Network I/O with Trapeze

Jeffrey S. Chase, Darrell C. Anderson, Andrew J. Gallatin, Alvin R. Lebeck, and Kenneth G. Yocum[*]
Department of Computer Science
Duke University
{*chase, anderson, gallatin, alvy, grant*}*@cs.duke.edu*

Abstract

Recent gains in communication speeds motivate the design of network storage systems whose performance tracks the rapid advances in network technology rather than the slower rate of advances in disk technology. Viewing the network as the primary access path to I/O is an attractive approach to building incrementally scalable, cost-effective, and easy-to-administer storage systems that move data at network speeds.

This paper gives an overview of research on high-speed network storage in the Trapeze project. Our work is directed primarily at delivering gigabit-per-second performance for network storage access, using custom firmware for Myrinet networks, a lightweight messaging system optimized for block I/O traffic, and a new kernel storage layer incorporating network memory and parallel disks. Our current prototype is capable of client file access bandwidths approaching 100 MB/s, with network memory fetch latencies below 150μs for 8KB blocks.

1 Introduction

Storage access is a driving application for high-speed LAN interconnects. Over the next few years, new high-speed network standards — primarily Gigabit Ethernet — will consolidate an order-of-magnitude gain in LAN performance already achieved with specialized cluster interconnects such as Myrinet and SCI. Combined with faster I/O bus standards, these networks greatly expand the capacity of even inexpensive PCs to handle large amounts of data for scalable computing, network services, multimedia and visualization.

These gains in communication speed enable a new generation of network storage systems whose performance tracks the rapid advances in network technology rather than the slower rate of advances in disk technology. With gigabit-per-second networks, a fetch request for a faulted page or file block can complete up to two orders of magnitude faster from remote memory than from a local disk (assuming a seek). Moreover, a storage system built from disks distributed through the network (e.g., attached to dedicated servers [11, 12, 10], cooperating peers [3, 13], or the network itself [8]) can be made incrementally scalable, and can source and sink data to and from individual clients at network speeds.

The Trapeze project is an effort to harness the power of gigabit-per-second networks to "cheat" the disk I/O bottleneck for I/O-intensive applications. We use the network as the sole access path to external storage, pushing all disk storage out into the network. This network-centric approach to I/O views the client's file system and virtual memory system as extensions of the network protocol stack. The key elements of our approach are:

- **Emphasis on communication performance.** Our system is based on custom Myrinet firmware and a lightweight kernel-kernel messaging layer optimized for block I/O traffic. The firmware includes features for zero-copy block movement, and uses an adaptive message pipelining strategy to reduce block fetch latency while delivering high bandwidth under load.

- **Integration of network memory as an intermediate layer of the storage hierarchy.** The Trapeze project originated with communication support for a network memory service [6], which stresses network performance by removing disks from the critical path of I/O. We are investigating techniques to manage network memory as a distributed, low-overhead, "smart" file buffer cache between local memory and disks, to exploit its potential to mask disk access latencies.

[*]This work is supported by the National Science Foundation (CCR-96-24857, CDA-95-12356, and EIA-9870724) and equipment grants from Intel Corporation and Myricom. Anderson is supported by a U.S. Department of Education GAANN fellowship.

- **Parallel block-oriented I/O storage.** We are developing a new scalable storage layer, called Slice, that partitions file data and metadata across a collection of I/O servers. While the I/O nodes in our design could be network storage appliances, we have chosen to use generic PCs because they are cheap, fast, and programmable.

This paper is organized as follows. Section 2 gives a broad overview of the Trapeze project elements, with a focus on the features relevant to high-speed block I/O. Section 3 presents more detail on the adaptive message pipelining scheme implemented in the Trapeze/Myrinet firmware. Section 4 presents some experimental results showing the network storage access performance currently achievable with Slice and Trapeze. We conclude in Section 5.

