Proceedings of USENIX 1999, Linux Workshop, Monterey, CA, June 9 - 11, 1999.

NASD Scalable Storage Systems

Garth A. Gibson*, David F. Nagle†, William Courtright II*, Nat Lanza*,
Paul Mazaitis*, Marc Unangst*, Jim Zelenka*

School of Computer Science*
Department of Electrical and Computer Engineering†
Carnegie Mellon University, Pittsburgh, PA 15213
www.pdl.cs.cmu.edu

ABSTRACT

The goal of CMU's Network-Attached Secure Disks (NASD) project is to define the next era of storage system interfaces and architectures. To encourage industry standardization of a compliant storage device/subsystem interface, we are working closely with the National Storage Industry Consortium's working group on network-attached storage. Our experimental demonstration of the NASD interface's value is device and filesystem prototype software that delivers the scalability inherent in a NASD storage architecture. To engage the academic community and to provide a reference implementation for industry development, CMU is releasing its Linux and Digital UNIX ports of this software. In this paper, we overview the NASD scalable storage architecture and the code-base we are releasing for Linux.

1. INTRODUCTION

Demands for storage throughput continue to grow due to ever larger clusters sharing storage, rapidly increasing client performance, richer data types such as video, and data-intensive applications such as data mining. For storage subsystems to deliver scalable throughput, that is, linearly increasing application bandwidth and accesses per second with increasing numbers of storage devices and client processors, the data must be striped over many disks and network links [Patterson88], and name lookup and access rights checking must be decentralized [Hartman93, Anderson96]. With current 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 copies data between the storage (peripheral) network and the client (local area) network while adding functions such as concurrency control and metadata consistency.

Our prior work proposed a new scalable-bandwidth storage architecture, Network-Attached Secure Disks (NASD) [Gibson97a, Gibson97b, Gobioff97, Gibson98, Amiri99, Nagle99]. Fundamentally, NASD minimizes server-based data movement by separating management and filesystem semantics from store-and-forward copying and elevating commodity storage's interface to a richer object-based model (SCSI4 perhaps).

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. Of course, advancing storage interfaces and architecture requires industry collaboration and standardization. Fortunately, the storage industry is aggressively seeking to evolve their marketplace [Quantum99, Seagate99]. To promote network-attached storage, CMU is working closely with the National Storage Industry Consortium's (NSIC) working group on network-attached storage devices (www.nsic.org/nasd). Over the past three years, NSIC has hosted about a dozen public workshops where academics and practitioners exchange perspectives on next generation storage. Currently, the core NSIC working group is engaged in developing an ANSI standards proposal for a new storage interface.

Until recently, CMU publications have been sufficient for collaboration in the NSIC effort. Now, to more widely disseminate our work, CMU is providing, for public use, a reference implementation of NASD for the Linux 2.2 and Digital UNIX 3.2 environments. Our reference implementation includes NASD device code (running on a workstation or PC masquerading as a subsystem or disk drive), an NFS-like distributed file system designed to use NASD subsystems or devices, and NASD-inspired striping middleware to provide scalable bandwidth to large striped files. The rest of this extended abstract describes this prototype software and summarizes prior research predictions for its performance.

2. BACKGROUND AND RELATED WORK

Figure 1 illustrates the principal network-attached storage architectures. The simplest implementation runs on a standalone server with attached disks (SAD), as shown in Figure 1a. Data makes two network trips on its way to the client, making the server a potential bottleneck; particularly since a server usually manages a large numbers of disks to amortize cost. Companies such as Network Appliance have improved the performance of SAD implementations, specifically the number of clients supported, by using special purpose server hardware and highly optimized software (SID) [Hitz94].

