Proceedings of Hot Interconnects 1999, August 18 - 20, 1999, Stanford University, Stanford, California, U.S.A.

Network Support for Network-Attached Storage

David F. Nagle, Gregory R. Ganger, Jeff Butler, Garth Goodson, and Chris Sabol

Carnegie Mellon University, Pittsburgh, PA 15213-3891
{bassoon, ganger}@ece.cmu.edu http://www.pdl.cs.cmu.edu

1 Introduction

Storage systems represent a vital market with storage densities growing at 60%/year, resulting in 35%-50%/year decreases in the cost per byte. In recent years, the amount of storage sold almost doubled each year and is expected to sustain annual growth of at least 60%. Secondary storage has a healthy place in future computer systems.

While many storage products are directly attached to personal computers, most disk array products (65% and rising) are deployed in local area network file servers. This centralization of storage resources enables effective sharing, better administrative control and less redundancy. However, it also increases dependence on network and file server performance. With the emergence of high-performance cluster systems based on commodity personal computers and scalable network switching, rapidly increasing demands on storage performance are anticipated. Specifically, storage performance must cost-effectively scale with customer investments in client processors, network links and storage capacity.

Unfortunately, current distributed file system architectures severely limit scalable storage. In current distributed file systems, all storage bytes are copied through file server machines between peripheral buses (typically SCSI) and client LANs. In essence, these file server machines act as application-level inter-network routers, converting namespaces (disk block versus file range) and protocol layers (SCSI versus RPC/UDP/IP). This is a critical limitation for cost-effective scalable storage. Specifically, the sustained bandwidth of storage devices is rapidly outstripping installed interconnection technologies and rendering store-and-forward servers impractical. With disk data rates growing at 40% per year, 25-40 MB/s sustained disk bandwidths will be a reality by the end of the decade. With this much bandwidth from each commodity drive, conventional server architectures can not be cost-effective.

Point of contact: Prof. David Nagle, Department of Electrical and Computer Engineering, Carnegie Mellon Univ, Pittsburgh, PA 15213.

This work was supported in part by DARPA contract N00174-96-0002, by an NSF Graduate Research Fellowship, and by the Parallel Data Consortium member companies (3COM, Clariion, Hitachi, HP, IBM, Intel, LSI Logic, Novell, Quantum, Seagate, Siemens, Wind River Systems). The US government has certain rights in this material. This work was performed in part according to the National Storage Industry Consortium (NSIC) NASD project agreement. The views contained in this document are those of the authors and should not be interpreted as representing the policies, either expressed or implied, of any supporting agency.

Storage devices, however, are already effective network data transfer engines. For example, Seagate's Fibre Channel Baracuda drives burst packetized SCSI at 1 Gbps. Moreover, through careful hardware support for interlayer processing, the marginal cost of these network-attached disk drives is expected to be similar to that of high-end drive interfaces, such as differential SCSI [Anderson95]. It is our contention that cost-effective scalable storage performance depends on eliminating the file server's role as an inter-network router. Instead, we advocate exploiting the drive's ability to inject packets directly into the clients' network at high-bandwidth. With effective network-attached storage, striping of data over multiple devices can effectively scale storage bandwidth [Patterson88, Hartman93].

The other dimension to delivering scalable storage bandwidth is efficient networking. Networking technologies, such as Fibre Channel, are specifically designed for storage systems. However, traditional client-server protocol stacks and operating system software layers often copy data several times in delivering it to applications, significantly reducing network-attached storage performance. Newer technologies, such as user-level networking, avoid this problem by allowing applications to directly access the network. Hence, user-level networking could be an ideal substrate for network-attached storage.

This papers examines networking requirements for storage and the integration of user-level networking with network-attached storage (NAS). To provide context for the networking demands of NAS, we begin by describing alternative network-attached storage architectures and CMU's network-attached storage system. Next, we survey storage's networking requirements and describe how one user-level networking architecture, the VI Architecture (VIA), can be effectively mapped onto our network-attached storage prototype.