2 Overview of Trapeze and Slice

The Trapeze messaging system consists of two components: a messaging library that is linked into the kernel or user programs, and a firmware program that runs on the Myrinet network interface card (NIC). The firmware defines the interface between the host CPU and the network device; it interprets commands issued by the host and masters DMA transactions to move data between host memory and the network link. The host accesses the network using the Trapeze library, which defines the lowest-level API for network communication. Since Myrinet firmware is customer-loadable, any Myrinet site can use Trapeze.

Trapeze was designed primarily to support fast kernel-to-kernel messaging alongside conventional TCP/IP networking. Figure 1 depicts the structure of our current prototype client based on FreeBSD 4.0. The Trapeze library is linked into the kernel along with a network device driver that interfaces to the TCP/IP protocol stack. Network storage access bypasses the TCP/IP stack, instead using *NetRPC*, a lightweight communication layer that supports an extended Remote Procedure Call (RPC) model optimized for block I/O traffic. Since copying overhead can consume a large share of CPU cycles at gigabit-per-second bandwidths, Trapeze is designed to allow copy-free data movement, which is supported by page-oriented buffering strategies in the socket layer, network device driver, and NetRPC.

We are experimenting with Slice, a new scalable network I/O service based on Trapeze. The current Slice prototype is implemented as a set of loadable kernel modules for FreeBSD. The client side consists of 3000 lines of code interposed as a stackable file system layer above the Network File System (NFS) protocol stack. This module intercepts read and write operations on file vnodes and redirects them to an array of block I/O servers using NetRPC. It incorporates a simple striping layer and cacheable block maps that track the location of blocks in the storage system. Name space operations are handled by a file manager using the NFS protocol, decoupling name space management (and access control) from block management. This structure is similar to other systems that use independent file managers, including Swift [4], Zebra [10], and Cheops/NASD [9]. To scale the file manager service, Slice uses a hashing scheme that partitions the name space across an array of file managers, implemented in a packet filter that redirects NFS requests to the appropriate server.

2.1 The Slice Block I/O Service

The Slice block I/O service is built from a collection of PCs, each with a handful of disks and a high-speed network interface. We call this approach to network storage "PCAD" (PC-attached disks), indicating an intermediate approach between network-attached disks (NASD) and conventional file servers (server-attached disks, or SAD). While the CMU NASD group has determined that SAD can add up to 80% to the cost of disk capacity [9], it is interesting to note that the cost of the CPU, memory, and network interface in PCAD is comparable to the price differential between IDE and SCSI storage today. Our current IDE PCAD nodes serve 88 GB of storage on four IBM DeskStar 22GXP drives at a cost under $60/GB, including a PC tower, a separate Promise Ultra/33 IDE channel for each drive, and a Myrinet NIC and switch port. With the right software, a collection of PCAD nodes can act as a unified network storage volume with incrementally scalable bandwidth and capacity, at a per-gigabyte cost equivalent to a medium-range raw SCSI disk system.[1] Moreover, the PCAD nodes feature a 450 MHz Pentium-III CPU and 256MB of DRAM, and are sharable on the network.

The chief drawback of the PCAD architecture is that the I/O bus in the storage nodes limits the number of disks that can be used effectively on each node, presenting a fundamental obstacle to lowering the price per gigabyte without also compromising the bandwidth per unit of capacity. Our current IDE/PCAD configurations use a single 32-bit 33 MHz PCI bus, which is capable of streaming data between the network and disks at 40 MB/s. Thus the PCAD/IDE network storage service delivers only about 30% of the bandwidth per gigabyte of capacity as the SCSI disks of equivalent cost. Even so, the bus bandwidth limitation is a cost issue rather than a fundamental limit to performance, since bandwidth can

[1]Seagate Barracuda 9GB drives were priced between $50/GB and $80/GB in April 1999, with an average price of $60/GB.