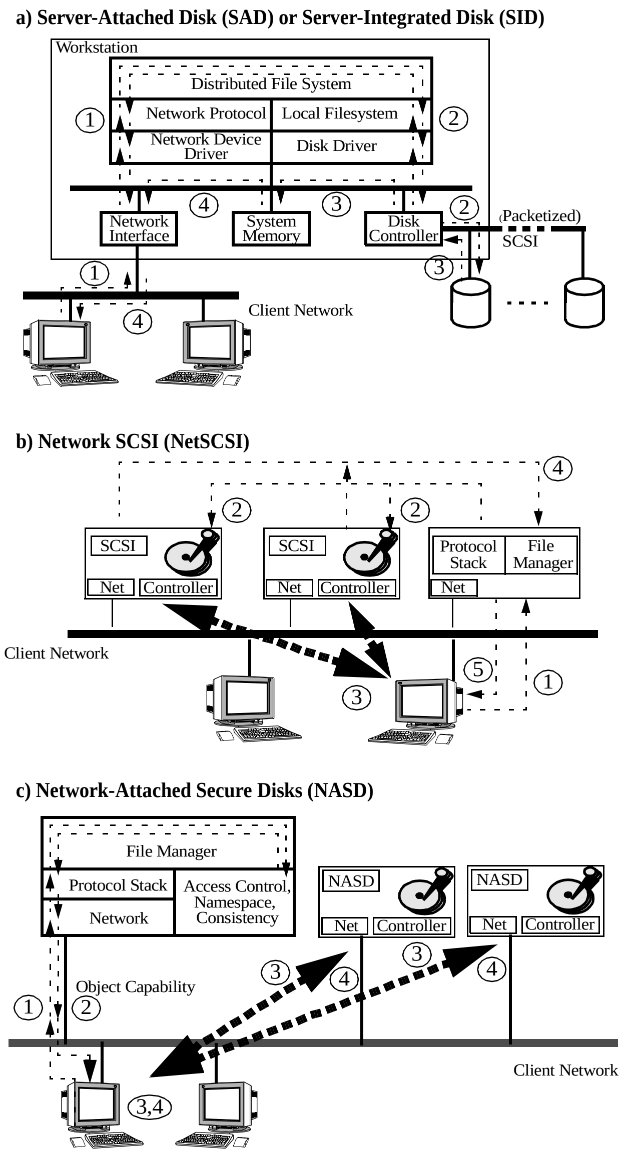

Figure 1: Taxonomy for reaching storage through a network. Figure (a) shows the architecture of storage attached to clients through a separate or a specially integrated server. Clients ask the server for data (1), which forwards the request to storage (2). Storage answers to the server (3), which forwards data to the client (4). Figure (b) shows a system in which the server's forwarded request is a DMA command returning data directly to the client (3). On completion of each DMA, status is returned to the server (4), collected and forwarded to the client (5). This SCSI-over-the-network scheme, NetSCSI, provides scalable bandwidth for large request workloads only. Figure (c) shows our Network-Attached Secure Disks architecture. Only on a first access does a client contact the server for access checks (1). The server grants reusable rights, or capabilities (2). Under normal conditions clients present requests directly to storage (3) which can verify capabilities and directly reply (4). Because lookup, rights verification and small accesses can occur without server intervention, NASD scales bandwidth and accesses per second. A variation on NASD, USC/ISI's Derived Virtual Devices, replaces a capability with a server-installed secure connection with object definition state in each drive [VanMeter98]

If a storage system is (re)organized to "DMA" data directly to clients sharing the client network (NetSCSI), rather than copy it through its server (Figure 1b), the number of network transits for data is reduced from two to one. This organization has been examined extensively and is in use in the HPSS implementation of the Mass Storage Reference Model [Drapeau94, Long94, Watson95].

The third architecture in our taxonomy is the NASD architecture (Figure 1c). NASD embeds disk management functions into the device, presenting storage through a variable-length object interface. In this organization, file managers allow clients to directly access specific storage objects repeatedly by granting a cachable capability. Hence, all data and most control information travels across the network once and there is no expensive store-and-forward computer. A quantitative analysis of each architecture can be found in [Gibson97a].

3. NETWORK-ATTACHED SECURE DISKS

NASD enables cost-effective throughput scaling. NASD presents a flat name space of variable-length *objects* that is both simple enough to be implemented efficiently and 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, policies defining the high-level file system should be managed by a *file manager* and NASD devices should implement simple storage primitives efficiently and operate as independently of the file manager as possible.

NASD's object interface allows data layout management be handled by the disk or storage subsystem. 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 [Lee96]. Object-based storage also supports quality-of-service at the device, transparent performance optimizations, and drive supported data sharing [Anderson98]. Most importantly, an in-drive object store can securely employ storage metadata on behalf of a client without that client needing to consult a file manager on each access. The alternative, exporting storage metadata to clients and enabling client access to arbitrary disk blocks, is worse than insecure, it is accident-prone and puts the entire storage system at risk.

