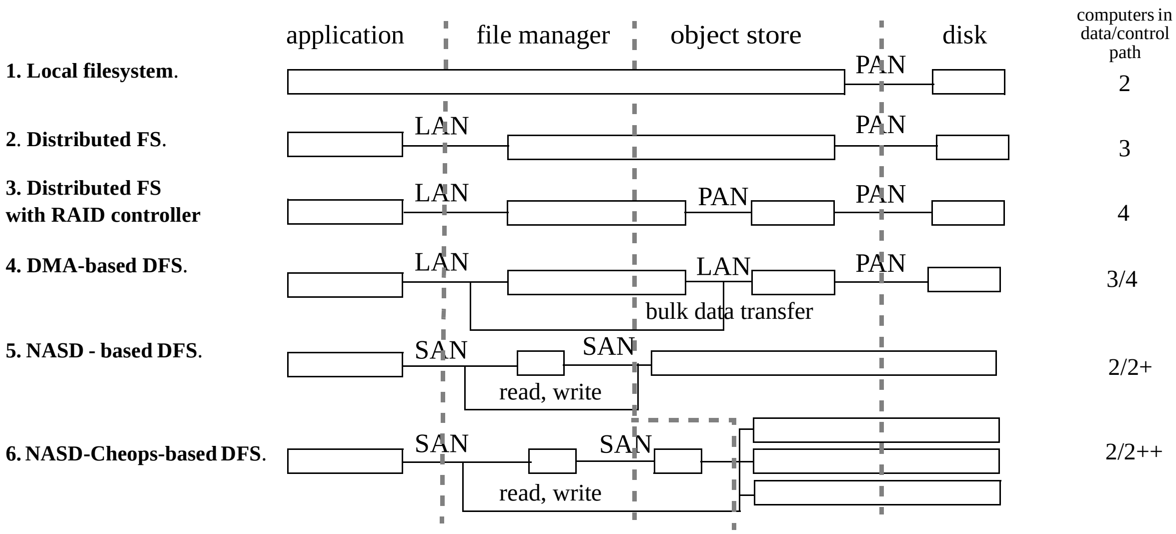


Figure 2: Evolution of storage architectures for untrusted networks and clients. Boxes are computers, horizontal lines are communication paths and vertical lines are internal and external interfaces. *LAN* is a local area network such as Ethernet or FDDI. *PAN* is a peripheral area network such as SCSI, Fibrechannel or IBM's ESCON. *SAN* is an emerging system area network such as ServerNet, Myrinet or perhaps Fibrechannel or Ethernet that is common across clients, servers and devices. On the far right, a *disk* is capable of functions such as seek, read, write, readahead, and simple caching. The *object store* binds blocks into variable-length objects and manages the layout of these objects in the storage space offered by the device(s). The *file manager* provides naming, directory hierarchies, consistency, access control, and concurrency control. In NASD, storage management is done by recursion on the object interface on the SAN.

Hence, all data and most control travels across the network once and there is no expensive store-and-forward computer.

The idea of a simple, disk-like network-attached storage server as a building block for high-level distributed filesystems has been around for a long time. Cambridge's Universal File Server (UFS) used an abstraction similar to NASD along with a directory-like index structure [Birrell80]. The UFS would reclaim any space that was not reachable from a root index. The successor project at Cambridge, CFS, also performed automatic reclamation and added undoable (for a period of time after initiation) transactions into the filesystem interface [Mitchell81]. To minimize coupling of file manager and device implementations, NASD offers less powerful semantics, with no automatic reclamation or transaction rollback.

Using an object interface in storage rather than a fixed-block interface moves data layout management to the disk. In addition, NASD partitions are variable-sized groupings of objects, not physical regions of disk media, enabling the total partition space to be managed easily, in a manner similar to virtual volumes or virtual disks [IEEE95, Lee96]. We also believe that specific implementations can exploit NASD's uninterpreted filesystem-specific attribute fields to respond to higher-level capacity planning and reservation systems such as HP's attribute-managed storage [Golding95]. Object-based storage is also being pursued for quality-of-service at the device, transparent performance optimizations, and drive supported data sharing [Anderson98a].

ISI's Netstation project [VanMeter96] proposes a form of object-based storage called Derived Virtual Devices (DVD) in which the state of an open network connection is augmented with access control policies and object metadata, provided by the file manager using Kerberos [Neuman94] for underlying security guarantees. This is similar to NASD's mechanism except that NASD's access control policies are embedded in unforgeable capabilities separate from communication state, so that their interpretation persists (as objects) when a connection is terminated. Moreover, Netstation's use of DVD as a physical partition server in VISA [VanMeter98] is not similar to our use of NASD as a single-object server in a parallel distributed filesystem.

In contrast to the ISI approach, NASD security is based on capabilities, a well-established concept for regulating access to resources [Dennis66]. In the past, many systems have used capabilities that rely on hardware support or trusted operating system kernels to protect system integrity [Wulf74, Wilkes79, Karger88]. Within NASD, we make no assumptions about the integrity of the client to properly maintain capabilities. Therefore, we utilize cryptographic techniques similar to ISCAP [Gong89] and Amoeba [Tanenbaum86]. In these systems, both the entity issuing a capability and the entity validating a capability must share a large amount of private information about all of the issued capabilities. These systems are generally implemented as single entities issuing and validating capabilities, while in NASD these functions are done in distinct machines and no per-capability state is exchanged between issuer and validator.

To offer disk striping and redundancy for NASD, we layer the NASD interface. In this organization (6), a storage manager replaces the file manager's capability with a set of capabilities for the objects that actually make up the high-level striped object. This costs an additional control message but once equipped with these capabilities, clients again access storage objects directly. Redundancy and striping are done within the objects accessible with the client's set of capabilities, not the physical disk addresses.

Our storage management system, Cheops, differs from other storage subsystems with scalable processing power such as Swift, TickerTAIP and Petal [Long94, Cao93, Lee96] in that Cheops uses client processing power rather than scaling the computational power of the storage subsystem. Cheops is similar to the Zebra and xFS filesystems except that client trust is not required because the client manipulates only objects it can access [Hartman93, Anderson96].

