The following paper was originally published in the
Proceedings of the USENIX Annual Technical Conference (NO 98)
New Orleans, Louisiana, June 1998

Cheating the I/O Bottleneck: Network Storage with Trapeze/Myrinet

Darrell C. Anderson, Jeffrey S. Chase, Syam Gadde, Andrew J. Gallatin, and Kenneth G. Yocum,
Duke University
Michael J. Feeley,
University of British Columbia

Cheating the I/O Bottleneck: Network Storage with Trapeze/Myrinet

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

Michael J. Feeley
Department of Computer Science
University of British Columbia
feeley@cs.ubc.ca

Abstract

Recent advances in I/O bus structures (e.g., PCI), high-speed networks, and fast, cheap disks have significantly expanded the I/O capacity of desktop-class systems. This paper describes a messaging system designed to deliver the potential of these advances for network storage systems including cluster file systems and network memory. We describe *gms_net*, an RPC-like kernel-kernel messaging system based on Trapeze, a new firmware program for Myrinet network interfaces. We show how the communication features of Trapeze and *gms_net* are used by the Global Memory Service (GMS), a kernel-based network memory system.

The paper focuses on support for zero-copy page migration in GMS/Trapeze using two RPC variants important for peer-peer distributed services: (1) *delegated RPC* in which a request is delegated to a third party, and (2) *nonblocking RPC* in which replies are processed from the Trapeze receive interrupt handler. We present measurements of sequential file access from network memory in the GMS/Trapeze prototype on a Myrinet/Alpha cluster, showing the bandwidth effects of file system interfaces and communication choices. GMS/Trapeze delivers a peak read bandwidth of 96 MB/s using memory-mapped file I/O.

1 Introduction

Two recent hardware advances boost the potential of cluster computing: switched cluster interconnects that can carry 1Gb/s or more of point-to-point bandwidth, and high-quality PCI bus implementations that can handle data streams at gigabit speeds. We are developing system facilities to realize the potential for high-speed data transfer over Myricom's 1.28 Gb/s Myrinet LAN [2], and harness it for cluster file systems, network memory systems, and other distributed OS services that cooperatively share data across the cluster. Our broad goal is to use the power of the network to "cheat" the I/O bottleneck for data-intensive computing on workstation clusters.

This paper describes use of the Trapeze messaging system [27, 5] for high-speed data transfer in a network memory system, the Global Memory Service (GMS) [14, 18]. Trapeze is a firmware program for Myrinet/PCI adapters, and an associated messaging library for DEC AlphaStations running Digital Unix 4.0 and Intel platforms running FreeBSD 2.2. Trapeze communication delivers the performance of the underlying I/O bus hardware, balancing low latency with high bandwidth. Since the Myrinet firmware is customer-loadable, any Myrinet network site with PCI-based machines can use Trapeze.

GMS [14] is a Unix kernel facility that manages the memories of cluster nodes as a shared, distributed page cache. GMS supports remote paging [8, 15] and cooperative caching [10] of file blocks and virtual memory pages, unified at a low level of the Digital Unix 4.0 kernel (a FreeBSD port is in progress). The purpose of GMS is to exploit high-speed networks to improve performance of data-intensive workloads by replacing disk activity with memory-memory transfers across the network whenever possible. The GMS mechanisms manage the movement of VM pages and file blocks between each node's *local page cache* — the file buffer cache and the set of resident virtual pages — and the network memory *global page cache*.

This paper deals with the communication mechanisms and network performance of GMS systems using Trapeze/Myrinet, with particular focus on the support for zero-copy read-ahead and write-behind of sequentially accessed files. Cluster file systems that stripe data across multiple servers are typically limited by

[*]This work is supported by the National Science Foundation under grants CCR-96-24857 and CDA-95-12356, equipment grants from Intel Corporation and Myricom, and a software grant from the Open Group.

the bandwidth of the network and communication system [23, 16, 1]. We measure synthetic bandwidth tests that access files in network memory, in order to determine the maximum bandwidth achievable through the file system interface by any network storage system using Trapeze. The current GMS/Trapeze prototype can read files from network memory at 96 MB/s on an AlphaStation/Myrinet network. Since these speeds approach the physical limits of the hardware, unnecessary overheads (e.g., copying) can have significant effects on performance. These overheads can occur in the file access interface as well as in the messaging system. We evaluate three file access interfaces, including two that use the Unix *mmap* system call to eliminate copying.

Central to GMS is an RPC-like messaging facility (*gms_net*) that works with the Trapeze interface to support the messaging patterns and block migration traffic characteristic of GMS and other network storage services. This includes a mix of asynchronous and request/response messaging (RPC) that is *peer-to-peer* in the sense that each "client" may also act as a "server". The support for RPC includes two variants important for network storage: (1) *delegated RPCs* in which requests are delegated to third parties, and (2) *nonblocking RPC* in which the replies are processed by *continuation* procedures executing from an interrupt handler. These features are important for peer-to-peer network storage services: the first supports directory lookups for fetched data, and the second supports lightweight asynchronous calls, which are useful for prefetching. When using these features, GMS and *gms_net* cooperate with Trapeze to unify buffering of migrated pages, eliminating all page copies by sending and receiving directly from the file buffer cache and local VM page cache.