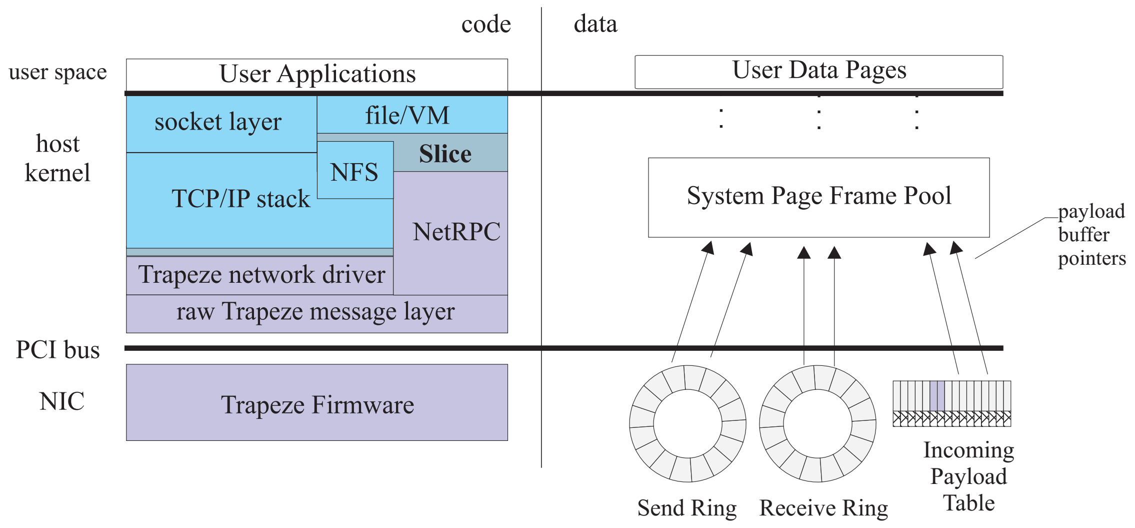


Figure 1: A view of the Slice/Trapeze prototype.

be expanded by adding more I/O nodes, and bus latencies are insignificant where disk accesses are involved.

In our Slice prototype, the block I/O servers run FreeBSD 4.0 kernels supplemented with a loadable module that maps incoming requests to collections of files in dedicated local file systems. Slice includes features that enable I/O servers to act as caches over NFS file servers, including tertiary storage servers or Internet gateways supporting the NFS protocol [2]. In other respects, the block I/O protocol is compatible with NASD, which is emerging as a promising storage architecture that would eliminate the I/O server bus bottleneck.

One benefit of PCAD I/O nodes is that they support flexible use of network memory as a shared high-speed I/O cache integrated with the storage service. Trapeze was originally designed as a messaging substrate for the Global Memory Service (GMS) [6], which supports remote paging and cooperative caching [5] of file blocks and virtual memory pages, unified at a low level of the operating system kernel. The GMS work showed significant benefits for network memory as a fast temporary backing store for virtual memory or scratch files. The Slice block I/O service retains the performance emphasis of the network memory orientation, and the basic protocol and mechanisms for moving, caching, and locating blocks are derived from GMS. We are investigating techniques for using the I/O server CPU to actively manage server memory as a prefetch buffer, using speculative prediction or hinting directives from clients [13].

2.2 Trapeze Messaging and NetRPC

Trapeze messages are short (128-byte) *control messages* with optional attached *payloads* typically containing application data not interpreted by the messaging system, e.g., file blocks, virtual memory pages, or TCP segments. The data structures in NIC memory include two message rings, one for sending and one for receiving. Each message ring is a circular producer/consumer array of 128-byte control message buffers and related state, shown in Figure 1. The host attaches a payload buffer to a message by placing its DMA address in a designated field of the control message header.

The Trapeze messaging system has several features useful for high-speed network storage access:

- **Separation of header and payload.** A Trapeze control message and its payload (if any) are sent as a single packet on the network, but they are handled separately by the message system, and the separation is preserved at the receiver. This enables the TCP/IP socket layer and NetRPC to avoid copying, e.g., by remapping aligned payload buffers. To simplify zero-copy block fetches, the NIC can demultiplex incoming payloads into a specific frame, based on a token in the message that indirects through an *incoming payload table* on the NIC.