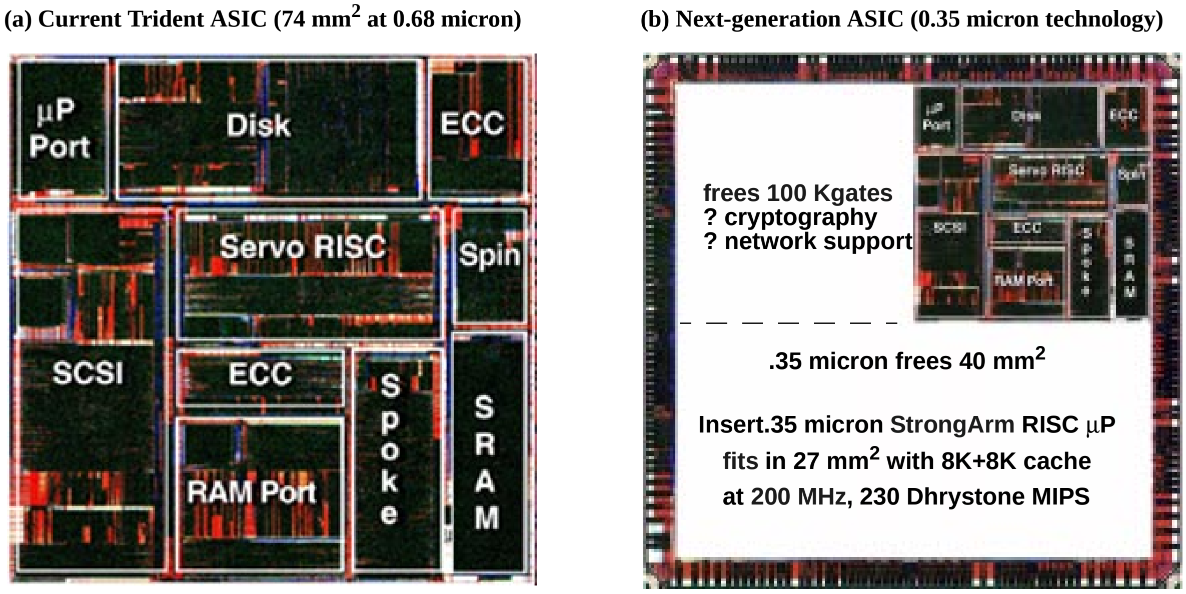
3. ENABLING TECHNOLOGY

Storage architecture is ready to change as a result of the synergy between five overriding factors: I/O bound applications, new drive attachment technologies, an excess of on-drive transistors, the convergence of peripheral and interprocessor switched networks, and the cost of storage systems.

I/O-bound applications: Traditional distributed filesystem workloads are dominated by small random accesses to small files whose sizes are growing with time, though not dramatically [Baker91, TPC98]. In contrast, new workloads are much more I/O-bound, including data types such as video and audio, and applications such as data mining of retail transactions, medical records, or telecommunication call records.

New drive attachment technology: The same technology improvements that are increasing disk density by 60% per year are also driving up disk bandwidth at 40% per year [Grochowski96]. High transfer rates have increased pressure on the physical and electrical design of drive busses, dramatically reducing maximum bus length. At the same time, people are building systems of clustered computers with shared storage. For these reasons, the storage industry is moving toward encapsulating drive communication over Fibrechannel [Benner96], a serial, switched, packet-based peripheral network that supports long cable lengths, more ports, and more bandwidth. One impact of NASD is to evolve the SCSI command set that is currently being encapsulated over Fibrechannel to take full advantage of the promises of that switched-network technology for both higher bandwidth and increased flexibility.

Excess of on-drive transistors: The increasing transistor density in inexpensive ASIC technology has allowed disk drive designers to lower cost and increase performance by integrating sophisticated special-purpose functional units into a small number of chips. Figure 3 shows the block diagram for the ASIC at the heart of Quantum's Trident drive. When drive ASIC technology advances from 0.68 micron CMOS to 0.35 micron CMOS, they could insert a 200 MHz StrongARM microcontroller, leaving 100,000 gate-equivalent space for functions such as onchip DRAM or cryptographic support. While this may seem like a major jump, Siemen's TriCore integrated microcontroller and ASIC architecture promises to deliver a 100 MHz, 3-way issue,


(a) Current Trident ASIC (74 mm^2 at 0.68 micron)

(b) Next-generation ASIC (0.35 micron technology)

Figure 3: Quantum's Trident disk drive features the ASIC on the left (a). Integrated onto this chip in four independent clock domains are 10 function units with a total of about 110,000 logic gates and a 3 KB SRAM: a disk formatter, a SCSI controller, ECC detection, ECC correction, spindle motor control, a servo signal processor and its SRAM, a servo data formatter (spoke), a DRAM controller, and a microprocessor port connected to a Motorola 68000 class processor. By advancing to the next higher ASIC density, this same die area could also accommodate a 200 MHz StrongARM microcontroller and still have space left over for DRAM or additional functional units such as cryptographic or network accelerators.

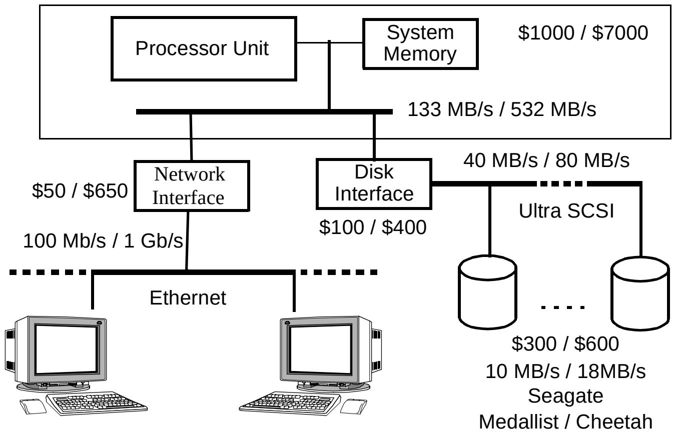


Figure 4: Cost model for the traditional server architecture. In this simple model, a machine serves a set of disks to clients using a set of disk (wide Ultra and Ultra2 SCSI) and network (Fast and Gigabit Ethernet) interfaces. Using peak bandwidths and neglecting host CPU and memory bottlenecks, we estimate the server cost overhead at maximum bandwidth as the sum of the machine cost and the costs of sufficient numbers of interfaces to transfer the disks' aggregate bandwidth divided by the total cost of the disks. While the prices are probably already out of date, the basic problem of a high server overhead is likely to remain. We report pairs of costs and bandwidth estimates. On the left, we show values for a low cost system built from high-volume components. On the right, we show values for a high-performance reliable system built from components recommended for mid-range and enterprise servers [Pricewatch98].

32-bit datapath with up to 2 MB of onchip DRAM and customer defined logic in 1998 [TriCore97].