This paper is organized as follows. Section 2 gives an overview of the Trapeze network interface and the features relevant to GMS communication. Section 3 deals with the *gms_net* messaging layer for Trapeze, focusing on the RPC variants and zero-copy handling of page transfers. Section 4 presents performance results from the GMS/Trapeze prototype. We conclude in Section 5.

2 High-Speed Data Transfer with Trapeze

The Trapeze messaging system consists of two components: a messaging library that is linked into programs using the package, and a firmware program that runs on the Myrinet network interface card (NIC). The Trapeze firmware and the host interact by exchanging commands and data through a block of memory on the NIC, which is addressable in the host's physical address space using programmed I/O. The firmware defines the interface between the host CPU and the network device; it interprets commands issued by the host and controls the movement of data between the host and the network link. The host accesses the network using macros and procedures in the Trapeze library, which defines the lowest level API for network communication across the Myrinet.

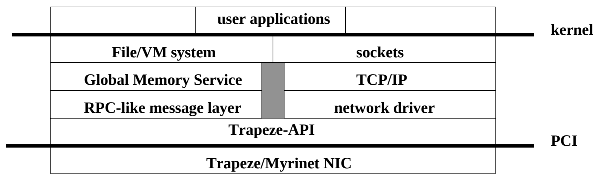

Figure 1: Using Trapeze for TCP/IP and for kernel-kernel messaging for network memory.

Like other network interfaces based on Myrinet (e.g., Hamlyn [4], VMMC-2 [13], Active Messages [9], FM [21]), Trapeze can be used as a memory-mapped network interface for user applications, e.g., parallel programs. However, Trapeze was designed primarily to support fast kernel-to-kernel messaging alongside conventional TCP/IP networking. The Trapeze distribution includes a network device driver that allows the native TCP/IP protocol stack to use a Trapeze network alongside the *gms_net* layer. Figure 1 depicts this structure. The kernel-to-kernel messaging layer is intended for GMS and other services that assume mutually trusting kernels.

2.1 Trapeze Overview

Trapeze messages are short *control messages* (maximum 128 bytes) with optional attached *payloads* typically containing application data not interpreted by the message system, e.g., a file block, a virtual memory page, or a TCP segment. Each message can have at most one payload attached to it. Separation of control messages and bulk data transfer is common to a large number of messaging systems since the V system [6].

A Trapeze control message and its payload (if any) are sent as a single packet on the network. Since Myrinet has no fixed maximum packet size (MTU), the maximum payload size of a Trapeze network is configurable, and is typically set to the virtual memory page size (4K or 8K). The Trapeze MTU is the maximum control message size plus the payload size.

Payloads are sent and received using DMA to/from aligned buffers residing anywhere in host memory. The host attaches a payload to an outgoing message using a Trapeze macro that stores the payload's DMA address and length into designated fields of the send ring entry. On the receiving side, Trapeze deposits the payload into a host memory buffer before delivering the control message.

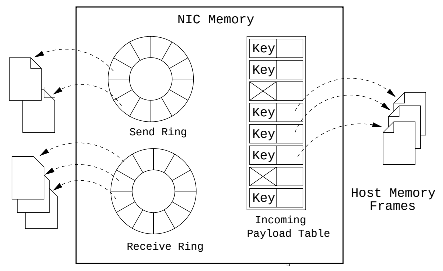

Figure 2: NIC Memory Structures for a Trapeze endpoint.

The data structures in NIC memory include an *endpoint* structure shared with the host. A Trapeze endpoint (shown in Figure 2) includes two message rings, one for sending and one for receiving. Each message ring is a circular array of 128-byte control message buffers and related state, managed as a producer/consumer queue. From the perspective of a host CPU, the NIC produces incoming messages in the receive ring and consumes outgoing messages in the send ring. The host sends a message by forming it in the next free send ring entry and setting a bit to indicate that the message is ready to send. When a message arrives from the network, the firmware deposits it into the next free receive ring entry, sets a bit to inform the host that the message is ready to consume, and optionally signals the host with an interrupt.

Handling of incoming messages is interrupt-driven when Trapeze is used from within the kernel. Each kernel protocol module using Trapeze (i.e., *gms_net* and the IP network driver) registers a receiver interrupt handler upcalled from the Trapeze interrupt handler.