2 Managing Network-Attached Storage

The fundamental goal of our network-attached storage research is to enable scalable storage systems while minimizing the file manager bottleneck. One solution is to use homogeneous clusters of trusted clients that issue unchecked commands to shared storage. However, few environments can tolerate such weak integrity and security guarantees. Even if only for accident prevention, file protec-

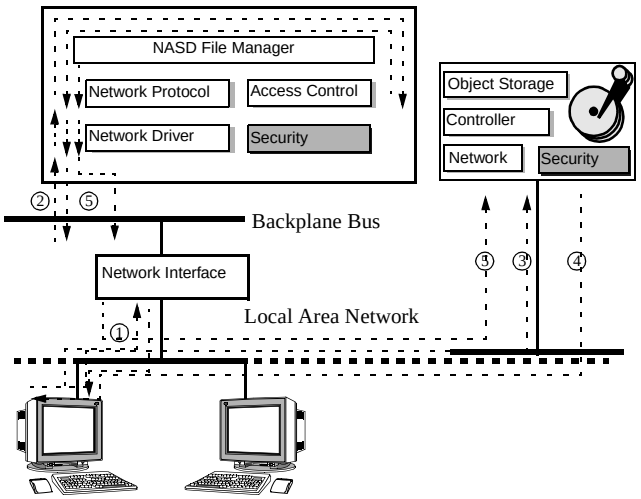

Figure 1: Network-attached secure disks (NASD) are designed to offload more of the file system's simple and performance-critical operations. For example, in one potential protocol, a client, prior to reading a file, requests access to that file from the file manager (1), which delivers a capability to 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).

tions and data/metadata boundaries should be checked by a small number of administrator-controlled file manager machines.

To provide this more appropriate degree of integrity and security, we identify two basic architectures for direct network-attached storage [Gibson97]. The first, NetSCSI, makes minimal changes to the hardware and software of SCSI disks, while allowing NetSCSI disks to send data directly to clients, similar to the support for third-party transfers already supported by SCSI [Miller88, Drapeau94]. Drives' efficient data-transfer engines ensure that each drive's sustained bandwidth is available to clients. Further, by eliminating file management from the data path, manager workload per active client decreases. Cryptographic hashes and encryption, verified by the NetSCSI disks, can provide for integrity and privacy. The principal limitation of NetSCSI is that the file manager is still involved in each storage access; it translates namespaces and sets up the third-party transfer on each request.

The second architecture, Network-Attached Secure Disks (NASD, see Figure 1), relaxes the constraint of minimal change from the existing SCSI interface. The NASD architecture provides a command interface that reduces the number of client-storage interactions that must be relayed through the file manager, thus avoiding a file manager bottleneck without integrating file system policy into the disk. In NASD, data-intensive operations (e.g., reads and writes) go straight to the disk, while less-common policy making operations (e.g., namespace and access control manipulations) go to the file manager.

Because clients directly request their data, a NASD drive must have sufficient metadata to map and authorize a request for disk sectors. Authorization, in the form of a time-limited capability applicable to a given file's map and contents, is provided by the file manager to protect the manager's control over storage access policy. The storage mapping metadata is maintained by the drive, allowing smart drives to better exploit detailed knowledge of their own resources to optimize data layout, read-ahead, and cache management [Cao94, Patterson95, Golding95]. This is precisely the type of value-add 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 striped files [Hartman93], Macintosh-style resource forks, or logically-contiguous chunks of complex files [deJong93].

3 NASD Implementation

To experiment with the performance and scalability of NASD, we designed and implemented a prototype NASD storage interface, ported two popular distributed file systems (AFS and NFS) to use this interface, and implemented a striped version of NFS on top of this interface [Gibson97b]. The NASD interface offers logical partitions containing a flat name space of variable length objects with size, time, security, clustering, cloning, and uninterpreted attributes. Access control is enforced by cryptographic capabilities authenticating the arguments of each request to a file manager/drive secret through the use of a digest.