- **Large MTUs with scatter/gather DMA.** Since Myrinet has no fixed maximum packet size (MTU), the maximum payload size of a Trapeze network is easily configurable. Trapeze supports scatter/gather DMA so that payloads may span multiple noncon-

Host	*fixed pipeline*	*store & forward*	*adaptive pipeline*
450 MHz Pentium-III	124μs / 77 MB/s	217μs / 110 MB/s	112μs / 110 MB/s
500 MHz Alpha Miata	129μs / 71 MB/s	236μs / 93 MB/s	119μs / 93 MB/s

Table 1: Latency and bandwidth of NIC DMA pipelining alternatives for 8KB payloads.

tiguous page frames. Scatter/gather is useful for deep prefetching and write bursts, reducing per-packet overheads for high-volume data access.

- **Adaptive message pipelining.** The Trapeze firmware adaptively pipelines DMA transfers on the I/O bus and network link to minimize the latency of I/O block transfers, while delivering peak bandwidth under load. Section 3 discusses adaptive message pipelining in more detail.

The NetRPC package based on Trapeze is derived from the original RPC package for the Global Memory Service (*gms_net*), which was extended to use Trapeze with zero-copy block handling and support for asynchronous prefetching at high bandwidth [1].

To complement the zero-copy features of Trapeze, the socket layer, TCP/IP driver, and NetRPC share a common pool of aligned network payload buffers allocated from the virtual memory page frame pool. Since FreeBSD exchanges file block buffers between the virtual memory page pool and the file cache, this allows unified buffering among the network, file, and VM systems. For example, NetRPC can send any virtual memory page or cached file block out to the network by attaching it as a payload to an outgoing message. Similarly, every incoming payload is deposited in an aligned physical frame that can mapped into a user process or hashed into the file cache or VM page cache. This unified buffering also enables the socket layer to reduce copying by remapping pages, which significantly reduces overheads for TCP streams [7].

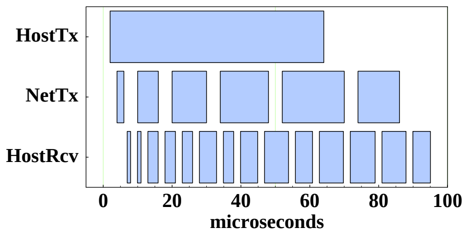
High-bandwidth network I/O requires support for asynchronous block operations for prefetching or write-behind. NFS clients typically support this asynchrony by handing off outgoing RPC calls to a system *I/O daemon* that can wait for RPC replies, allowing the user process that originated the request to continue. NetRPC supports a lower-overhead alternative using *nonblocking RPC*, in which the calling thread or process supplies a *continuation* procedure to be executed — typically from the receiver interrupt handler — when the reply arrives. The issuing thread may block at a later time, e.g., if it references a page that is marked in the I/O cache for a pending prefetch. In this case, the thread sleeps and is awakened directly from the receiver interrupt handler. Nonblocking RPCs are a simple extension of kernel facilities already in place for asynchronous I/O on disks; each network I/O operation applies to a buffer in the I/O cache, which acts as a convenient point for synchronizing with the operation or retrieving its status.

3 Balancing Latency and Bandwidth

From a network perspective, storage access presents challenges that are different from other driving applications of high-speed networks, such as parallel computing or streaming media. While small-message latency is important, server throughput and client I/O stall times are determined primarily by the latency and bandwidth of messages carrying file blocks or memory pages in the 4 KB to 16 KB range. The relative importance of latency and bandwidth varies with workload. A client issuing unpredicted fetch requests requires low latency; other clients may be bandwidth-limited due to multithreading, prefetching, or write-behind.

