

Optimizing Throughput in a Workstation-based Network File System over a High Bandwidth Local Area Network

*Theodore Faber*¹

University of Southern California/Information Sciences Institute
4676 Admiralty Way
Marina del Rey, CA 90292
Phone: 310-821-5080 x190
FAX: 310-823-6714
faber@isi.edu

ABSTRACT

This paper describes methods of optimizing a client/server network file system to take advantage of high bandwidth local area networks in a conventional distributed computing environment. The environment contains hardware that removes network and disk bandwidth bottlenecks. The remaining bottlenecks at clients include excessive context switching, inefficient data translation, and cumbersome data encapsulation methods. When these are removed, the null-write performance of a current implementation of Sun's Network File System improves by 30%. A prototype system including a high speed RAM disk demonstrates an 18% improvement in overall write throughput. The prototype system fully utilizes the available peripheral bandwidth of the server.

Keywords:

Distributed Systems, Network File Systems, High Speed LANs

1. Introduction

This paper describes optimizations made to a client/server networked file system environment to make full use of the high bandwidth of emerging local area networks (LANs). It concentrates on improving the bulk data transfer capability of the file system, by removing unnecessary data translations and reducing context switches at the clients. Application of these techniques results in a 30% increase in Sun Network File System Version 2 (NFS)[1,2] write bandwidth using the ATOMIC LAN[3] when files are not written to disk. The cost of implementation of data encapsulation methods, specifically Sun XDR[4], is also assessed. A prototype system was built using a Texas Memory Systems SAM-300 RAM disk; the system exhibits an 18% improvement in overall NFS write throughput, and fully utilizes the server peripheral bandwidth.

The ATOMIC-2 research group at the University of Southern California's Information Sciences Institute (USC/ISI) is exploring the issues associated with introducing the ATOMIC LAN, a 640 Mbps LAN invented at ISI and the California Institute of Technology[3], into a distributed computing environment.

¹ This work is supported by the Defense Advanced Research Projects Agency through Ft. Huachuca contract #DABT63-93-C-0062 entitled "Netstation Architecture and Advanced Atomic Network." The views and conclusions contained in this document are those of the author and should not be interpreted as representing the official policies, either expressed or implied, of the Department of the Army, the Defense Advanced Research Projects Agency, or the U.S. Government.

Users of this environment require a high-speed network file system that allows a client to utilize as much network bandwidth as possible for remote file access.

Most network file systems cannot use the increased bandwidth that technologies like the ATOMIC LAN provide. Because network file systems often provide less throughput than file transfer protocols, it can be more efficient to transfer a file to a local file system using a file transfer protocol and operate on the data there than to use a network file system.

If network file systems are to exploit high bandwidth environments, they must be able to make better use of bandwidth. File access bandwidth is becoming an important bottleneck in applications from data mining to serving web pages. However, such applications benefit from distributed solutions, for example multiple web servers. Such distributed systems frequently rely on efficient shared file systems.

To demonstrate the various hardware and software bottlenecks that normally constrain networked file system write performance, we compared SunOS 4.1.3 NFS file write performance in various hardware and software configurations. We compare write throughput because it directly measures file system use of the network rather than internal caching. The equipment and results are summarized in Figure 1. Horizontal lines are the throughputs of the disk and network. The “SAM-300 File System” line is the throughput of a UNIX fast file system mounted on the RAM disk used in the prototype system.

The bottlenecks removed are the synchronous server disk writes, the 10 Mb/sec Ethernet, and the commodity disk at the server. Each bar in Figure 1 corresponds to the removal of a system bottleneck:

- **Synch Ethernet:** NFS using synchronous disk writes which underuses both the network and disk bandwidth. NFS version 3 relaxes the synchronous write constraints[5]. The bottleneck in this configuration is the use of synchronous disk writes at the server.
- **Asynch Ethernet:** NFS using asynchronous disk writes at the server on a 10 Mb/sec Ethernet. The network bottlenecks this system.
- **Asynch ATOMIC:** NFS using asynchronous disk writes and the ATOMIC LAN. Replacing the Ethernet with the ATOMIC LAN improves the performance to roughly 20 Mb/sec, which is equivalent to local file system performance. The disk is the bottleneck in this system.
- **No Disk ATOMIC:** NFS exchanging the write data only, without updating the disk. Throughput is significantly below the TCP process to process and the SAM-300 local file system. throughput, which implies that NFS protocol throughput can be improved.

The experiment shown in Figure 1 led us to concentrate on improving the client transfer protocol implementation. The disparity between the TCP throughput and the NFS protocol bandwidth implies that the NFS protocol implementation can be improved; the SAM-300 file system is capable of supporting such throughput improvements. During the experiments the client CPU was completely utilized while the server utilization was less than 40%, implying that the bottleneck is at the client side. Although there are certainly interesting problems in engineering servers to scale, the client CPU bottleneck must be addressed to deliver higher network bandwidth to individual clients.