3.1 NASD Interface

Our prototype NASD device software offers a simple, capability-based, object-store interface [Gibson97b], based loosely on the inode interface of the UNIX filesystem [McKusick84]. Our NASD interface contains less than 20 commands including reading and writing object data; reading and writing object attributes; creating and removing

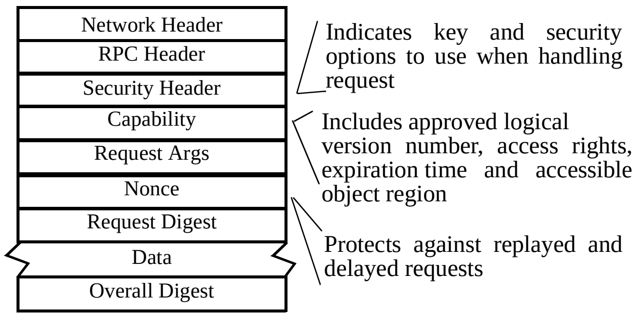
Network Header
RPC Header
Security Header
Capability
Request Args
Nonce
Request Digest
Data
Overall Digest
Indicates key and security options to use when handling request
Includes approved logical version number, access rights, expiration time and accessible object region
Protects against replayed and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

Figure 2: Packet diagram of the major security components of an individual NASD request.

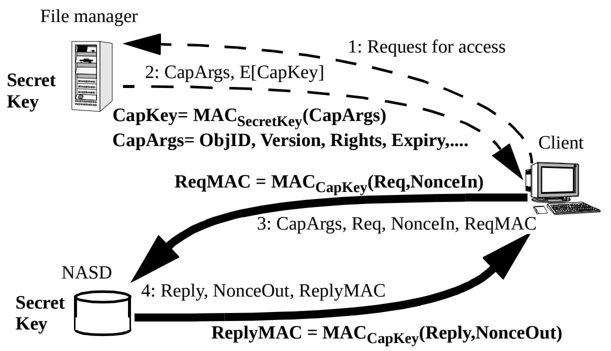
File manager
Secret Key
1: Request for access
2: CapArgs, E[CapKey]
CapKey= MAC_SecretKey(CapArgs)
CapArgs= ObjID, Version, Rights, Expiry,....
Client
ReqMAC = MAC_CapKey(Req,NonceIn)
3: CapArgs, Req, NonceIn, ReqMAC
NASD
Secret Key
4: Reply, NonceOut, ReplyMAC
ReplyMAC = MAC_CapKey(Reply,NonceOut)

Figure 3: The NASD security system in summary [Gobioff97]. The message sequence is outlined in Figure 1(c). When a client seeks a capability from the file manager (1), it gets back a possibly public description of its rights, CapArgs, and a secret (indicated by E[]) session key, CapKey (2). This session key will demonstrate to a NASD drive that a file manager authorized the rights in CapArgs, so its privacy must be maintained by clients using it. When accessing the drive, a client proves its authorization by signing each request with its session key (3). By receiving a copy of CapArgs with each request, the drive is able to reconstruct the session key and verify the request signature. The drive then replies with a similar signature (4). A single secret key is all that a file manager need install to enable a NASD to approve an indefinite number of accesses.

objects; creating, resizing, and removing (soft) partitions; constructing a copy-on-write version of an object; and setting a security key.

Resizeable partitions allow capacity quotas to be managed by a drive administrator. Object attributes describe timestamps and sizes, and facilitate capacity reservation and object linkages for clustering [deJonge93]. A logical version number attribute on an object may be changed by a file manager 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 [Gobioff97]. Figure 2 shows the security related fields in each NASD request message. Figure 3 summarizes the NASD security protocol. Clients obtain capabilities from a file manager using a secure and private protocol external to NASD. A capability consists of a public portion, CapArg, and a private key, CapKey. The CapArg portion specifies what rights are being granted for which object. The CapKey portion is a cryptographic key generated by the file manager using a keyed message digest (MAC) [Bellare96] of CapArg and a secret key shared only with the target drive. A client sends CapArg along with each request, and generates a CapKey-keyed digest of the request parameters and CapArg. Because the drive knows its secret keys and receives CapArg with each request, it can compute the client's CapKey. Using the calculated CapKey, the drive can verify the client-supplied message digest. If any field of the CapArg or the request has been changed, including the object's version number, the digest comparison will fail, the NASD will reject the request, and the client must return to the file manager for a new capability.