In the NASD/AFS and NASD/NFS filesystems, frequent data-moving operations and attribute read operations occur directly between client and NASD drive, while less-frequent requests are handled by the file manager. NFS's simple distributed filesystem model of a stateless server, weak cache consistency, and few mechanisms for filesystem management made it easy to port to a NASD environment; based on a client's RPC request opcode, RPC destination addresses are modified to deliver requests to the NASD drive. Our AFS port was more interesting, specifically in maintaining the sequential consistency guarantees of AFS, and in implementing volume quotas. In both cases, we exploited the ability of NASD capabilities to be revoked based on expired time or object attributes (e.g., size).

Using our implementations[1], we compared NASD/AFS and NASD/NFS performance against the traditional Server-Attached Disk (SAD) implementations of AFS and NFS.

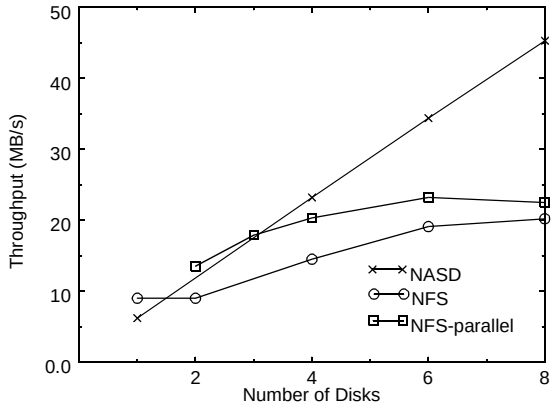

Figure 2: Scaling of a parallel data mining application. Measurements from our original NASD prototype running DCE-RPC over UDP show aggregate bandwidth computing frequent sets from 300 MB of sales transactions. All configurations show the maximum achievable bandwidth with the given number of disks or NASD drives and up to 10 clients. The *NASD* line shows the bandwidth of clients reading from a single NASD file striped across *n* drives and scales linearly to 45 MB/s. 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 separate file on an independent disk through the same server. This configuration performs better than the single file case, but only raises the maximum bandwidth from NFS to 22.5 MB/s, still a factor of two from what NASD provides.

Our perfectly load-balanced large-read benchmark (512K chunks) showed that NASD is able to scale linearly, up to the drive's aggregate transfer bandwidth, while SAD NFS and AFS systems are limited by the data throughput of the server to just three drives.

To demonstrate the ability of striping to automatically load balance requests in a NASD environment, we implemented a striped NFS prototype. In this implementation, striping is transparent to both NASD/NFS file manager and NASD drives, encapsulating striping control in a separate striping manager that exports a NASD interface to the NASD/NFS file manager. Figure 2 shows the results for a data mining application consisting of 1 to 8 clients. Striped NASD/NFS scales linearly while SAD's throughput saturates quickly.

4 Network Support for Network-Attached Storage

The success of the NASD architecture depends critically on its networking environment. Clearly, support for high-bandwidth, large data transfers is essential. Unfortunately, traditional client-server communication paths do not support efficient network transport. For example, measurements of our NASD prototype drive (running DCE/RPC over UDP/IP) show that non-cached read or write requests can easily be serviced by modest hardware. However, requests that hit in the drive cache incur order-of magnitude increases in service time due to the NASD drive and client both spending up to 97% of their time in the network stack [Gibson98].

This problem with traditional protocol stacks forces network-attached storage to explore alternative techniques for delivering scalable bandwidth to client applications. In the next section, we discuss a solution: the integration of user-level networking with NASD. Several other network issues are also important to consider in a NASD environment. These include:

File system traffic patterns: Network file access entails significant small message traffic: attribute manipulation, command and status, small file access, and metadata access. In our NASD prototype, modest size messages (between 100 and 300 bytes) account for over 75% of the total messaging in the storage system. Network protocols that impose significant connection overhead and long code-paths will be a primary determinant of cached storage response time and overall storage system scalability.