We studied the client implementation and improved its data translation behavior and its context switching behavior. The baseline system for these improvements was the NFS system using null writes across the ATOMIC LAN (the rightmost bar in Figure 1). These improvements resulted in a 30% performance increase. We also estimated the performance benefits of rewriting the XDR interface in the style of Macklem[6], i.e., inlining all function calls. Such If XDR overhead was completely removed by such inlining, throughput could approach TCP levels. We did not implement Macklem’s improvements, because we focused our efforts on evaluating new system performance enhancements.

To demonstrate that improvements to the protocol implementation can improve performance in a real system, we incorporated a high speed RAM disk into our server, and tested our improved protocol. Although the performance improvements were less than for disk-free operation, they were still substantial: an 18% write throughput increase. This completely utilizes server bandwidth, considering that the network

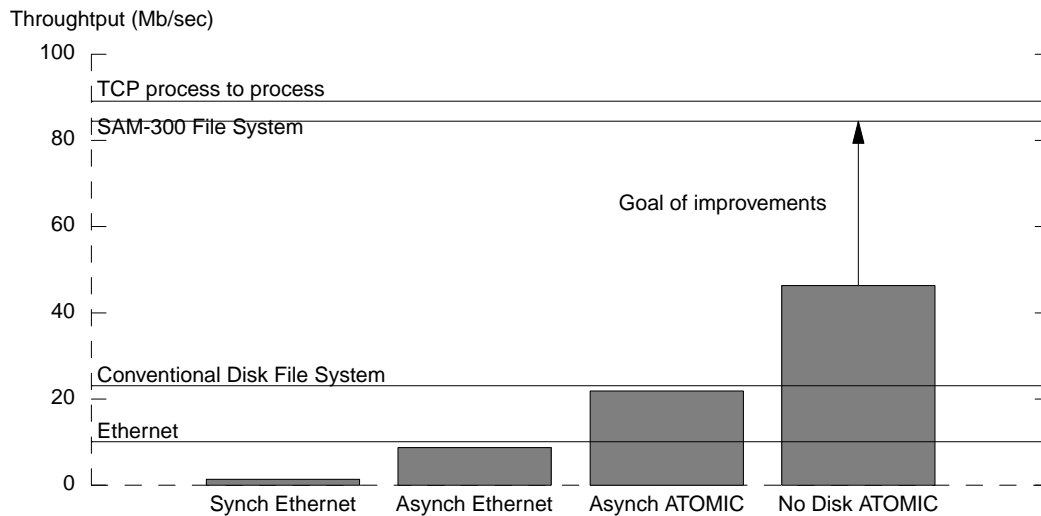


Figure 1: Bandwidth Comparison

All values are averages of 25 repetitions of writing a 64 megabyte (MB) file between two SPARCStation 20/71 workstations running SunOS 4.1.3 connected via a Myrinet. (Standard deviations are omitted, but are always less than 3% of the average.) Descriptions and discussions of the bars is in the text of the paper.

interface and SAM-300 share a bus.

The paper is structured as follows: Section 2 discusses our file system environment. Section 3 explores client implementation and design and describes the NFS optimizations. Section 4 discusses the performance of the prototype ATOMIC file server. Section 5 discusses related work, and Section 6 summarizes our conclusions.

2. The Testbed and Prototype Implementation

In order to provide the request/response file system communication to support frequent small file accesses, the ATOMIC file server was developed using Sun NFS as a starting point. NFS is also a good choice because it is an interface description, which allows our optimized clients to take advantages of improvements made to the local file systems of servers. For example a client using the modified NFS presented here would benefit from using an NFS server that implemented the embedded inode and grouping optimizations described by Ganger and Kaashoek[7]. Also, in order to have real users for our prototype system there were considerable benefits to maintaining the NFS semantics that are understood by our users.

As our experiment (in Figure 1) demonstrated, the protocol implementation bottlenecks of NFS are only visible when high-bandwidth hardware is in place. Our testbed, which is used for all the experiments described in Section 3, uses Sun SPARCStation 20/71 workstations for both clients and servers. The hosts are connected by Myricom's implementation of the ATOMIC LAN[3], Myrinet[8], which exhibits full duplex link rates of 640 Mb/sec, and a measured process-to-process TCP throughput in excess of 88 Mb/sec. NFS is not network-bound in this testbed. Null disk writes ensure that disk bottlenecks were

avoided; however, without disk writes the system is impractical.

The prototype ATOMIC file server, used for all the experiments described in Section 4, adds a SAM-300 RAM disk made by Texas Memory Systems to the server. The SAM-300 uses 512 MB of fast random access memory to emulate a disk, which eliminates delays due to the mechanical subsystems of rotating disks. The RAM disk makes the system practical for real use. When the SAM-300 is used as a local disk it can sustain a write throughput of 84.5 Mb/sec from process to disk using the SunOS UNIX file system. This throughput is considerably higher than our null write NFS performance.

3. Protocol Optimizations

Much of our work involves tuning the client implementations, which use more CPU cycles than servers. Our measurements indicated that clients were fully utilizing their CPU, so our improvement efforts concentrated on improving the useful work done by that CPU. Specifically we set out to avoid unnecessary data structure translation overheads and context switch overheads.