Convergence of peripheral and interprocessor networks: Scalable computing is increasingly based on clusters of workstations. In contrast to the special-purpose, highly reliable, low-latency interconnects of massively parallel processors such as the SP2, Paragon, and Cosmic Cube, clusters typically use Internet protocols over commodity LAN routers and switches. To make clusters effective, low-latency network protocols and user-level access to network adapters have been proposed, and a new adapter card interface, the Virtual Interface Architecture, is being standardized [Maeda93, Wilkes92, Boden95, Horst95, vonEicken95, Intel97]. These developments continue to narrow the gap between the channel properties of peripheral interconnects and the network properties of client interconnects [Sachs94] and make Fibrechannel and Gigabit Ethernet look more alike than different

Cost-ineffective storage servers: In high performance distributed filesystems, there is a high cost overhead associated with the server machine that manages filesystem semantics and bridges traffic between the storage network and the client network [Anderson96]. Figure 4 illustrates this problem for bandwidth-intensive applications in terms of maximum storage bandwidth. Based on these cost and peak performance estimates, we can compare the expected overhead cost of a storage server as a fraction of the raw storage cost. Servers built from high-end components have an overhead that starts at 1,300% for one server-attached disk! Assuming dual 64-bit PCI busses that deliver every byte into and out of memory once, the high-end server saturates with 14 disks, 2 network interfaces, and 4 disk interfaces with a 115% overhead cost. The low cost server is more cost effective. One disk suffers a 380% cost overhead and, with a 32-bit PCI bus limit, a six disk system still suffers an 80% cost overhead.

While we can not accurately anticipate the marginal increase in the cost of a NASD over current disks, we estimate that the disk industry would be happy to charge 10% more. This bound would mean a reduction in server overhead costs of at least a factor of 10 and in total storage system cost (neglecting the network infrastructure) of over 50%.

4. NETWORK-ATTACHED SECURE DISKS

Network-Attached Secure Disks (NASD) enable cost-effective bandwidth scaling. NASD eliminates the server bandwidth bottleneck by modifying storage devices to transfer data directly to clients and also repartitions traditional file server or database functionality between the drive, client and server as shown in Figure 1. NASD presents a flat name space of variable-length objects that is both simple enough to be implemented efficiently yet flexible enough for a wide variety of applications. Because the highest levels of distributed filesystem functionality—global naming, access control, concurrency control, and cache coherency—vary significantly, we do not advocate that storage devices subsume the file server entirely. Instead, the residual filesystem, the *file manager*, should define and manage these high-level policies while NASD devices should implement simple storage primitives efficiently and operate as independently of the file manager as possible.

Broadly, we define NASD to be storage that exhibits the following four properties:

Direct transfer: Data is transferred between drive and client without indirection or store-and-forward through a file server machine.

Asynchronous oversight: We define asynchronous oversight as the ability of the client to perform most operations without synchronous appeal to the file manager. Frequently consulted but infrequently changed policy decisions, such as authorization decisions, should be encoded into capabilities by the file manager and subsequently enforced by drives.

Cryptographic integrity: By attaching storage to the network, we open drives to direct attack from adversaries. Thus, it is necessary to apply cryptographic techniques to defend against potential attacks. Drives ensure that commands and data have not been tampered with by generating and verifying cryptographic keyed digests. This is essentially the same requirement for security as proposed for IPv6 [Deering95].

Object-based interface: To allow drives direct knowledge of the relationships between disk blocks and to minimize security overhead, drives export variable length "objects" instead of fixed-size blocks. This also improves opportunities for storage self-management by extending into a disk an understanding of the relationships between blocks on the disk [Anderson98a].

4.1 NASD Interface

For the experiments presented in this paper, we have constructed a simple, capability-based, object-store interface (documented separately [Gibson97b]). This interface contains less than 20 requests including: read and write object data; read and write object attributes; create and remove object; create, resize, and remove partition; construct a copy-on-write object version; and set security key. Figure 5 diagrams the components of a NASD request and illustrates the layering of networking and security.

Based loosely on the inode interface of a UNIX filesystem [McKusick84], our interface provides soft partitions, control objects, and per-object attributes for preallocation, clustering, and capability revocation. Resizeable partitions allow capacity quotas to be managed by a drive administrator. Objects with well-known names and structures allow configuration and bootstrap of drives and partitions. They also enable filesystems to find a fixed starting point for an object hierarchy and a complete list of allocated object names. Object attributes provide timestamps, size, and allow capacity to be reserved and objects to be linked for clustering [deJonge93]. A logical version number on the object may be changed by a filesystem to immediately revoke a capability (either temporarily or permanently). Finally, an uninterpreted block of attribute space is available to the file manager to record its own long-term, per-object state such as filesystem access control lists or mode bits.

NASD security is based on cryptographic capabilities which are documented in an earlier publication [Gobioff97]. Capabilities are protected by a small number of keys organized into a four-level hierarchy. The primary use of the keys is to manage the key hierarchy and construct capabilities for use by clients. Clients obtain capabilities from a file manager using a secure and private protocol external to NASD. A capability contains a public and a private portion. The public portion is a description of what rights are being granted for which object. The private portion is a cryptographic key generated via a keyed message digest [Bellare96] from the public portion and drive's secret keys. A drive verifies a client's right to perform an operation by confirming that the client holds the private portion of an appropriate capability. The client sends the public portion along with each request and generates a second digest of the request parameters keyed by the private field. Because the drive knows its keys, receives the public fields of a capability with each request, and knows the current version number of the object, it can compute the client's private field (which the client cannot compute on its own because only the file manager has the appropriate drive secret keys). If any field has been changed, including the object version number, the access fails and the client is sent back to the file manager.

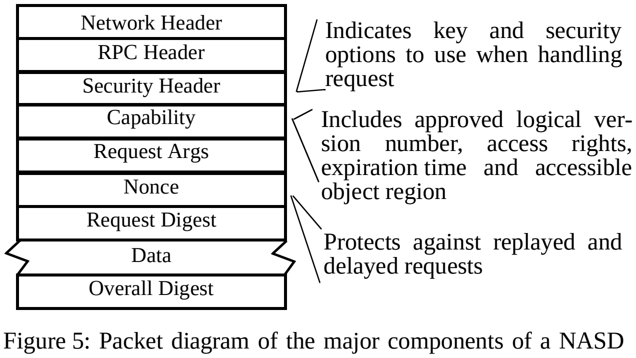

Network Header
RPC Header
Security Header
Capability
Request Args
Nonce
Request Digest
Data
Overall Digest
Indicates key and security options to use when handling request
Includes approved logical version number, access rights, expiration time and accessible object region
Protects against replayed and delayed requests

Figure 5: Packet diagram of the major components of a NASD request in our current prototype. The details of NASD objects, requests, and security are documented in separate papers [Gibson97b, Gobioff97].

These mechanisms ensure the *integrity* of requests in the presence of both attacks and simple accidents. Protecting the *integrity and/or privacy* of the data involves cryptographic operations on all the data which is potentially very expensive. Software implementations operating at disk rates are not available with the computational resources we expect on a disk, but schemes based on multiple DES function blocks in hardware can be implemented in a few tens of thousands of gates and operate faster than disk data rates [Verbauwhede87, Knudsen96]. For the measurements reported in this paper, we disabled these security protocols because our prototype does not currently support such hardware.