Drive Resources: Disk drives are price-conscious, resource-constrained devices. Current drives contain only 1-4 MBytes of RAM but are capable of streaming 25 MB/sec and bursting at 100 MBytes/sec. This efficiency is achieved with hardware-based network support and streamlined protocols. However, network trends are increasing the resource requirements and complexity of drives. For example, Fibre-Channel's rich set of service classes requires significant support that is unnecessary for many storage applications; a much smaller subset can be deployed to meet storage's essential needs [HP99].

Cluster SAN, LAN and WAN: High-performance clusters will be based on commodity system area networks (SANs), which will support protocols optimized for high-bandwidth and low-latency. Such SANs are a natural fit for the needs of scalable storage in a cluster environment.

LAN-based workgroups, however, typically access distributed file systems using internet-based protocols (e.g., RPC/UDP/IP), forcing them to suffer significant protocol processing overhead. Incorporating client LANs into a cluster SAN can overcome this problem. Specifically, using the same media and link layers for both cluster and workgroup storage will increase SANs' commodity advantages, enable thin protocols, and improve support for small messages. When necessary, remote access to cooperating workgroups' storage can use optimized servers or gateway protocol converters.

Reliability: Storage requires reliable delivery of data for most applications. Ideally, the network should provide reliability between client and storage. However, in most cluster SANs, errors are rare enough that the cost of complex hardware-based error handling is outweighed by the

1 The experimental testbed contained four NASD drives, each one a DEC Alpha 3000/400 (133MHz, 64 MB, Digital UNIX 3.2g-3) with a single 1.0 GB HP C2247 disk. We used four Alpha 3000/400's as clients. All were connected by a 155 Mb/s OC-3 ATM network (DEC Gigaswitch/ATM).

flexibility of efficiently exposing infrequent errors to higher-level software or firmware (e.g., Fibre Channel Class 3). Essential to efficient software error handling is hardware support to quickly identify errors (e.g., hardware checksum) and support for network problems that endpoint software cannot solve alone (e.g., switch buffer overflow). This provides both the efficiency and flexibility applications need for handling errors.

Failure Detection/Recovery: Large storage systems are designed to detect and handle drive failures (e.g., RAID). Client-based RAID over network-attached storage requires that the system expose both drive and network failures. Efficient recovery requires rapid remapping of communication channels to alternative storage devices.

Multicast: On-site and off-site redundancy is widely used for ensuring data integrity and availability. Current solutions either force clients to transmit copies of data to each storage replica or require one replica to transmit the data to other replicas. This creates significant bandwidth requirements in the client memory system, the client network interface, and/or the storage node responsible for replication. Network support for multicast, either in a NIC that automatically replicates data or in the network fabric, can significantly reduce bandwidth requirements.

5 Mapping NASD to User-level Networks

Effective network-attached storage requires efficient delivery of data to client applications, efficient in-drive implementations, and effective physical network support for storage traffic. In this section, we discuss our integration of NASD with modern networking techniques to deliver scalable bandwidth.

Research focused on providing applications with high-bandwidth has emphasized reducing data copies using two basic solutions: 1) integrating buffering/caching in the OS [Rodrigues97, Pai99] or 2) direct user-level access to the network [vonEicken95, Buzzard95, Kaashoek97]. Buffer integration is very effective in environments where caching plays a central role (e.g., small or high-locality data sets). High-bandwidth applications, however, are better served by direct access to the data stream. Hence, user-level networking provides an excellent opportunity for delivering scalable storage bandwidth to client applications.

To investigate the effectiveness of user-level networking for NASD, we have experimented with NASD on top of the VI-Architecture (VIA) interface (Figure 3; for a complete description of VIA see http://www.viarch.org). While numerous user-level networking architectures currently exist [Bhoedjang98], we chose VIA for three reasons. First, VIA integrates user-level NIC access with protection mechanisms, providing a simple application/NIC interface with basic send and receive primitives as well as remote DMA for reads and writes. Second, commercial VIA implementations are now available, providing applications with full network bandwidth while consuming less than 10% of a modern CPU's cycles [Giganet98]. Third, VIA is receiving significant industrial support from software and hardware vendors. These characteristics make VIA a potentially powerful substrate for commodity network-attached storage, which requires both high-performance and wide-spread adoption.