We also show that the XDR improvements suggested by Macklem[6] could improve NFS throughput to nearly TCP throughput, if they remove all XDR overhead. This is certainly an optimistic upper bound. Although we believe these optimizations to be effective, we did not implement them throughout our system because we focused our efforts on new improvements.

Improving data structure translations provided the bulk of our improvement, a 25% throughput increase. This implies that the primary bottleneck in the system is CPU-driven data copies. However, reducing context switches by creating a single-threaded client had a noticeable effect on throughput and offers an opportunity to tune the transfer implementation further.

3.1. Operating System Integration

Reducing the cost of a data type conversion between a representation used in the virtual memory (VM) system and one used in the networking subsystem increased NFS performance 25%. Translating between data structures used by the VM subsystem and the networking subsystems resulted in an extra copy. Removing the extraneous copy resulted in improved throughput.

SunOS implements an integrated file system cache/virtual memory system. When file data is written in SunOS, it is put into a VM page of memory and marked dirty. It is actually written out to secondary storage either when the paging system needs to reclaim the page, or when the file is flushed from the file cache explicitly, for example by a *close* or *sync* system call.

The SunOS networking system expects packets to be composed of *mbufs* and the translation from page to mbuf can require a data copy, which implies extra CPU overhead. Using cluster mbufs can avoid this overhead for NFS data packets. Cluster mbufs associate an mbuf structure with a region of memory, and can be used to link the VM page directly to the mbuf for encapsulation into an RPC packet. More aggressively using cluster mbufs to avoid data copies on outgoing data provided the majority of our performance improvement.

Using cluster mbufs for incoming data is difficult because packets must be parsed by the operating system in order to decide which should be stored in clusters. Using a cluster mbuf avoids a copy if the system can put incoming data into its final destination directly from the network device. Unless the packet can be efficiently parsed in network device memory, it must be copied to main memory so that the CPU can demultiplex it. Once that copy has been made, a second is unavoidable. Few systems have network devices that can support such early demultiplexing.

Cluster mbufs are a known optimization, but they are not commonly used on the client side of NFS. The extra CPU overhead incurred by using standard mbufs is masked by the network and disk bottlenecks in other environments. Use of cluster mbufs improves NFS performance significantly, yet it may adversely affect paging behavior, because the cluster's VM page remains pinned and dirty until the data in it has been

written to the NFS server. We noticed no such problems in our experiments, but our experiments were not exhaustive.

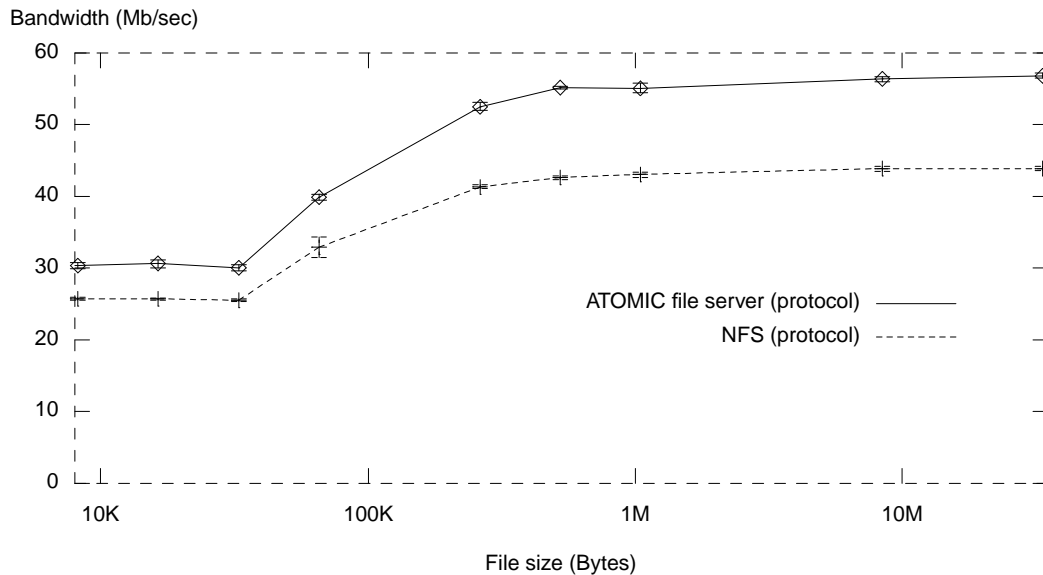


Figure 2: Performance Improvements

Averages of 25 writes various sized files. The environment described in Section 2 is used. “ATOMIC File Server” is SunOS NFS with the improvements of Section 3.1 and 3.2. applied, and with disk writes disabled at the server. “NFS” is SunOS NFS modified to not write to disk. Standard deviation error bars are plotted.

Figure 2 shows the 30% performance increase yielded by reducing the overhead of the data translation, and reducing context switches as described in Section 3.2. Although the improvement is most noticeable for large files, there is significant improvement for all sizes. Each point is the average of 25 trials of writing various sized files.