Reconciling these conflicting demands requires careful attention to data movement through the messaging system and network interface. One way to achieve high bandwidth is to use large transfers, reducing per-transfer overheads. On the other hand, a key technique for achieving low latency for large packets is to fragment each message and pipeline the fragments through the network, overlapping transfers on the network links and I/O buses [16, 14]. Since it is not possible to do both at once, systems must select which strategy to use. Table 1 shows the effect of this choice on Trapeze latency and bandwidth for 8KB payloads, which are typical of block I/O traffic. The first two columns show measured one-way latency and bandwidth using fixed-size 1408-byte DMA transfers and 8KB store-and-forward transfers. These experiments use raw Trapeze messaging over LANai-4 Myrinet NICs with firmware configured for each DMA policy. Fixed pipelining reduces latency by up to 45% relative to store-and-forward DMA through the NIC, but the resulting per-transfer overheads on the NIC and I/O bus reduce delivered bandwidth by up to 30%.

To balance latency and bandwidth, Trapeze uses an adaptive strategy that pipelines individual messages automatically for lowest latency, while dynamically adjusting the degree of pipelining to traffic patterns and congestion. The third column in Table 1 shows that this



```
do forever
    for each idle DMA sink in {NetTx, HostRcv}
        waiting = words awaiting transfer to sink
        if (waiting > MINPULSE)
            initiate transfer of waiting words to sink
    end
    for each idle DMA source in {HostTx, NetRcv}
        if (waiting transfer and buffer available)
            initiate transfer from source
    end
loop
```

Figure 2: Adaptive message pipelining policy and resulting pipeline transfers.

yields both low latency and high bandwidth. Adaptive message pipelining in Trapeze is implemented in the NIC firmware, eliminating host overheads for message fragmentation and reassembly.

Figure 2 outlines the message pipelining policy and the resulting overlapped transfers of a single 8KB packet across the sender's I/O bus, network link, and receiver's I/O bus. The basic function of the firmware running in each NIC is to move packet data from a source to a sink, in both sending and receiving directions. Data flows into the NIC from the source and accumulates in NIC buffers; the firmware ultimately moves the data to the sink by scheduling a transfer on the NIC DMA engine for the sink. When sending, the NIC's source is the host I/O bus (*hostTx*) and the sink is the network link (*netTx*). When receiving, the source is the network link (*netRcv*) and the sink is the I/O bus (*hostRcv*). The Trapeze firmware issues large transfers from each source as soon as data is available and there is sufficient buffer space to accept it. Each NIC makes independent choices about when to move data from its local buffers to its sinks.

The policy behind the Trapeze pipelining strategy is simple: if a sink is idle, initiate a transfer of all buffered data to the sink if and only if the amount of data exceeds a configurable threshhold (*minpulse*). This policy produces near-optimal pipeline schedules automatically because it naturally adapts to speed variations between the source and the sink. For example, if a fast source feeds a slow sink, data builds up in the NIC buffers behind the sink, triggering larger transfers through the bottleneck to reduce the total per-transfer overhead. Similarly, if a slow source feeds a fast sink, the policy produces a sequence of small transfers that use the idle sink bandwidth to reduce latency.

The adaptive message pipelining strategy falls back to larger transfers during bursts or network congestion, because buffer queues on the NICs allow the adaptive behavior to carry over to multiple packets headed for the same sink. Even if the speeds and overheads at each pipeline stage are evenly matched, the higher overhead of initial small transfers on the downstream links quickly causes data to build up in the buffers of the sending and receiving NICs, triggering larger transfers.

Figure 3 illustrates the adaptive pipelining behavior for a one-way burst of packets with 8KB payloads. This packet flow graph was generated from logs of DMA activity taken by an instrumented version of the Trapeze firmware on the sending and receiving NICs. The transfers for successive packets are shown in alternating shadings; all consecutive stripes with the same shading are from the same packet. The width of each stripe shows the duration of the transfer, measured by a cycle counter on the NIC. This duration is proportional to the transfer size in the absence of contention. Contention effects can be seen in the long first transfer on the sender's I/O bus, which results from the host CPU contending for the bus as it initiates send requests for the remaining packets.