Integrating VIA with NASD presents several interesting differences from traditional workstation-based client-server implementations. Within the drive, the NASD software runs in kernel mode[2], making drive-based user-level networking unnecessary. The drive, however, must support the external VIA interface and semantics. This requires the drive to support VI connections for each client application requesting data from the storage device. In large-scale NASD systems, this can result in 100s of connections per drive, each requiring approximately 2KBytes of state. Further, each VI connection must provide buffers for receiving data transfers (writes) from client applications. Using an aggressive flow-control mechanism similar to [Chun99], each connection would require approximately four 8KByte buffers for a Gigabit stream of data. To support 100 connections, a drive would need to commit over 3 MBytes of RAM — a significant increase over modern disk drives.

At the interface level, mapping client read commands to VIA is straightforward. A client application preposts a set of buffers (wired memory) matching the read request size, and then issues a NASD read command to the drive. The drive returns the data, in order, to the client's preposted buffers. File system writes follow a similar path, but potentially require the drive to provide a significant amount of buffering (preposted buffers) to absorb large bursts (100 MB/second today over Fibre Channel and 200 MB/second in the near future). Ironically, modern drives are only capable of writing data to disk platters at 25 MB/sec peak. Therefore, network bursts often copy data from client RAM to drive RAM where it sits for milliseconds before being committed to stable storage. Moreover, many operating systems buffer disk writes for some period of time (e.g., 30 seconds in Unix) before sending writes to storage, creating sudden and often unnecessary bursts of write traffic that tax on-drive memory resources.

VIA's Remote DMA (RDMA) support provides an interesting solution to reducing the write-burst problem while minimizing on-drive buffer resources. Instead of clients sending data to the drive with a write command, a cli-

2 The original implementations of the NASD drive support both a user-level and kernel-level drive running over DCE-RPC/UDP. Our new version of NASD replaces UDP with VIA while using the user-level drive — however, production NASDs would run an embedded operating system with the NASD drive software running in "kernel" mode.

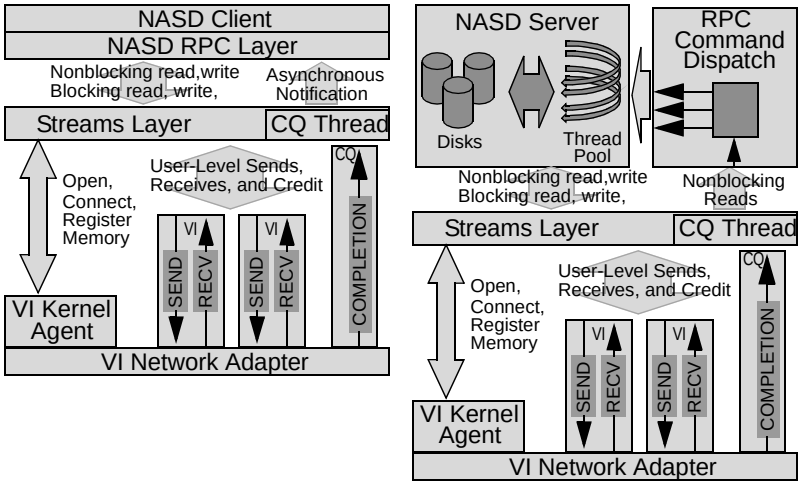

Figure 3: NASD over VIA
Both the client and NASD drive use a streams layer to interface the NASD RPC to the underlying VIA library interface (VIPL). This streams layer provides credit based flow control, buffer management and a stream of bytes abstraction. In addition, streams provides blocking and nonblocking read and write calls and call-backs when nonblocking reads or writes become possible. The drive uses these call-backs with non-blocking reads to implement command dispatching. NASD threads handle the request through a combination of blocking and nonblocking operations. The client RPC layer handles RPC requests using blocking streams operations.

ent could send only a write command with a pointer to the data stored in the client's pinned memory. The drive could then use the VIA RDMA read command to pull data out of the client's memory (without interrupting the client CPU), enabling drive-based scheduling of data transfers. With network latencies much lower than disk access times, the drive would treat the client's RAM as an extended cache/buffer, co-scheduling network transfers and disk transfers to minimize buffer requirements on the drive.