4.2 Prototype Implementation

We have implemented a working prototype of the NASD drive software running as a kernel module in Digital UNIX. Each NASD prototype drive runs on a DEC Alpha 3000/400 (133 MHz, 64 MB, Digital UNIX 3.2g) with two Seagate ST52160 Medallist disks attached by two 5 MB/s SCSI busses. While this is certainly a bulky "drive", the performance of this five year old machine is similar to what we predict will be available in drive controllers soon. We use two physical drives managed by a software striping driver to approximate the 10 MB/s rates we expect from more modern drives.

Because our prototype code is intended to operate directly in a drive, our NASD object system implements its own internal object access, cache, and disk space management modules (a total of 16,000 lines of code) and interacts minimally with Digital UNIX. For communications, our prototype uses DCE RPC 1.0.3 over UDP/IP. The implementation of these networking services is quite heavyweight. The appropriate protocol suite and implementation is currently an issue of active research [Anderson98b, Anderson98c, VanMeter98]

Figure 6 shows the disks' baseline sequential access bandwidth as a function of request size, labeled raw read and write. This test measures the latency of each request. Because these drives have write-behind caching enabled, a write's actual completion time is not measured accurately, resulting in a write throughput (~7 MB/s) that appears to exceed the read throughput (~5 MB/s). To evaluate object access performance, we modified the prototype to serve NASD requests from a user-level process on the same machine (without the use of RPC) and compared that to the performance of the local filesystem (a variant of Berkeley's FFS [McKusick84]). Figure 6 shows apparent throughput as a function of request size with NASD and FFS being roughly comparable. Aside from FFS's strange write-behind implementation, the principle differences are that NASD is better tuned for disk access (~5 MB/s versus ~2.5 MB/s on reads that miss in the cache), while FFS is better tuned for cache accesses (fewer copies give it ~48 MB/s versus ~40 MB/s on reads that hit in the memory cache).

4.3 Scalability

Figure 7 demonstrates the bandwidth scalability of our NASD prototype satisfying requests from cache. In this experiment there are 13 NASD drives, each linked by OC-3 ATM to 10 client machines, each a DEC AlphaStation 255 (233 MHz, 128 MB, Digital UNIX 3.2g). Each client issues a series of sequential 2 MB read requests striped across four NASDs. From Figure 6, we know that each NASD can deliver 32 MB/s from its cache to the RPC protocol stack. However, DCE RPC cannot push more than 80 Mb/s through a 155 Mb/s ATM link before the receiving client saturates. While commodity NASD drives must have a less costly RPC mechanism, this test does show a simple access pattern for which a NASD array can deliver scalable aggregate bandwidth.

4.4 Computational Requirements

Using our prototype drive software as a baseline, we can estimate the computational power needed in a drive microcontroller to support the basic NASD functions. We used the ATOM code annotation tool [Srivastava94] and the Alpha on-chip counters to measure the code paths of read and write operations. These measurements are reported in the Total Instructions columns of Table 1. For the one byte requests, our measurements with DCPI [Anderson97] also show that the prototype consumes 2.2 cycles per instruction (CPI). There are many reasons why using these numbers to predict drive performance is approximate. Our prototype uses an Alpha processor (which has different CPI properties than an embedded processor), our estimates neglect poorer CPI during copying (which would have hardware assist in a real drive), and our communications implementation is more expensive than we believe to be appropriate in a drive protocol stack. However, these numbers are still useful for broadly addressing the question of implementing NASD in a drive ASIC.

Table 1 shows that a 200 MHz version of our prototype

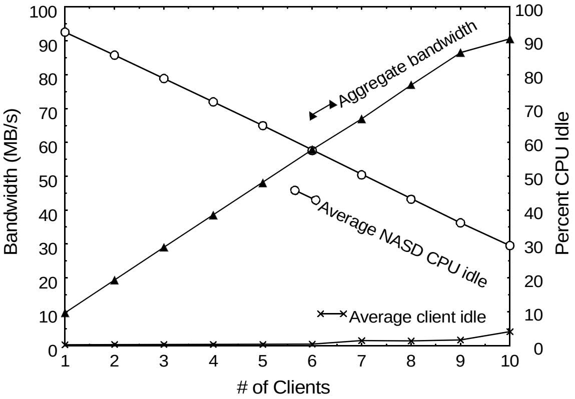


Figure 7: Prototype NASD cache read bandwidth. Read bandwidth obtained by clients accessing a single large cached file striped over 13 NASD drives with a stripe unit of 512 KB. As shown by the client idle values, the limiting factor is the CPU power of the clients within this range.

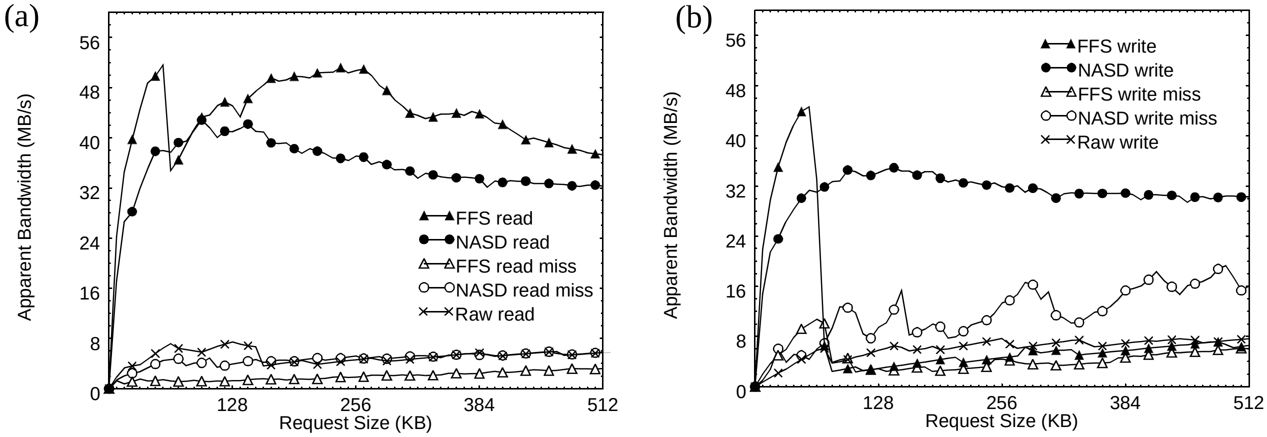


Figure 6: NASD prototype bandwidth comparing NASD, the local filesystem (FFS) and the raw device during sequential reads (a) and writes (b). The raw device stripes data in 32 KB units over two disks each on a separate 5 MB/s SCSI bus. Response timing is done by a user-level process issuing a single request for the specified amount of data. Raw disk readahead is effective for requests smaller than about 128 KB. In the "miss" cases, not even metadata is cached. For cached accesses, FFS benefits from doing one less data copy than does the NASD code. Both exhibit degradation as the processor's L2 cache (512 KB) overflows, though NASD's extra copy makes this more severe. The strange write performance of FFS occurs because it acknowledges immediately for writes of up to 64 KB (write-behind), and otherwise waits for disk media to be updated. In this test, NASD has write-behind (fully) enabled as do the disks.