Figure 3 shows that both the sending and receiving NICs automatically drop out of pipelining and fall back to full 8KB transfers about one millisecond into the packet burst. While the pipelining yields low latency for individual packets at low utilization, the adaptive behavior yields peak bandwidth for streams of packets. The policy is automatic and self-tuning, and requires no direction from the host software. Experiments have shown that the policy is robust, and responds well to a range of congestion conditions [15].

4 Performance

Our goal with Trapeze and Slice is to push the performance bounds for network storage systems using Myrinet and similar networks. Although our work with Slice is preliminary, our initial prototype shows the performance that can be achieved with network storage systems using today's technology and the right network support.

Figure 4 shows read and write bandwidths from disk for high-volume sequential file access through the FreeBSD *read* and *write* system call interface using the current Slice prototype. For these tests, the client was a DEC Miata (Personal Workstation 500au) with a 500 MHz Alpha 21164 CPU and a 32-bit 33 MHz PCI bus

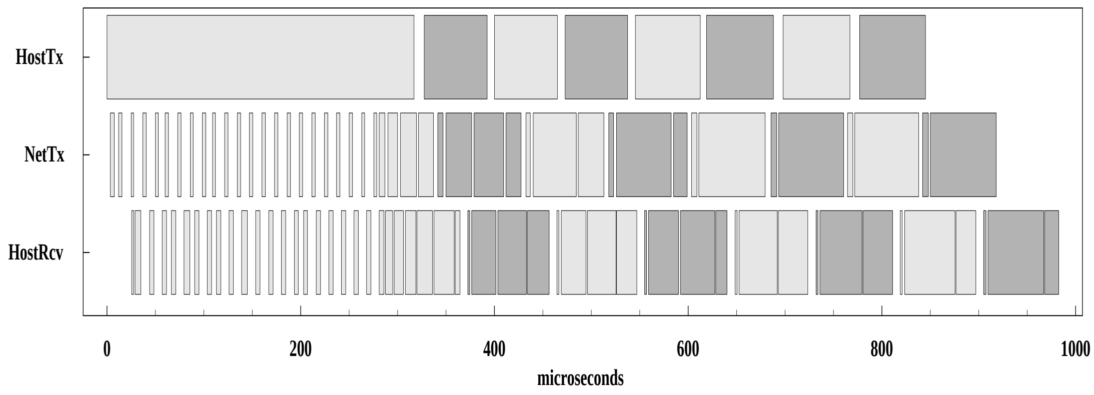


Figure 3: Adaptive message pipelining reverts to larger transfer sizes for a stream of 8K payloads. The fixed-size transfer at the start of each packet on *NetTx* and *HostRcv* is the control message data, which is always handled as a separate transfer. Control messages do not appear on *HostTx* because they are sent using programmed I/O rather than DMA on this platform (300 MHz Pentium-II/440LX).