During this experiment, files were written in 32 KB blocks to minimize kernel crossing overhead, which accounts for the flat performance at low file sizes. All the points on the flat part of the curve were files that were written in one write. We used the relatively large 32 KB buffers in this experiment to concentrate on the effects of protocol overhead.

Data structure translations are often encountered in systems that integrate kernel subsystems, and must be carefully tuned for high performance. The data structure translation from VM data structure (page) to networking data structure (mbuf) is an example. As a matter of practice, such translations must be evaluated for efficiency. Another method to address this issue would be to use fewer, more unified data structures in kernel subsystems to reduce the number of translations and bottlenecks.

3.2. System Overhead – Context Switches

The current implementation of NFS uses process parallelism to pipeline data transfer requests, which results in considerable context switching. Using an alternate implementation of the data transfer protocol reduced context switches by more than 40%.

Optimizing for throughput requires having multiple data requests outstanding to fill the bandwidth-delay pipeline between the server and client. In NFS, data transfers are effected via Sun's RPC protocol[9], which, in SunOS 4.1.3, allows only one request per process to be pending. SunOS achieves request parallelism by using multiple processes, called *biod* processes, each running an instantiation of the RPC protocol. This is a common technique for file systems, especially NFS systems, and other transaction-based services. It is used because it simplifies the code for data transfer, and because many file system requests are not dependent on each other.

Context switching concerns aside, using a multiprocess implementation ties the number of outstanding requests to the number of running *biod* processes. In practice the maximum window size, i.e., the number of packets NFS can have outstanding, is fixed at boot time by the choice of how many *biod* processes to start. Reconfiguring an NFS client to use fewer *biod* processes in response to a change in network or server state is rarely, if ever, done. Although it is possible to vary the number of *biod*s in use dynamically, doing so requires either the coordination of running *biod*s or the addition of a *biod* controller process, either of which adds complexity to the design. In our single-threaded system, the number of outstanding requests allowed, or window size, is a parameter, which can be easily manipulated dynamically.

To reduce the context switching overhead of using multiple processes, we modified the implementation of Sun RPC to run in a single process and allow multiple requests to be outstanding. Creating such a data transfer mechanism in SunOS was straightforward, because Sun RPC is designed to allow such an implementation[9]. The replacement protocol sends and receives standard Sun RPC messages, using the transaction ID to associate requests with responses. NFS was modified to use our single-process RPC implementation instead of *biod* processes.

When NFS uses the single-threaded RPC, the process that is requesting the read or write sends the message, caches information about response handling, and returns, rather than dispatching a *biod* process to send it. A single process is responsible for receiving all the responses, resending lost requests, and freeing or filling VM pages. Context switches are saved when a client sends a request, because there is no intermediate step of communicating the request to a *biod* process and dispatching it. When a client receives a response fewer context switches are incurred as well, because the receiver process may remain active to receive several messages, each of which would have required a context switch otherwise.

Converting multiple processes to a single process is similar in spirit and implementation to the use of continuations in Mach, and related systems[10]. The state of each incomplete NFS transaction is kept in the single process explicitly as a continuation rather than implicitly in the process stack. As Draves et al. point out, this reduces the code size of our implementation in addition to reducing the number of context switches. However, we only optimize the *biod* execution; there are no stack hand-offs.

We compared the single-threaded implementation to the multiprocess (*biod*) implementation across a range of window sizes. Table 1 summarizes the data for the transfer of a 32 MB file varying the window size. The systems do not include the copy avoidance described in Section 3.1.

The single-threaded system shows a 4% improvement compared to the best *biod* throughput, and a 40% reduction in context switching. The values compared are circled in the table. As the window gets larger, the single-threaded receiving process can process more packets per context switch.

Converting to a single-threaded model enables further enhancements. For example, the number of outstanding requests is now a variable in a single process rather than a function of how many *biod* processes are running. This could allow the ATOMIC file server to react dynamically to changing network state more simply than a multiprocess NFS implementation, but this is not implemented yet.

3.3. Data Encapsulation and Portability

Macklem has shown that XDR implementations in the kernel are often a significant source of overhead[6]. XDR is a standard data encapsulation method used to communicate between servers and clients running on different hardware[4]. He observed significant increases in throughput of an NFS system

Window Size	Biod		Single Thread	
	Throughput (Mb/sec)	Context Switches	Throughput (Mb/sec)	Context Switches
3	41.0	9302	47.5	4951
4	42.5	8587	44.5	2529
5	44.1	9107	44.2	2226
6	45.1	8586	44.6	2242
7	45.3	9450	44.9	2227
8	44.0	9469	44.7	2220

Table 1: Summary of Context Switch reductions

Averages of 25 writes of a 32 MB file using various window sizes (or numbers of bios). The environment described in Section 2 was used. “Single thread” is the single-threaded implementation, and “NFS” is SunOS NFS. Both implementations have disk accesses disabled at the server. Standard deviations are omitted, but were on the order of tenths of megabits/sec or tens of context switches. Circled values are comparison values.

running under the 4.3BSD Reno system after replacing the subroutine calls that implemented XDR conversion with macros. Rather than repeat his work here, we attempt to estimate the impact that such an improvement would have on the ATOMIC file server. Although we estimate the impact of this improvement, we focused on in-kernel implementations of the new improvements described in Sections 3.1 and 3.2.