These mechanisms ensure the *integrity* of requests in the presence of attacks (both by a rogue client or by an eavesdropping "man in the middle" third party) and simple accidents. Protecting the *integrity and/or privacy* of the data involves potentially very expensive cryptographic operations on all data transferred. 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 [Knudsen96]. For the measurements reported in this paper, we disabled these security computations because our prototype did not and does not yet support such hardware.

4. FILESYSTEMS FOR NASD

Scalability in file managers has traditionally meant that as the total storage capacity increases with new clients' data, the total throughput also increases and response time does not. To demonstrate traditional scaling in a NASD system, we have constructed a distributed filesystem with NFS-like semantics [Sandberg85] and tailored it specifically for NASD. The NASD architecture, however, is also designed to scale a single file's achievable bandwidth with increasing storage capacity. To show scalable bandwidth with NASD we have also constructed library-based user-level parallel access to a striped file.

4.1 NFS 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-like filesystem, 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

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%**

Figure 4: Offloading work from an NFS file manager to NASD drives enables each file manager to support many more drives and clients [Gibson97a]. In this 1996 analysis, a traditional NFS implementation was instrumented and cycle counts recorded for each NFS operation type. The product of these numbers and NFS operation frequencies extracted from the busiest 2% of an NFS trace taken by UC Berkeley estimates the work a traditional NFS server would do when its customers are most critical (column labelled SAD). The columns labelled NetSCSI and NASD model the expected workload on a file manager employing NetSCSI and NASD drives when presented with the same (small access) workload, reported as a percentage of the total work done in the SAD case (*).

file attributes (e.g. owner and mode bits) are stored in the uninterpreted section of the object's attributes. Because the filesystem makes policy decisions based on these file attributes, the client may not directly modify object meta-data; 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 sent directly 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. mode and uid/gid). In a break from NFS, directories are parsed in the client. This allows pathname lookup to avoid accessing the file manager, provided the client has appropriate capabilities.

Figure 4 shows a simple analysis of the amount of file server work entailed by an NFS workload in a traditional server-attached disk system, in a network SCSI system and in the above NASD system. Offloading data access and attribute and directory reads may reduce file manager work by over an order of magnitude in a NASD system.

4.2 Parallel Access to Striped Files

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 5, our layered approach allows the NASD-NFS filesystem to manage a simple "logical" object store provided by our storage management system, *Cheops*. Cheops exports the same object interface as the underlying NASD devices, and maintains the mapping of each NASD-NFS object to the set of objects on the individual devices that implement that NASD-NFS object. 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.

To evaluate performance for I/O-intensive parallel applications, we implemented a simple parallel filesystem, NASD PFS, which offers the SIO low-level parallel filesystem interface [Corbett96] on top of NASD-NFS files that are striped using user-level Cheops middleware. Figure 6 shows the bandwidth scaling of the most I/O bound of the phases (the generation of *1-itemsets*) of a parallel data mining application that discovers association rules in sales transactions [Agrawal94] 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 while a much faster NFS server limits the application to half this performance.

4.3 Continuous Media from NASD

Although not central to scalable file system throughput, continuous media service, such as video, is clearly a requirement for next generation storage systems. NASD's object interface, with its higher level understanding of the data in an object, is naturally extended with type-specific behaviors. To demonstrate this, we have extended our NASD drive prototype with an MPEG-2 streaming video playback system. Moreover, because of our emphasis on scalable throughput, we designed a NASD video service that stripes video data over NASD drives. The video middleware is logically equivalent to Cheops high-bandwidth storage management middleware, but is not yet integrated with Cheops.