Operation	Total Instructions / % Communications								Operation time (msec) (@ 200 MHz, CPI = 2.2)			
Request Size	1 B		8 KB		64 KB		512 KB		1 B	8 KB	64 KB	512KB
read - cold cache	46k	70	67k	79	247k	90	1,488k	92	0.51	0.74	2.7	16.4
read - warm cache	38k	92	57k	94	224k	97	1,410k	97	0.42	0.63	2.5	15.6
write - cold cache	43k	73	71k	82	269k	92	1,947k	96	0.47	0.78	3.0	21.3
write - warm cache	37k	92	57k	94	253k	97	1,871k	97	0.41	0.64	2.8	20.4

Table 1: Measured cost and estimated performance of read and write requests. The instruction counts and distribution were obtained by instrumenting our prototype with ATOM and using the Alpha on-chip counters. The values shown are the total number of instructions required to service a particular request size and include all communications (DCE RPC, UDP/IP) and NASD code including kernel work done on their behalf. The measured number of cycles per instruction (CPI) for 1-byte requests was 2.2 (for larger requests our processor copying implementation suffers significantly while real disks have substantial hardware assistance as shown in Figure 3). The second set of columns use these instruction counts to estimate the duration of each operation on a 200 MHz processor, assuming a CPI of 2.2 for all instructions (including copying and communications instructions). For comparison purposes, we experimented with a Seagate Barracuda (ST 34371W). This drive is able to read the next sequential sector from its cache in 0.30 msec and read a random single sector from the media in 9.4 msec. With 64 KB requests, it reads from cache in 2.2 msec and from the media, at a random location, in 11.1 msec.
“Write - warm cache” means that the needed metadata is in the cache before the operation starts.

should take 0.4-0.5 msecs for a small request, 70-90% of which is spent in the communications codepath. For 64 KB requests, we estimate 2.5-3.0 msec would be used with 90-97% of the work in communications. For comparison, we examined a Seagate Barracuda drive executing sequential reads. Because this is the most important operation for current drives, a large fraction of each operation is directly handled in hardware. For single sector reads the Barracuda takes only 0.3 msecs and for 64 KB reads it takes only 2.2 msecs.

We conclude that NASD control is not necessarily too expensive but that workstation-class implementations of communications certainly are [VanMeter98].

5. FILESYSTEMS FOR NASD

As shown in the previous section, NASD drives attached to clients by a high-bandwidth switched network are capable of scalable aggregate bandwidth. However, most client applications do not access data directly – they access data through higher-level distributed filesystems. To demonstrate the feasibility of NASD, we have ported two popular distributed filesystems, NFS and AFS [Sandberg85, Howard88], to our NASD environment. We have also implemented a minimal distributed filesystem structure that passes the scalable bandwidth of network-attached storage on to applications.

Scalability in file managers has traditionally meant increasing the number of clients supported as the total amount of storage is increased. This topic is outside the scope of this paper and has been addressed elsewhere [Howard88, Anderson96, Gibson97a, Thekkath97]. To scale achievable bandwidth with increasing storage capacity, however, requires more than simply attaching storage to the network. For example, even if the application issues sufficiently large requests, NFS and AFS break these requests into small transfer units and limit the number of requests that are concurrently issued to storage.

Pragmatically, a new class of disk devices that requires high sales volumes for cost effectiveness and new filesystems for effective performance will likely fail unless there is a strategy for evolving existing systems to the new architecture. If NASD drives can be used in traditional distributed filesystems without penalty, then customers must be moved to new NASD-optimized filesystems only when bandwidth is the primary concern.

5.1 NFS and AFS in a NASD environment

In a NASD-adapted filesystem, files and directories are stored in NASD objects. The mapping of files and directories to objects depends upon the filesystem. For our NFS and AFS ports, we use a simple approach: each file and each directory occupies exactly one NASD object, and offsets in files are the same as offsets in objects. This allows common file attributes (e.g. file length and last modify time) to correspond directly to NASD-maintained object attributes. The remainder of the file attributes (e.g. owner and mode bits) are stored in the object’s uninterpreted attributes. Because the filesystem makes policy decisions based on these file attributes, the client may not directly modify object metadata; commands that may impact policy decisions such as quota or access rights must go through the file manager.

The combination of a stateless server, weak cache consistency, and few filesystem management mechanisms make porting NFS to a NASD environment straightforward. Data-moving operations (`read`, `write`) and attribute reads (`getattr`) are directed to the NASD drive while all other requests are handled by the file manager. Capabilities are piggybacked on the file manager’s response to `lookup` operations. File attributes are either computed from NASD object attributes (e.g. modify times and object size) or stored in the uninterpreted filesystem-specific attribute (e.g.

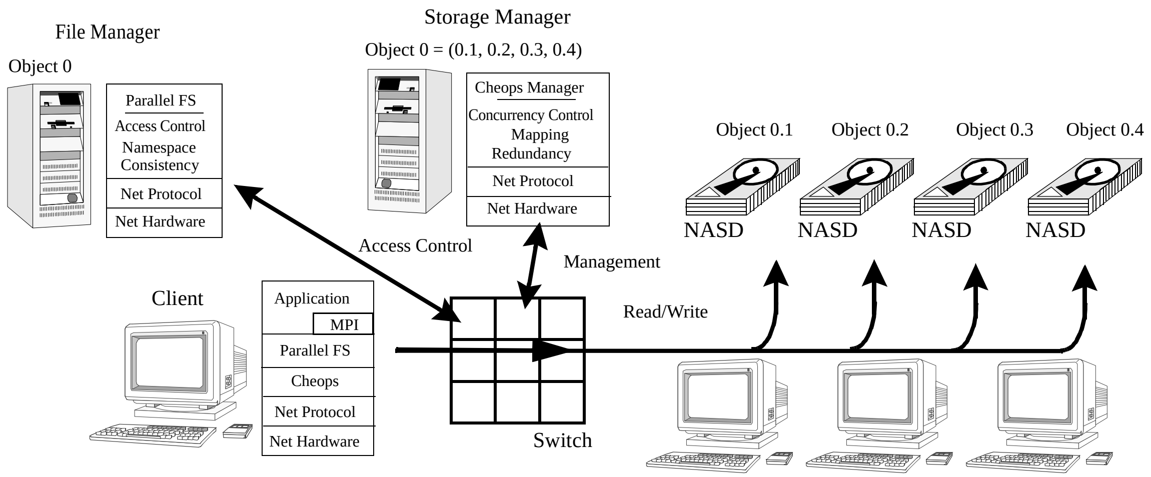


Figure 8: A NASD-optimized parallel filesystem. NASD PFS is used in conjunction with MPI for parallel applications in a cluster of workstations. The filesystem manages objects which are not directly backed by data. Instead, they are backed by a storage manager, Cheops, which redirects clients to the underlying component NASD objects. Our parallel filesystem extends a simple Unix filesystem interface with the SIO low-level interface [Corbett96] and inherits a name service, directory hierarchy, and access controls from the filesystem.

mode and uid/gid).