We compare user level programs that model NFS write exchanges, but with different data encapsulation mechanisms. One version encapsulates the data via XDR routines, the other by simpler network byte ordering routines. The simpler encapsulation is more efficient due to avoiding the overhead of the function calls to the XDR routines and extra data copies therein, as predicted by Macklem. The functionality of XDR is provided by both. One reason the user level Sun RPC routines have a higher throughput than the NFS protocol throughput is that they do not marshal their data to mbufs as the kernel routines do.

Figure 3 shows the result of an experiment in which 64 MB of data is transferred. Values are reported for multiprocess pipelined versions that mimic NFS biod processes. We compare process pipelining to process pipelining rather than using a single-threaded implementation because we want to isolate XDR overhead. The lightweight encapsulation is capable of matching TCP data transfer speeds. In fact, the effect of multiprocess transfer makes the aggregate throughput of the 8 processes higher than that of a single process using TCP.

This experiment suggests that careful implementation of the data translation routines could boost throughput considerably. Although we did not rewrite the kernel as Macklem did, we have shown that user level processes can be optimized to deliver considerably better raw throughput than XDR provides without loss of functionality. The experiment is encouraging, but idealistic. The user-level processes need not interact with other kernel systems; implementations in the kernel may not show the same improvement.

3.4. Summary of Client Issues

Our experiments on NFS have demonstrated the tangible performance benefits of a protocol implementation that is sensitive to processor utilization. Using single-threaded protocol processing is more appropriate than using heavyweight multiprocess pipelining, and that avoiding conversion between incompatible kernel data structures increases NFS throughput. These improvements increased NFS protocol performance by 30% and there is evidence that it can be combined with existing work, such as Macklem’s macro implementation of XDR to improve performance further. The performance improvement from each optimization is shown in Table 2.

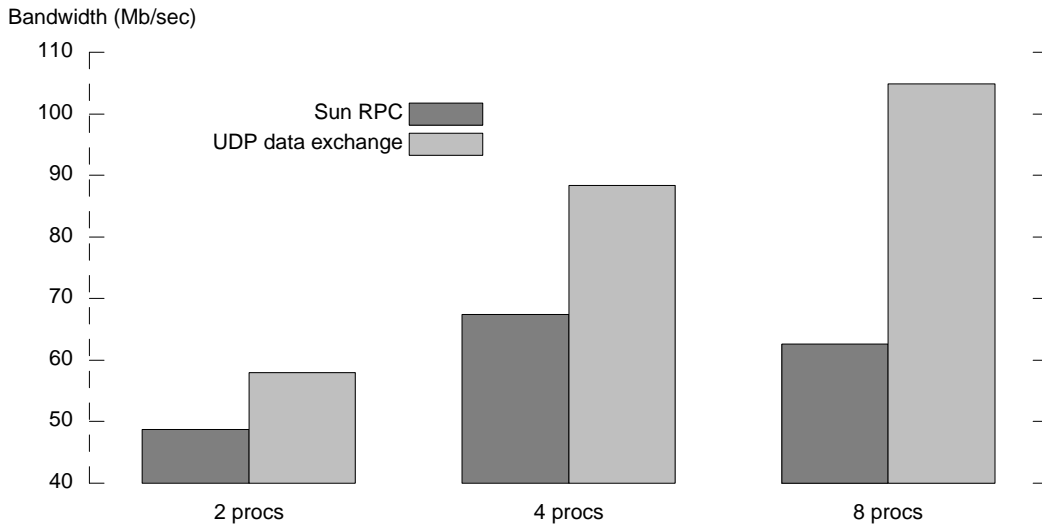


Figure 3: XDR vs. simple data encapsulation

Averages of 25 simulated 64 MB file writes. Standard deviations are omitted, but are no more than 3% of the mean, except for the 8 process custom application which had an 18% std. dev. The larger standard deviation is because that application had noticeable losses on some runs, which necessitated retransmissions. No disks are used. User level Sun RPC and custom UDP reliable data transfer are used.

Optimization Name	Throughput (32 MB file, no disk writes)
SunOS Implementation	45.3
Context Switch Avoidance	47.5
Copy Avoidance (and context switch avoidance)	56.9

Table 2: Impact of optimization

Because copy avoidance is the largest factor in improving performance, it implies that more system time is spent in copying data than in context switching. However, using single-process RPC has benefits beyond context switch reduction. Code size is reduced and collecting the multiple process state into one process encourages optimization of that state, e.g. better flow control.