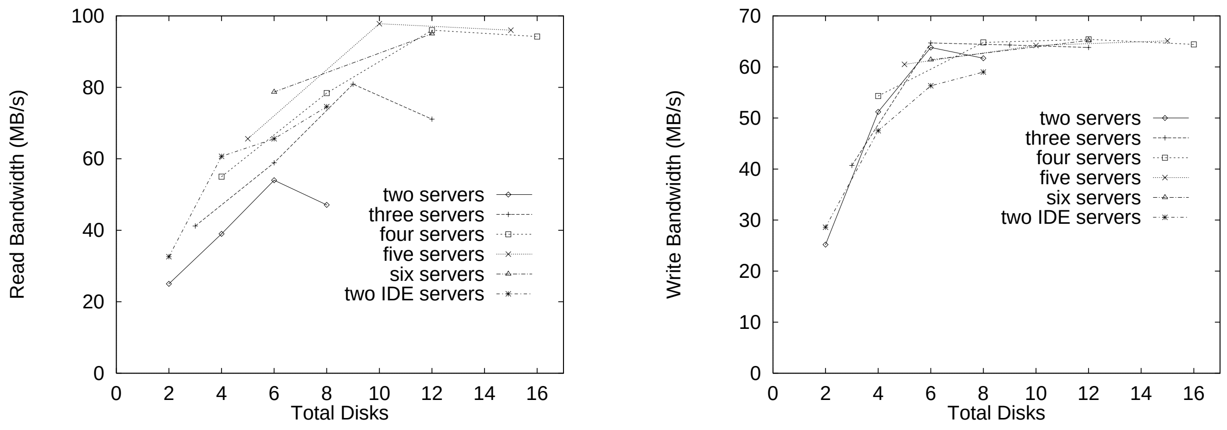


Figure 4: Single-client sequential disk read and write bandwidths with Slice/Trapeze.

using the Digital 21174 "Pyxis" chipset. The block I/O servers have a 450 MHz Pentium-III on an Asus P2B motherboard with an Intel 440BX chipset, and either of two disk configurations: (1) four Seagate Medalist disks on two separate Ultra-Wide SCSI channels, or (2) four IBM DeskStar 22GXP drives on separate Promise Ultra/33 IDE channels. All machines are equipped with Myricom LANai 4.1 SAN adapters, and run kernels built from the same FreeBSD 4.0 source pool. The Slice client uses a simple round-robin striping policy with a stripe grain of 32KB. The test program (*dd*) reads or writes 1.25 GB in 64K chunks, but it does not touch the data. Client and server I/O caches are flushed before each run.

Each line in Figure 4 shows the measured I/O bandwidth delivered to a single client using a fixed number of block I/O servers. The four points on each line represent the number of disks on each server; the x-axis gives the total number of disks used for each point. The peak write bandwidth is 66 MB/s; at this point the client CPU saturates due to copying at the system call layer. The peak read bandwidth is 97 MB/s. While reading at 97 MB/s the client CPU utilization is only 28%, since FreeBSD 4.0 avoids most copying for large reads by page remapping at the system call layer. In this configuration, an unpredicted 8KB page fault from I/O server memory completes in under 150 μs.

We have also experimented with TCP/IP communication using Trapeze. Recent point-to-point TCP bandwidth tests (*netperf*) yielded a peak bandwidth of 956 Mb/s through the socket interface on a pair of Compaq XP 1000 workstations using Myricom's LANai-5 NICs. These machines have a 500 MHz Alpha 21264 CPU and a 64-bit 33 MHz PCI bus with a Digital 21272 "Tsunami" chipset, and were running FreeBSD 4.0 augmented with page remapping for sockets [7].

5 Conclusion

This paper summarizes recent and current research in the Trapeze project at Duke University. The goal of our work has been to push the performance bounds for network storage access using Myrinet networks, by exploring new techniques and optimizations in the network interface and messaging system, and in the operating system kernel components for file systems and virtual memory. Key elements of our approach include:

- An adaptive message pipelining strategy implemented in custom firmware for Myrinet NICs. The Trapeze firmware combines low-latency transfers of I/O blocks with high bandwidth under load.
- A lightweight kernel-kernel RPC layer optimized for network I/O traffic. NetRPC allows zero-copy sends and receives directly from the I/O cache, and supports nonblocking RPCs with interrupt-time reply handling for high-bandwidth prefetching.
- A scalable storage system incorporating network memory and parallel disks on a collection of PC-based I/O servers. The system achieves high bandwidth and capacity by using many I/O nodes in tandem; it can scale incrementally with demand by adding more I/O nodes to the network.

Slice uses Trapeze and NetRPC to access network storage at speeds close to the limit of the network and client I/O bus. We expect that this level of performance can also be achieved with new network standards that are rapidly gaining commodity status, such as Gigabit Ethernet.