AFS is a more complex distributed filesystem personality, but is also readily mapped to a NASD environment. As with NFS, data-moving requests (`FetchData`, `StoreData`) and attribute reads (`FetchStatus`, `BulkStatus`) are directed to NASD drives, while all other requests are sent to the file manager. Because AFS clients perform lookup operations by parsing directory files locally, there was no obvious operation on which to piggyback the issuing of capabilities so AFS RPCs were added to obtain and relinquish capabilities explicitly. AFS's sequential conistency is provided by breaking callbacks (notifying holders of potentially stale copies) when a write capability is issued. With NASD, the file manager no longer knows that a write operation arrived at a drive so must inform clients as soon as a write may occur. The issuing of new callbacks on a file with an outstanding write capability are blocked. Expiration times set by the file manager in every capability and the ability to directly invalidate capabilities allows file managers to bound the waiting time for a callback.

AFS also requires enforcement of a per-volume quota on allocated disk space. This is more difficult in NASD because quotas are logically managed by the file manager on each write but the file manager is not accessed on each write. However, because NASD has a byte range restriction in its capabilities, the file manager can create a write capability that escrows space for the file to grow by selecting a byte range larger than the current object. After the capability has been relinquished to the file manager (or has expired), the file manager can examine the object to determine its new size and update the quota data structures appropriately.

Both the NFS and AFS ports were straightforward. Specifically, transfers remain quite small, directory parsing in NFS is still done by the server, and the AFS server still has a concurrency limitations caused by its coroutine-based user-level threads package. Our primary goal was to demonstrate that simple modifications to existing filesystems allow NASD devices to be used without performance loss. Using the Andrew benchmark [Howard88] as a basis for comparison, we found that NASD-NFS and NFS had benchmark times within 5% of each other for configurations with 1 drive/1 client and 8 drives/8 clients [Gibson97b]. We do not report AFS numbers because the AFS server's severe concurrency limitations would make a comparison unfair.

5.2 A Parallel Filesystem for NASD Clusters

To fully exploit the potential bandwidth in a NASD system, higher-level filesystems should make large, parallel requests to files striped across multiple NASD drives. As illustrated in Figure 8, our layered approach allows the filesystem to manage a "logical" object store provided by our storage management system called *Cheops*. Cheops exports the same object interface as the underlying NASD devices, and maintains the mapping of these higher-level objects to the objects on the individual devices. Our prototype system implements a Cheops client library that translates application requests and manages both levels of capabilities across multiple NASD drives. A separate Cheops *storage manager* (possibly co-located with the file manager) manages mappings for striped objects and supports concurrency control for multi-disk accesses. The Cheops client and manager is less than 10,000 lines of code.

To provide support for parallel applications, we implemented a simple parallel filesystem, NASD PFS, which offers the SIO low-level parallel filesystem interface [Corbett96] and employs Cheops as its storage management layer. We used MPICH for communications within our parallel applications [MPI95], while Cheops uses the DCE RPC mechanism required by our NASD prototype.

To evaluate the performance of Cheops, we used a parallel data mining system that discovers association rules in sales

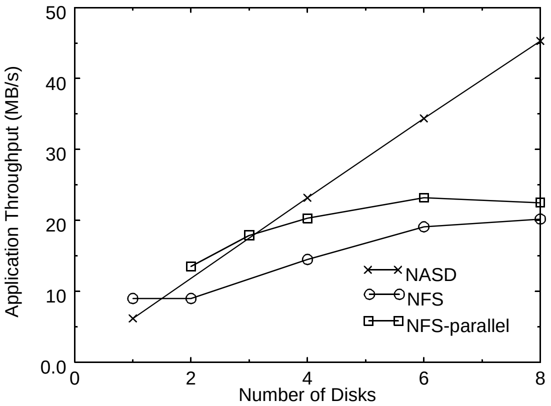


Figure 9: Scaling of a parallel data mining application. The aggregate bandwidth computing frequent sets from 300 MB of sales transactions is shown. The *NASD* line shows the bandwidth of *n* clients reading from a single NASD PFS file striped across *n* drives and scales linearly to 45 MB/s. All NFS configurations show the maximum achievable bandwidth with the given number of disks, each twice as fast as a NASD, and up to 10 clients spread over two OC-3 ATM links. The comparable *NFS* line shows the performance all the clients reading from a single file striped across *n* disks on the server and bottlenecks near 20 MB/s. This configuration causes poor read-ahead performance inside the NFS server, so we add the *NFS-parallel* line where each client reads from a replica of the file on an independent disk through the one server. This configuration performs better than the single file case, but only raises the maximum bandwidth from NFS to 22.5 MB/s.

transactions [Agrawal94]. The application's goal is to discover rules of the form "*if a customer purchases milk and eggs, then they are also likely to purchase bread*" to be used for store layout or inventory decisions. It does this in several full scans over the data, first determining the items that occur most often in the transactions (the *1-itemsets*), then using this information to generate pairs of items that occur most often (*2-itemsets*) and then larger groupings (*k-itemsets*) in subsequent passes. Our parallel implementation avoids splitting records over 2 MB boundaries and uses a simple round-robin scheme to assign 2 MB chunks to clients. Each client is implemented as four producer threads and a single consumer. Producer threads read data in 512 KB requests (which is the stripe unit for Cheops objects in this configuration) and the consumer thread performs the frequent sets computation, maintaining a set of itemset counts that are combined at a single master client. This threading maximizes overlapping and storage utilization.

Figure 9 shows the bandwidth scalability of the most I/O bound of the phases (the generation of *1-itemsets*) processing a 300 MB sales transaction file. A single NASD provides 6.2 MB/s per drive and our array scales linearly up to 45 MB/s with 8 NASD drives.