This section has described improvements to the implementation of the file system data transfer mechanism. Because the section is concerned with improving the protocol implementation, no disk was used, which results in a system of only academic interest. The following section discusses a prototype that includes a RAM disk that can take advantage of the protocol improvements discussed here.

4. Implementation with Storage

To demonstrate that the protocol changes translate to real performance improvements in the prototype file server, we compare its throughput to SunOS NFS with asynchronous disk writes on the same hardware. The prototype system consists of the code described in Section 3 and a server equipped with a SAM-300. Note that in this section all writes at the server are asynchronous including those of the SunOS NFS. Unless asynchronous writes are implemented, as in NFS Version 3[5], protocol performance issues are moot, because the system performance will be limited by the disk's synchronous write performance, as seen in Figure 1.

A direct comparison of the ATOMIC file server and an asynchronous SunOS NFS server is shown in insert_ref f write . Both systems used identical hardware, including the SAM-300. This experiment directly measures the improvements in the NFS software in a usable file system. The ATOMIC file server outperforms unmodified NFS across the entire range of file sizes from a few kilobytes to multi-megabyte files, which confirms our protocol results.

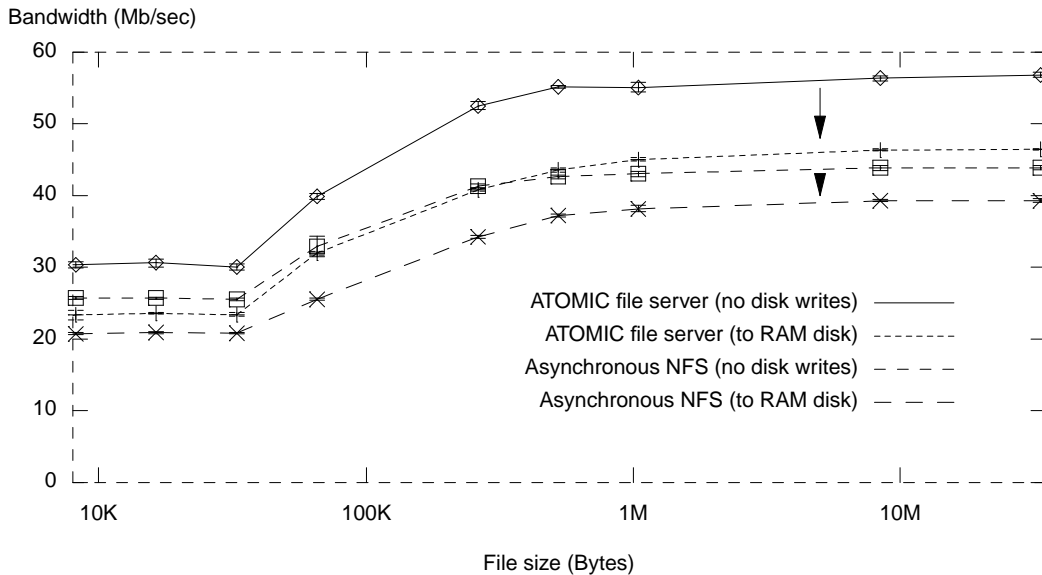


Figure 4: Write throughputs for the ATOMIC file server and SunOS NFS.

Protocol bandwidth is the bandwidth of the system when write requests are discarded at the server. Standard deviation error bars are plotted.

This difference in throughputs between NFS without disk writes NFS the SAM-300 is due to interactions between the networking subsystem and the file system. For example, writing files requires synchronous updates of metadata even if the data writes are asynchronous, which can stall the data transfer pipeline. Transferring data from mbufs to file cache blocks requires work by the server, which is another example of how data structure translation issues arise. The SAM controller and the ATOMIC network interface card share the same I/O bus in our file server, which increases the overhead of arbitration for that bus, as both interfaces are trying to transfer data simultaneously.

The ATOMIC file server is limited by the bandwidth available at the server. To demonstrate this we compared a well-tuned TCP data transfer program to the ATOMIC file server on the same hardware. The TCP program transfers data between processes at nearly 90 Mb/sec, but when this data was written to a file

by the receiving process the throughput approaches the ATOMIC file server throughput. Results are reported in Figure 5.