Client-side write-back caching can also benefit from VIA's RDMA. Client writes could cache data in the client write-back cache, while forwarding the write command to the drive. The drive can schedule these writes during idle periods, using VIA RDMA reads to pull the data out of the client cache. Writes could thus be committed to disk between sync periods, significantly reducing the amount of data burst to the drive during a sync. Perhaps more importantly, the drive can request data in any order, since RDMA uses a memory model rather than a stream model. This allows the drive to read data from the client in an order optimized for disk scheduling. Optimized disk scheduling also applies to file reads, were the drive can use RDMA writes to send data into an application's memory in any order, minimizing the amount of on-drive buffering.

It is also important to efficiently support mirrored storage. Currently, mirroring requires two VI connections (one per drive) and clients must send two separate copies of the data. This imposes a significant load on the client's memory system and network interface. To avoid this problem, some existing systems ask the storage device to mirror the data [EMC98]. This creates a large window of failure and doubles the replicating node's load. An alternative is to support multicast in the network interface or in a network switch. We are currently exploring both solutions.

Figure 3 depicts our current implementation of NASD over VIA. Both the client and NASD drive use a streams layer to interface the NASD RPC to the underlying VIA library interface (VIPL). This streams layer provides credit based flow control, buffer management and a stream of bytes abstraction. All of the VI commands, except RDMA, are currently supported and we are extending the system to support RDMA.

5.1 Network Striping and Incast

File striping across multiple storage devices provides significant bandwidth gains for applications. Instead of hiding striping behind a server machine, NASD exposes it to the client. Clients use a middleware layer, called Cheops, to aggregate multiple NASD objects into a single (striped) application object. Figure 2 shows Cheops performance.

Unfortunately, current networks do not provide adequate support for striped storage's *incast* (i.e., many-to-one) traffic pattern. Ideally, a receiving client should receive equal bandwidth from each source. However, without link-level flow control, network buffer overruns occur frequently, forcing costly retransmissions and significantly increasing latency. Worse, even with link-level flow control, network fabrics can create uneven distributions of bandwidth. Figure 4 demonstrates this problem using a Myrinet M2F-SW8, 8-port full-crossbar LAN switch. For some configurations (i.e., 1, 2, 3, 5, 7 sources) bandwidth is evenly distributed, while for 4 or 6 sources, there is a 3X variation in bandwidth between sources. One solution is to utilize network-based flow control mechanisms (i.e., credit-based) to correct the bandwidth discrepancy. Unfortunately, network-managed credit-based flow control does not understand the higher-level notion of striping. VIA sidesteps this problem by pushing flow control to the application level, allowing application-level striping software to manage credit-based flow control across the striped storage. Figure 4 (synchronized) shows the same Myrinet switch and incast traffic pattern, but with stripe-conscious flow control at the storage management layer, which balances the flow of data from senders to receiver.