In comparison, we also show the bandwidth achieved when NASD PFS fetches from a single higher-performance traditional NFS file instead of a Cheops NASD object. The NFS file server is an AlphaStation 500/500 (500 MHz, 256 MB, Digital UNIX 4.0b) with two OC-3 ATM links (half the clients communicate over each link), and eight Seagate ST34501W Cheetah disks (13.5 MB/s) attached over two 40 MB/s Wide UltraSCSI busses. Using optimal code, this machine can internally read as much as 54.1 MB/s from these disks through the raw disk interface. We show two application throughput lines for this server. The line marked *NFS-parallel* shows the performance of each client reading from an individual file on an independent disk and achieves performance up to 22.5 MB/s. The results show an NFS server (with 35+ MB/s of network bandwidth, 54 MB/s of disk bandwidth and a perfect sequential access pattern on each disk) loses much of its potential performance to CPU and interface limits. In comparison, each NASD is able to achieve 6.2 MB/s of the raw 7.5 MB/s available from its underlying dual Medallists. Finally, the *NFS* line is the one most comparable to the *NASD* line and shows the bandwidth when all clients read from a single NFS file striped across *n* disks. This configuration is slower at 20.2 MB/s than *NFS-parallel* because its prefetching heuristics fail in the presence of multiple request streams to a single file.

In summary, NASD PFS on Cheops delivers nearly all of the bandwidth of the NASD drives, while the same application using a powerful NFS server fails to deliver half the performance of the underlying Cheetah drives.

6. ACTIVE DISKS

Recent work in our group has focused on the logical extension to exploiting the growing amount of on-drive computation by providing full application-level programmability of the drives in what we call *Active Disks* [Riedel98, Acharaya98]. This next generation of storage devices provides an execution environment directly at individual drives and allows code to execute near the data and before it is placed on the interconnect network. This provides the capability to customize functionality for specific data-intensive applications. By extending the object notion of the basic NASD interface to include code that provides specialized "methods" for accessing and operating on a particular data type, there is a natural way to tie computation to the data and scale as capacity is added to the system. NASD enables this type of extension functionality for the first time because the object-based interface provides sufficient knowledge of the data at the individual devices without having to resort to external metadata.

We have explored data mining and multimedia applications for use in Active Disks. One of the applications we examined is the frequent sets computation discussed above. In our Active Disk experiments, we also distribute the sales transaction data across a set of drives, but instead of reading the data across the network into a set of clients to do the itemset counting, the core frequent sets counting code is executed directly inside the individual drives. This allows us to take advantage of the excess computational power available at the drives and completely eliminates the need for the client nodes (for this particular application). Using the same prototype drives discussed above and approximate Active Disks functionality, we achieve 45 MB/s with low-bandwidth 10 Mb/s ethernet networking and only 1/3 of the hardware used in the NASD PFS tests of Figure 9. While the exploration of Active Disks has just begun, the potential value for some applications is dramatic.

7. CONCLUSIONS

Scalable storage bandwidth in clusters can be achieved by striping data over both storage devices and network links, provided that a switched network with sufficient bisection bandwidth exists. Unfortunately, the cost of the workstation server, network adapters, and peripheral adapters generally exceeds the cost of the storage devices, increasing the total cost by at least 80% over the cost of simply buying the storage. We have presented a promising direction for the evolution of storage that transfers data directly on the client's network and dramatically reduces this cost overhead.

Our scalable network-attached storage is defined by four properties. First, it must support direct device-to-client transfers. Second, it must provide secure interfaces (e.g. via cryptography). Third, it must support asynchronous oversight, whereby file managers provide clients with capabilities that allow them to issue authorized commands directly to devices. Fourth, devices must serve variable-length objects with separate attributes, rather than fixed-length blocks, to enable self-management and avoid the need to trust client operating systems.

To demonstrate these concepts, we have described the design and implementation of a NASD prototype that manages disks as efficiently as a UNIX filesystem. Measurements of this prototype show that available microprocessor cores embedded into the ASIC of a modern disk drive should provide more than adequate on-drive support for NASD, provided there is cryptographic hardware support for the security functions.

Using a simple parallel, distributed filesystem designed for NASD, we show that the NASD architecture can provide scalable bandwidth. We report our experiments with a data mining application for which we achieve 6.2 MB/s per client-drive pair in a system up to 8 drives, providing 45 MB/s overall. In addition, we describe how conventional distributed filesystems (NFS and AFS) can be ported to use NASD with performance comparable to current server-based systems.

8. ACKNOWLEDGEMENTS

We thank Mike Leis of Quantum for the Trident chip diagram of Figure 2. We thank Paul Mazaitis for his heroic efforts in getting our prototype environment configured and keeping it running. We thank Dan Stodolsky, Bill Courtright, Joan Digney, Greg Ganger, Tara Madhyastha, Todd Mowry, John Wilkes, Ted Wong, and the anonymous reviewers for taking their valuable time to provide us with comments that much improved the paper. We also thank the members of the NSIC working group on network-attached storage, especially Dave Anderson, Mike Leis, and John Wilkes, for many useful conversations and site visits. Finally, we thank all the other members of the Parallel Data Lab make our research possible and enjoyable.

This research is sponsored by DARPA/ITO through DARPA Order D306, and issued by Indian Head Division, NSWC under contract N00174-96-0002. The project team is indebted to generous contributions from the member companies of the Parallel Data Consortium, including: Hewlett-Packard Laboratories, Intel, Quantum, Seagate Technology, Storage Technology, Wind River Systems, 3Com Corporation, Compaq, Data General/Clariion, and Symbios Logic.

9. REFERENCES

[Acharya98] Acharaya, A. et al, Active Disks, *ACM ASPLOS*, Oct 1998.

[Agrawal94] Agrawal, R. and Srikant, R. Fast Algorithms for Mining Association Rules, *VLDB*, Sept 1994.

[Anderson96] Anderson, T., et al. Serverless Network File Systems, *ACM TOCS* 14(1), Feb 1996.

[Anderson97] Anderson, J.M. et al., Continuous Profiling: Where Have All the Cycles Gone?, *ACM SOSP*, Oct 1997.

[Anderson98a] Anderson, D. Network Attached Storage Research, *www.nsic.org/nasd/meetings.html*, March 1998.

[Anderson98b] Anderson, D. Network Attached Storage Research, *www.nsic.org/nasd/meetings.html*, June1998.

[Anderson98c] Anderson, D., et al. Cheating the I/O Bottleneck: Network Storage with Trapeze/Myrinet, USENIX, June 1998.

[Baker91] Baker, M.G. et al., Measurements of a Distributed File System", *ACM SOSP*, Oct 1991.

[Bellare96] Bellare, M., Canetti, R. and Krawczyk, H., Keying Hash Functions for Message Authentication, *Crypto '96*, 1996.

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

[Birrell80] Birell, A.D. and Needham, R.M., A Universal File Server, *IEEE TSE* 6 (5), Sept1980.