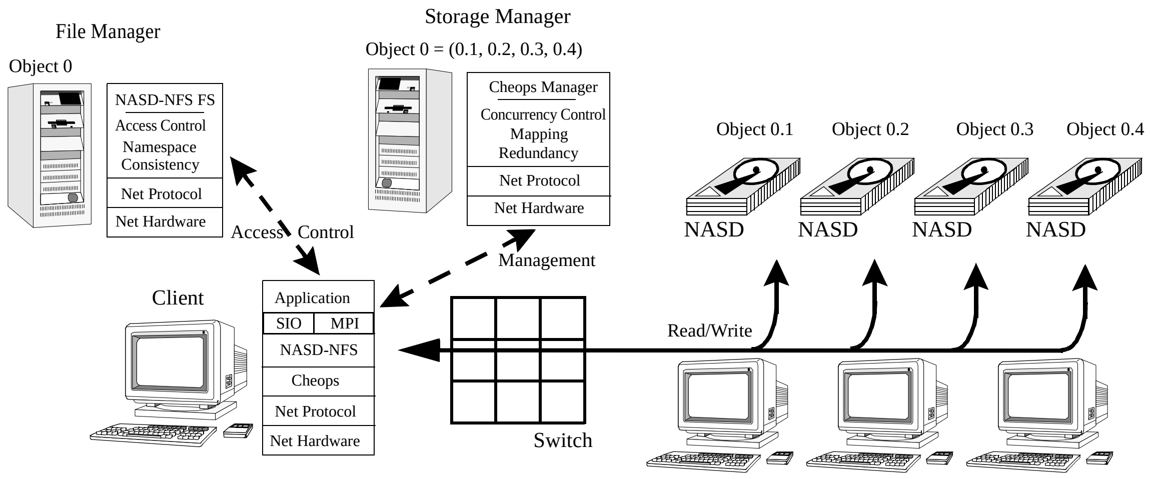

Figure 5: A NASD-optimized parallel access filesystem. NASD PFS is used in conjunction with MPI for parallel applications in a cluster of workstations. The filesystem manages objects which are not directly backed by data. Instead, they are backed by a storage manager, Cheops, which redirects clients to the underlying component NASD objects. Our parallel filesystem extends a simple NASD-NFS filesystem interface with the SIO low-level interface [Corbett96] and inherits a name service, directory hierarchy, and access controls from the filesystem.

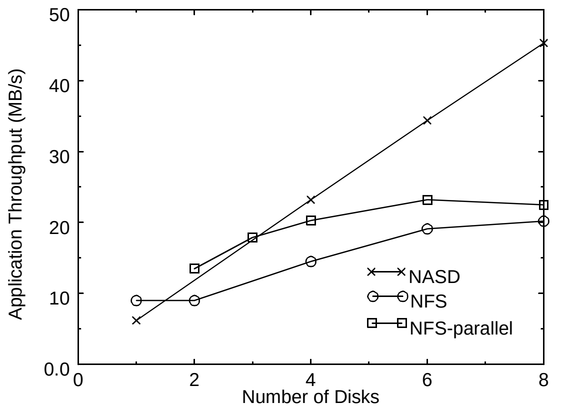

Figure 6: Bandwidth in a parallel data mining application [Gibson98]. This figure shows the aggregate application bandwidth during a run of the “1-item frequent sets” application on 300 MB of sales transactions. The *NASD* line shows the bandwidth of *n* clients reading over a single OC-3 ATM link (using DCE RPC over UDP) from a single file striped across *n* drives. Bandwidth scales linearly to 45 MB/s. Each NASD prototype drive runs on a DEC Alpha 3000/400 (133 MHz, 64 MB, Digital UNIX 3.2g) with two ST52160 Medallist disks attached by two 5 MB/s SCSI busses. The performance of this obsolete system is comparable to storage subsystems today.
Both traditional NFS configurations show the maximum achievable bandwidth on a DEC Alpha 500/500 (500 MHz, 256 MB, Digital UNIX 4.0b) with the given number of ST34501W Cheetah disks (13.5 MB/s), each disk twice as fast as the NASDs used in this test, and up to 10 clients each reachable over two OC-3 ATM links. The *NFS* line shows that the performance when all clients read from a single file striped across *n* disks on the server, bottlenecks near 20 MB/s. This configuration causes poor read-ahead performance inside the NFS server, so we also report the *NFS-parallel* case in which each client reads, through the one server, from a private replica of the file located on an client-devoted disk. This configuration performs better than the single, shared file case, but still achieves only 22.5 MB/s at best.

5. CONCLUSIONS

The Network-Attached Secure Disks (NASD) project is defining new, more scalable storage interfaces characterized by four properties. First, direct storage-device-to-client transfers. Second, secure interfaces (e.g. via cryptography). Third, asynchronous oversight, whereby file managers provide clients with capabilities that allow them to issue authorized commands directly to devices. Fourth, an interface that provides variable-length objects with separate attributes, rather than fixed-length blocks, to enable self-management and avoid the need to trust client operating systems.