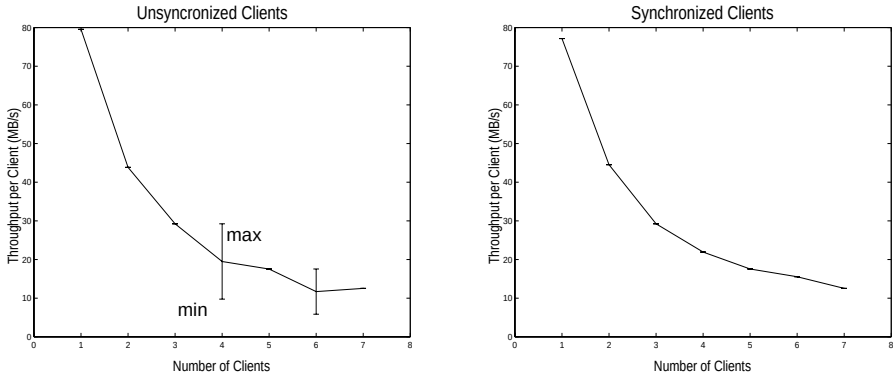

Figure 4: Incast Traffic Pattern
For a striped workload, the graph shows the average client bandwidths using 3 data points per configuration (average, minimum and maximum). The line passes through the average bandwidth across all the sending clients. The min and max data points denote the maximum and minimum average bandwidth seen at the corresponding sending clients. For the 1, 2, 3, 5, and 7 client configurations, the average, min and max data points are almost identical, demonstrating that link-level + network credit flow control provides even distribution of bandwidth among the sending clients. However, the "Unsynchonized Clients" graph shows that the 4 and 6 client configurations suffer significant variation. The "Synchronized Clients" graph shows how moving credit-based flow control into the storage management layer guarantees even bandwidth at all sending machines (clients).

6 Conclusions

High-performance, low-latency networking is essential to achieving the potential of scalable network-attached storage. User-level networking solutions, such as VIA, have the potential to do this, but must be mindful of the amount of on-drive resources required — connection state and buffering can consume considerable resources. However, Remote DMA can help minimize drive resources while providing a great deal of flexibility in drive scheduling and buffer management. Further, VIA's application-level flow control enables aggregation of flow control across arbitrary storage components, something low-level network flow control is not designed to support.

Bibliography

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

[Bhoedjang98] Bhoedjang, R.A.F., et al., "User-Level Network Interface Protocols", IEEE Computer, November 1998, pp. 53-60.

[Buzzard95] Buzzard, G., et al., "Hamlyn: a high performance network interface with sender-based memory management", HotIO III, August 1995.

[Cao94] Cao, P., et al., "The TickerTAIP parallel RAID Architecture," ACM Transactions on Computer Systems 12(3). August 1994, 236-269.

[Chun97] Chun, B., et al., "Virtual Network Transport Protocols for Myrinet," Hot Interconnects V,. August 1997.

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

[deJonge93] de Jonge, W., Kaashoek, M. F., Hsieh, W.C., "The Logical Disk: A New Approach to Improving File Systems," 14th SOSP, Dec. 1993.

[EMC98] EMC Corporation, "No Data Loss Standby Database", White Paper, September 1998.

[Gibson97] Gibson, G.A., et al., "File Server Scaling with Network-Attached Secure Disks," SIGMETRICS 1997, June 1997.

[Gibson97b] Gibson, G.A., et al., "Filesystems for Network-Attached Secure Disks," in preparation, CMU-CS-97-118.

[Gibson98] Gibson, G. A., et. al., "A Cost-Effective, High-Bandwidth Storage Arch.", 8th ASPLOS, 1998.

[Giganet98] Giganet. http://www.giganet.com.

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

[Hartman93] Hartman, J.H., Ousterhout, J.K., "The Zebra Striped Network File System", 14th SOSP, Dec. 1993, pp. 29-43.

[HP99] Hewlett Packard, "Tachyon TL 33 MHz PCI to Fibre Channel Controller", Technical Data, April 1999.

[Kaashoek97] Kaashoek, M.F., et al., "Application Performance and Flexibility on Exokernel Systems", Sixteenth ACM Symposium on Operating System Principles, December 1997.

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

[Pai99] Pai, V., Druschel, P., Zwaenepoel, W., "IO-Lite: A Unified I/O Buffering and Caching System", OSDI 99, New Orleans, LA, February 1999.

[Patterson88] Patterson, D.A., et. al., "A Case for Redundant Arrays of Inexpensive Disks (RAID)", 1988 SIGMOD, June 1988, pp. 109-116.

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

[Rodrigues97] Rodrigues, S.H., Anderson, T.E., Culler D.E., "High-Performance Local Area Communication with Fast Sockets", USENIX97, 1997.

[vonEicken95] von Eicken, T., et al., "U-Net: A User level Network Interface for Parallel and Distributed Computing" Department of Computer Science. Cornell University. Ithaca, NY. 14853. 8/21/95.