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

[Cabrera91] Cabrera, L. and Long, D., Swift: Using Distributed Disk Striping to Provide High I/O Data Rates, *Computing Systems* 4:4, Fall 1991.

[Cao93] Cao, P., et al., The TickerTAIP Parallel RAID Architecture, *ACM ISCA*, May 1993.

[Corbett96] Corbett, P., et al., Proposal for a Common Parallel File System Programming Language, *Scalable I/O Initiative CalTech CACR 130*, Nov 1996.

[Deering95] Deering, S. and Hinden, R., Internet Protocol Version 6 Specification, *RFC 1883*, Dec 1995.

[deJonge93] deJonge, W., Kaashoek, M.F. and Hsieh. W.C. The Logical Disk: A New Approach to Improving File Systems, *ACM SOSP*, Dec 1993.

[Dennis66] Dennis, J.B. and Van Horn, E.C., "Programming Semantics for Multiprogrammed Computations", *CACM* 9, 3, 1966

[Drapeau94] Drapeau, A.L., et al., RAID-II: A High-Bandwidth Network File Server, *ACM ISCA*, 1994.

[Gibson97a] Gibson, G., et al., File Server Scaling with Network-Attached Secure Disks, *ACM SIGMETRICS*, June 1997.

[Gibson97b] Gibson, G., et al. Filesystems for Network-Attached Secure Disks, *TR CMU-CS-97-118*, July 1997.

[Gobioff97] Gobioff, H., Gibson, G. and Tygar, D., Security for Network Attached Storage Devices, *TR CMU-CS-97-185*, Oct 1997.

[Golding95] Golding, R., Shriver, E., Sullivan, T., and Wilkes, J., “Attribute-managed storage,” Workshop on Modeling and Specification of I/O, San Antonio, TX, Oct 1995.

[Gong89] Gong, L., A Secure Identity-Based Capability System *IEEE Symp. on Security and Privacy*, May 1989.

[Grochowski96] Grochowski, E.G. and Hoyt, R.F., Future Trends in Hard Disk Drives, *IEEE Trans. on Magnetics* 32 (3), May 1996.

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

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

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

[Horst95] Horst, R.W. TNet: A Reliable System Area Network, *IEEE Micro*, Feb1995.

[Howard88] Howard, J.H. et al., Scale and Performance in a Distributed File System, *ACM TOCS* 6 (1), February 1988.

[IEEE95] IEEE P1244. “Reference Model for Open Storage Systems Interconnection-Mass Storage System Reference Model Version 5”, Sept 1995

[Intel97] Intel Corporation, Virtual Interface (VI) Architecture, *www.viarch.org*, Dec 1997.

[Karger88] Karger, P.A., “Improving Security and Performance for Capability Systems”, University of Cambridge Computer Laboratory Technical Report No. 149, Oct 1988.

[Knudsen96] Knudsen, L. and Preneel, B., Hash functions based on block ciphers and quaternary codes. *Advances in Cryptology ASIACRYPT*, Nov 1996.

[Lee96] Lee, E.K. and Thekkath, C.A., Petal: Distributed Virtual Disks, *ACM ASPLOS*, Oct 1996.

[Long94] Long, D.D.E., et al, Swift/RAID: A Distributed RAID System, *Computing Systems* 7,3, Summer 1994.

[Maeda93] Maeda, C., and Bershad, B., “Protocol Service Decomposition for High-Performance Networking”, 14th *ACM* SOSP, Dec. 1993.

[McKusick84] McKusick, M.K. et al., A Fast File System for UNIX, *ACM TOCS* 2, August 1984.

[Miller88] Miller, S.W., A Reference Model for Mass Storage Systems, *Advances in Computers* 27, 1988.

[Mitchell81] Mitchell, J. and Dion, J., A Comparison of Two Network-Based File Servers, *ACM SOSP*, Dec 1981.

[MPI95] The MPI Forum, The Message-Passing Interface Standard, *www.mcs.anl.gov/mpi/standard.html*, May 1995.

[Neuman94] Neuman, B.C. and Ts’o, T., Kerberos: An Authentication Service for Computer Networks, *IEEE Communications* 32,9, Sept 1994.

[Patterson88] Patterson, D.A., et al., A Case for Redundant Arrays of Inexpensive Disks, *ACM SIGMOD*, June 1988.

[Pricewatch98] *www.pricewatch.com*, July 1998.

[Riedel98] Riedel, E., et al., “Active Storage for Large-Scale Data Mining and Multimedia” *VLDB*, Aug 1998.

[Sachs94] Sachs, M.W. et al., LAN and I/O Convergence: A Survey of the Issues, IEEE Computer, Dec 1994.

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

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

[Tanenbaum86] Tanenbaum, A.S., Mullender, S.J. and van Renesse, R., Using Sparse Capabilities in a Distributed System, *Sixth Conference on Distributed Computing*, 1986.

[Thekkath97] Thekkath, C., et al., Frangipani: A Scalable Distributed File System, *ACM SOSP*, Oct 1997.

[TPC98] Transaction Performance Council TPC-C Executive Summaries, URL: *www.tpc.org*, Mar 1998.

[TriCore97] TriCore News Release, Siemens’ New 32-bit Embedded Chip Architecture Enables Next Level of Performance in Real-Time Electronics Design, *www.tri-core.com*, Sept 1997.

[VanMeter96] Van Meter, R., Hotz, S. and Finn, G., Derived Virtual Devices: A Secure Distributed File System Mechanism, *Fifth NASA Goddard Conference on Mass Storage Systems and Technologies*, Sep 1996.

[VanMeter98] Van Meter, R., et al., VISA: Netstation’s Virtual Internet SCSI Adapter, *ACM ASPLOS*, Oct 1998.

[Verbauwhede87] Verbauwhede, I. et al., H. Security Considerations in the Design and Implementation of a New DES Chip, *EUROCRYPT*, 1987.

[vonEicken95] von Eicken, T., Basu, A., Buch, V. and Vogels, W. U-Net: A User-Level Network Interface for Parallel and Distributed Computing, *ACM SOSP*, Dec 1995.

[Watson95] Watson, R., Coyne, R., The Parallel I/O Architecture of the High-Performance Storage System (HPSS), *14th IEEE Symposium on Mass Storage Systems*, September 1995.

[Wilkes79] Wilkes, M.V. and Needham, R.M., *The Cambridge CAP Computer and Its Operating System*, 1979.

[Wilkes92] Wilkes, J. Hamlyn - An Interface for Sender-based Communications, *Hewlett-Packard Laboratories Technical Report HPL-OSR-92-13*, Nov 1992.

[Wulf74] Wulf, W.A. et al., “HYDRA: The Kernel of a Multiprocessor Operating System”, *CACM*, 17,6, June 1974