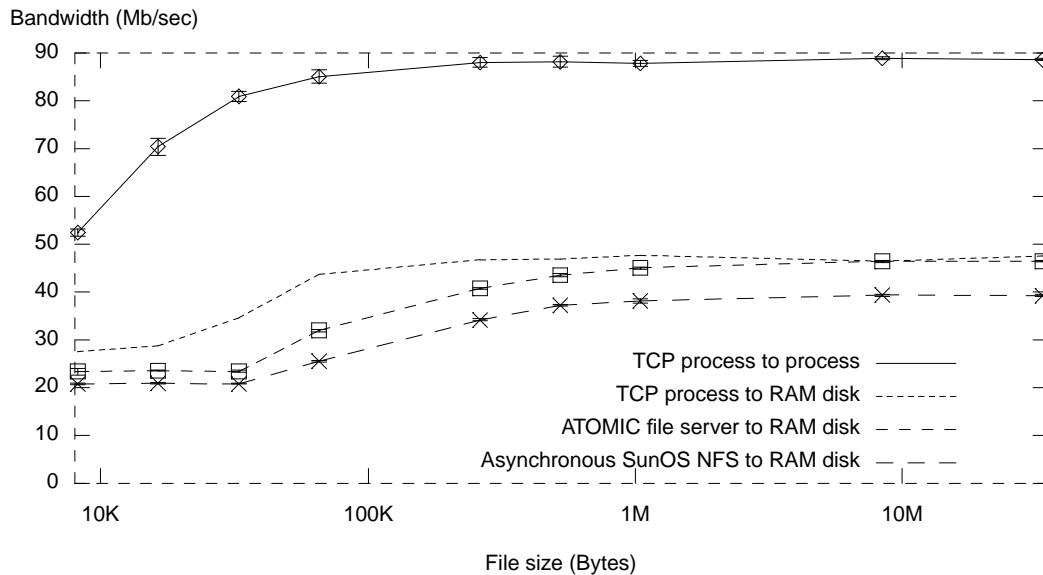


Figure 5: TCP throughput vs. ATOMIC file server throughput

TCP protocol is a process to process data transfer that discards data, TCP to SAM writes the received file to the SAM as it is received, Modified NFS and NFS write the file to SAM. One TCP process is used, 4 biod processes are used by NFS and 4 requests are allowed to be outstanding by the ATOMIC file system. Standard deviation error bars are plotted.

Both the ATOMIC file system and the TCP file transfer fully utilize the system. The network interface and the SAM-300 are both attached to the SBus, and provide approximately 85 Mb/sec each. However, file data must traverse that bus twice, so the approximately 45 MB/sec represents a full utilization of the bandwidth available to these peripherals.

We believe that the TCP file transfer exhibits higher throughput at small file sizes because it does not have the overhead of caching within the kernel. As described in Section 3.1, NFS does not immediately begin sending data to the server when a file is written. The block is cached first and then forced out on page reclamation or close operation. TCP data does not have this initial stall in its pipeline. As the stall becomes less significant, the throughputs converge, indicating that over the long term both systems fully utilize server bandwidth.

5. Related Work

The ATOMIC project is working to scale a traditional distributed computing environment along the bandwidth axis, in which small file accesses are common. RAID systems are another way to achieve high throughput by striping data across clusters of disks[11]. They are inapplicable to our environment because they perform best under the sustained load facilitated by large files, rather than the bursty load of workstation users. Furthermore, accessing a RAID across a network entails all the software problems addressed by the ATOMIC file server.

The ratio of CPU power to network bandwidth in our environment requires the ATOMIC file server to make conservative choices in data transfer systems. Rather than the transaction-based approach to data transfer that the ATOMIC file server uses, some systems advocate a streaming data protocol such as TCP to transfer files. This technique, used by NFS/bds, has shown promise, especially on machines using very powerful CPUs[12]. However, the added overhead of converting the data from an undelimited byte stream to operating system structures and realigning data in the absence of specialized hardware is CPU intensive at the client. The work described in this paper found existing data translations to be in need of optimization, so adding additional translations seems counterproductive.

The ATOMIC file server addresses the problem of utilizing network bandwidth. Other systems that use the same request/response communication model concentrated their efforts on adding functionality to the file system, and optimizing other parameters, for example improving caching or availability. Such systems include NFS, AFS, the Sprite file system[13], the Spring file system[14], and Ficus[15]. The ATOMIC file server optimizes such a communication model for high bandwidth access.

Scaling for throughput has been the goal of some file systems, but not in the conventional workstation environment. The Extensible File System(ELFS)[16], NFS/bds[12], and numerous RAID-based systems[11] are examples of high bandwidth systems. These systems are in use in supercomputer environments, which have less bursty workloads, and hosts with more powerful CPUs. Because most scientific visualization and network of workstations environments exhibit architectures and workloads that are very similar to supercomputing environments, they are equivalent for the purpose of this discussion. Zebra[17] and the xFS serverless file system[18] have addressed the issue of high throughput in a cluster of workstations environment. Such systems are optimized for the demands of supercomputing applications that require access to large data files rather than the small file accesses of workstation users.

The ATOMIC file server is based on an implementation of the NFS Version 2 specification[2], which has been superseded by the NFS Version 3 specification[5]. The most important change with respect to file transfer bandwidth is that asynchronous writes are explicitly supported by Version 3. This is a prerequisite for any high-bandwidth system based on NFS. In most other aspects, the techniques applied to the SunOS NFS Version 2 implementation are applicable to any similarly structured NFS Version 3 implementation. Such implementations should consider the benefits of reducing data structure translations and context switches. Implementations should also also implement data encapsulation systems efficiently.

6. Conclusions

We have described several optimizations to a networked file system designed to take advantage of emerging high-bandwidth LANs. These optimizations have improved Sun NFS protocol throughput by 30%, and when applied to a full system fully utilize server bandwidth. Prototype performance improves by 18% compared to SunOS NFS using asynchronous writes on the same hardware.