In this paper we have reviewed the NASD architecture. We report: 1) how NASD scales the number of clients that a file manager supports by offloading metadata lookup and access checking to storage devices; and 2) how NASD scales single file bandwidth by striping data and distributing cachable mapping information to clients that enables direct and parallel accesses to a file’s component objects.

This paper’s larger purpose is to introduce NASD to file and storage system researchers and practitioners interested in the CMU NASD public code. We intend to release this code soon after the June 1999 Extreme Linux workshop. It includes Linux (X86) and Digital UNIX (Alpha) ports of NASD device code, a file manager and client module for a NASD-customized NFS-like filesystem, and pseudo-NASD middleware allowing clients to locate and directly access components of striped files without compromising data integrity.

6. AVAILABILITY

Code will soon be available on the Extreme NASD web pages, *http://www.pdl.cs.cmu.edu/extreme/*.

We have implemented and are releasing a prototype of the NASD drive software (including security), a file manager and client for our prototype filesystem, which we call EDRFS, and a prototype of the Cheops storage manager. Some configuration and management tools for the NASD drive and EDRFS filesystem are also included.

The prototype code as a whole is easily ported. At this time the drive and file manager all run as user processes on Linux, Digital UNIX 3.2G, IRIX, and Solaris. In addition, the Digital UNIX port can run the drive and file manager inside the kernel.

On Linux, both the drive and EDRFS file manager can be executed as either a user process or as a loadable kernel module (LKM). Other modules are mostly implemented as user processes, although the EDRFS client is available only as a LKM.

Because NASD is intended to directly manage drive hardware, our NASD object system implements its own internal object access, cache, and disk space management modules and interacts minimally with its host operating system.

For communication, our prototype uses either a customized TCP-based RPC (SRPC) or DCE RPC over UDP/IP. The DCE implementation of these networking services is quite heavyweight, and is not available inside the Linux kernel. The appropriate protocol suite and implementation is currently an issue of active research [Anderson98, VanMeter98, Nagle99].

7. ACKNOWLEDGEMENTS

This release of NASD for Linux is based upon the research of the entire NASD team—we specifically acknowledge Khalil Amiri, Jeff Butler, Fay Chang, Howard Gobioff, Charles Hardin, Erik Riedel and David Rochberg, whose research enabled this release. This research is sponsored by DARPA/ITO through DARPA Order D306, under contract N00174-96-0002. The project team is indebted to generous contributions from our corporate sponsors: Clariion, Compaq, Hewlett-Packard Laboratories, Hitachi, IBM, Intel, LSI Logic, Novell, Quantum, Seagate Technology, Siemens, Storage Technology, Wind River Systems, and 3Com Corp.

8. REFERENCES

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

[Amiri99] Amiri, K., Gibson, G, Golding, R., Scalable Concurrency Control and Recovery for Shared Storage Arrays, *TR CMU-CS-99-111*, February 1999.

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

[Anderson98] Anderson, D., Network Attached Storage Research, *www.nsic.org/nasd/meetings.html*, Mar., June 1998.

[Bellare96] Bellare, M., et. al., Keying Hash Functions for Message Authentication, *Crypto '96*, 1996.

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

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

[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.

[Gibson98] Gibson, G., et al, A Cost-Effective, High-Bandwidth Storage Architecture, *ACM ASPLOS*, October 1998.

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

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

[Hitz94] Hitz, D., Lau, J., Malcolm, M., File System Design for an NFS File Server Appliance, *USENIX*, Jan. 1994.

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

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

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

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

[Nagle99] Nagle, D., Ganger, G., Butler, J., Goodson, G., Sabol, C., Network Suppport for Network-attached Storage, HotInterconnects 1999, Stanford, CA, August, 1999.

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

[Quantum99] Backgrounder on Storage Systems Strategy, *http://www.quantum.com/corporate/pc/leadership/storage-systems.html*, March 1999.

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

[Seagate99] JINI: A Pathway for Intelligent Network Storage, *http://www.seagate.com/corp/vpr/literature/papers/jini.shtml* Jan. 1999.

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

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