References

[1] D. Anderson, J. S. Chase, S. Gadde, A. J. Gallatin, K. G. Yocum, and M. J. Feeley. Cheating the I/O bottleneck: Network storage with Trapeze/Myrinet. In *1998 Usenix Technical Conference*, June 1998.

[2] D. C. Anderson, K. G. Yocum, and J. S. Chase. A case for buffer servers. In *IEEE Workshop on Hot Topics in Operating Systems (HOTOS)*, Apr. 1999.

[3] T. Anderson, M. Dahlin, J. Neefe, D. Patterson, D. Roselli, and R. Wang. Serverless network file systems. In *Proceedings of the ACM Symposium on Operating Systems Principles*, pages 109–126, Dec. 1995.

[4] L.-F. Cabrera and D. D. E. Long. Swift: Using distributed disk striping to provide high I/O data rates. *Computing Systems*, 4(4):405–436, Fall 1991.

[5] M. D. Dahlin, R. Y. Wang, and T. E. Anderson. Cooperative caching: Using remote client memory to improve file system performance. In *Proceedings of the First Symposium on Operating System Design and Implementation*, pages 267–280, Nov. 1994.

[6] M. J. Feeley, W. E. Morgan, F. H. Pighin, A. R. Karlin, and H. M. Levy. Implementing global memory management in a workstation cluster. In *Proceedings of the Fifteenth ACM Symposium on Operating Systems Principles*, 1995.

[7] A. J. Gallatin, J. S. Chase, and K. G. Yocum. Trapeze/IP:TCP/IP at near-gigabit speeds. In *1999 Usenix Technical Conference (Freenix track)*, June 1999.

[8] G. A. Gibson, D. F. Nagle, K. Amiri, F. W. Chang, E. M. Feinberg, H. Gobioff, C. Lee, B. Ozceri, E. Riedel, D. Rochberg, and J. Zelenka. File server scaling with network-attached secure disks. In *Proceedings of ACM International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS 97)*, June 1997.

[9] G. A. Gibson, D. F. Nagle, K. Amiri, F. W. Chang, H. Gobioff, C. Hardin, E. Riedel, D. Rochberg, and J. Zelenka. A cost-effective, high-bandwidth storage architecture. In *Proceedings of the Eight Conference on Architectural Support for Programming Languages and Operating Systems*, Oct. 1998.

[10] J. H. Hartman and J. K. Ousterhout. The Zebra striped network file system. In *Proceedings of the Fourteenth ACM Symposium on Operating Systems Principles*, pages 29–43, 1993.

[11] E. K. Lee and C. A. Thekkath. Petal: Distributed virtual disks. In *Proceedings of the Seventh International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 84–92, Cambridge, MA, Oct. 1996.

[12] C. A. Thekkath, T. Mann, and E. K. Lee. Frangipani: A scalable distributed file system. In *Proceedings of the Sixteenth ACM Symposium on Operating System Principles (SOSP)*, Oct. 1997.

[13] G. M. Voelker, E. J. Anderson, T. Kimbrel, M. J. Feeley, J. S. Chase, A. R. Karlin, and H. M. Levy. Implementing cooperative prefetching and caching in a globally-managed memory system. In *Proceedings of ACM International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS '98)*, June 1998.

[14] R. Y. Wang, A. Krishnamurthy, R. P. Martin, T. E. Anderson, and D. E. Culler. Modeling and optimizing communication pipelines. In *Proceedings of ACM International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS 98)*, June 1998.

[15] K. G. Yocum, D. C. Anderson, J. S. Chase, S. Gadde, A. J. Gallatin, and A. R. Lebeck. Adaptive message pipelining for network memory and network storage. Technical Report CS-1998-10, Duke University Department of Computer Science, Apr. 1998.

[16] K. G. Yocum, J. S. Chase, A. J. Gallatin, and A. R. Lebeck. Cut-through delivery in Trapeze: An exercise in low-latency messaging. In *Sixth IEEE International Symposium on High Performance Distributed Computing (HPDC-6)*, pages 243–252, Aug. 1997.