Network clients were tuned to avoid extraneous data type translations and context switches. Reducing data type translations resulted in fewer data copies which improved performance by 25% when disk writes are ignored. Reducing the number of context switches by converting the implementation from a multiprocess model to a single-process model improved performance another 4%. The single-process model also offers new opportunities for dynamic control of file system bandwidth.

We also showed that there is promise in inlining data encapsulations in the ATOMIC environment, as proposed by Macklem[6]. Streamlined data encapsulation improved data transfer speed to that of TCP.

The protocol enhancements were then tested in a functional system including a high-bandwidth RAM disk, and the system was found to use the full bandwidth available. The system provides common file system semantics while offering high-bandwidth remote data access. Although this work was carried out on an NFS system, the general principles of avoiding unnecessary data translations and process-based pipelining should be generally applicable to the implementation of other network file systems.

References

1. Russel Sandberg, David Goldberg, Steve Kleiman, Dan Walsh, and Bob Lyon, "Design and Implementation of the Sun Network File System," *Proceedings of the USENIX Conference*, pp. 119-130, USENIX (June 1985).
2. Sun Microsystems, Inc., "Network Filesystem Specification," *RFC-1094* (March 1, 1989).
3. Robert Felderman, Annette DeSchon, Danny Cohen, and Gregory Finn, "Atomic: A High Speed Local Communication Architecture," *Journal of High Speed Networks*, vol. 3, pp. 1-29 (1994).
4. R. Srinivasan, "XDR: External Data Representation Standard," *RFC-1832* (August 1995).
5. Sun Microsystems, Inc., *NFS: Network File System Version 3 Protocol Specification*, Sun Microsystems, Inc., Mountain View, CA (February 16, 1994).
6. R. Macklem, "Lessons Learned from Tuning the 4.3BSD Reno Implementation of the NFS Protocol," *Proceedings of the Winter USENIX Conference*, pp. 53-64, USENIX, Dallas, TX (January 1991).
7. Gregory R. Ganger and M. Franz Kaashoek, "Embedded Inodes and Explicit Grouping: Exploiting Disk Bandwidth for Small Files," *USENIX 1997 Annual Technical Conference*, pp. 1-18, USENIX, Anaheim, CA (January 6-10, 1997).
8. Myricom, Inc., Nannette J. Boden, Danny Cohen, Robert E. Felderman, Alan E. Kulawik, Charles L. Seitz, Jakov N. Selovic, and Wen-King Su, "Myrinet: A Gigabit-per-second Local Area Network," *IEEE Micro*, pp. 29-36, IEEE (February 1995).
9. Sun Microsystems, Inc., "Remote Procedure Call Specification," *RFC-1057* (June 1, 1988).
10. Richard P. Draves, Brian N. Bershad, Richard F. Rashid, and Randall W. Dean, "Using Continuations to Implement Thread Management and Communication in Operating Systems," *Proceedings of the 13th Symposium on Operating System Principles*, pp. 122-136, ACM (October 1991).
11. David A. Patterson, Garth Gibson, and Randy H. Katz, "A Case for Redundant Arrays of Inexpensive Disks (RAID)," *Proceedings of the ACM SIGMOD*, pp. 109-116 (June 1988).
12. Larry McVoy, *NFS/bds - NFS goes to the gym* (December 1995), available electronically from <http://reality.sgi.com/lm/talks/bds.ps>.
13. Michael N. Nelson, Brent B. Welch, and John K. Ousterhout, "Caching in the Sprite Network File System," *ACM Transactions on Operating Systems*, vol. 6, no. 1, pp. 134-154 (February 1988).
14. Michael N. Nelson and Yousef A. Khalidi, "Extensible File Systems in Spring," *Proceedings of the Fourteenth ACM Symposium on Operating Systems Principles*, pp. 1-14, ACM, Asheville, NC (December 1993).
15. Richard G. Guy, John S. Heidemann, Wai Mak, Thomas W. Page, Jr., Gerald J. Popek, and Dieter Rothmeier, "Implementation of the Ficus Replicated File System," *USENIX Conference Proceeding*, pp. 63-71, USENIX (June 1990).
16. John F. Karpovich, Andrew S. Grimshaw, and James C. French, "Extensible File Systems (ELFS): An Object-Oriented Approach to High Performance File I/O," *Proceedings of the Ninth Annual Conference on Object-Oriented Programming Systems, Languages and Applications* (October 1994).
17. J. Hartman and J. Osterhout, "The Zebra Striped Network File System," *Proceedings of the Fourteenth ACM Symposium on Operating Systems Principles*, pp. 29-36, ACM, Asheville, NC (December 1993).
18. Thomas E. Anderson, Michael D. Dahlin, Jeanna M. Neefe, David A. Patterson, Drew S. Roselli, and Randolph Y. Wang, "Serverless Network File Systems," *Proceedings of the 15th Symposium on Operating Systems Principles*, pp. 109-126, ACM, Copper Mountain Resort, Colorado (December 1995).