Trapeze is designed to optimize handling of payloads as well as to deliver good performance for small messages. In a network memory system, page fault stall time is determined primarily by the time to transfer the requested page on the network. On the other hand, bursts of page transfers (e.g., for read-ahead for sequential access) require high bandwidth. The Trapeze firmware employs a message pipelining technique called *cut-through delivery* [27] to balance low payload latency with high bandwidth under load. With this technique, the one-way raw Trapeze latency for a 4K page transfer is 70μs on 300MHz Pentium-II/440LX systems with LANai 4.1 M2M-PCI32 Myrinet adapters. On these systems, Trapeze delivers 112 MB/s for a stream of 8K payloads; with 64K payloads, Trapeze can use over 95% of the peak bandwidth of the I/O bus, achieving 126 MB/s of user-to-user point-to-point bandwidth.[1]

2.2 Unified Buffering for In-Kernel Trapeze

All kernel-based Trapeze protocol modules share a common pool of receive buffers allocated from the virtual memory page frame pool; the maximum payload size is set to the virtual memory page size. Since Digital Unix allocates its file block buffers from the virtual memory page frame pool as well, this allows unified buffering among the network, file, and VM systems. For example, the system 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. Since file caching and virtual memory management are reasonably unified, we often refer to the two subsystems collectively as "the file/VM system", and use the term "page" to include file blocks.

The TCP/IP stack can also benefit from the unified buffering of Trapeze payloads to reduce copying overhead by *payload remapping* (similar to [11, 3, 17]). On a normal transmission, IP message data is copied from a user memory buffer into an mbuf chain [20] on the sending side; on the receiving side, the driver copies the header into a small mbuf, points a BSD-style external mbuf at the payload buffer, and passes the chain through the IP stack to the socket layer, which copies the payload into user memory and frees the kernel buffer. We have modified the Digital Unix socket layer to avoid copying when size and alignment properties allow. On the sending side, the socket layer builds mbuf chains by pinning the user buffer frames, marking them copy-on-write, referencing them with external mbufs, and passing them through the TCP/IP stack to the network driver, which attaches them to outgoing messages as payloads. On the receiving side, the socket layer unmaps the frames of the user buffer, replaces them with the kernel payload buffer frames, and frees the user frames. With payload remapping, AlphaStations running the standard *netperf* TCP benchmark over Trapeze sustain point-to-point bandwidth of 87 MB/s.[2]

Since outgoing payload frames attached to the send ring may be owned by the file/VM system, they must be protected from modification or reuse while a transmit is in progress. Trapeze notifies the system that it is safe to overwrite an outgoing frame by upcalling a specified *transmit completion handler* routine. For example, when an IP send on a user frame completes, Trapeze upcalls the completion routine, which unpins the frame and

[1]These bandwidth numbers define a "megabyte" as one million bytes. All other bandwidth numbers in this paper define 1MB as 1024*1024 bytes.

[2]Measured Alcor (266 MHz AS 500) to Miata (500 MHz PWS 500au), 8320-byte MTU, 1M netperf transfers, socket buffers at 1M, software TCP checksums disabled (hardware CRC only): 732 Mb/s.

releases its copy-on-write protection.

However, to reduce overhead Trapeze does not generate transmit-complete interrupts. Instead, Trapeze saves the handler pointer in host memory and upcalls the handler only when the send ring entry is reused for another send. Since messages may be sent from interrupt handlers, a completion routine could be called in the context of an interrupt handler that happened to reuse the same send ring entry as the original message. For this reason, completion handlers must not block, and the structures they manipulate must be protected by disabling interrupts. Since completion upcalls may be arbitrarily delayed, the Trapeze API includes a routine to poll all pending transmits and call their handlers if they have completed.

2.3 Incoming Payload Table

The benefits of high-speed networking are easily overshadowed by processing costs and copying overhead in the hosts. To support zero-copy communication, a Trapeze receiver can designate a region of memory as the receive buffer space for a specific incoming payload identified by a tag field. When the message arrives, the firmware recognizes the tag and deposits the payload directly into the waiting buffer. Handling of tagged payloads is governed by a third structure in NIC memory, the *incoming payload table* (IPT).

GMS uses the Trapeze IPT for copy-free handling of fetched pages in RPC replies, as described in Section 3. Ordinarily, Trapeze payloads are received into buffers attached by the host to the receive ring entries; since the firmware places messages in the ring in the order they arrive, the host cannot know in advance which generic buffer will be selected to receive any given payload, and the payload may need to be copied within the host if it cannot be remapped. Early demultiplexing with the IPT avoids this copy.

To set up an IPT mapping, the host calls a Trapeze API routine to allocate a free entry in the IPT, initialize it with the DMA address of the designated payload buffer, and return a tag value (*payload token*) consisting of an IPT index and a protection key. The payload token is a weak form of capability that can be passed in a message to another node; any node that knows the token can use it to tag a message and transmit a payload into the buffer. When the firmware receives a tagged message from the network, it validates the key against the indexed IPT entry before initiating a DMA into the designated receive buffer. The receiving host may cancel the IPT entry at any time (e.g., request timeout); similarly, the firmware protects against dangling tokens and duplicate messages by cancelling the entry when a matching message is received. If the key is not valid, the NIC drops the payload and delivers the control portion with a payload length of zero, so the receive message handler can recognize and handle the error.

At present, the IPT maps only a few megabytes of host memory, enough for the reply payloads of all outstanding requests (e.g., outstanding page fetches). This is a modest approach that meets our needs, relative to more ambitious approaches that indirect through TLB-like structures on the NIC [13, 26, 7]. We have considered a larger IPT with support for multiple transfers to the same buffer at different offsets, as in Hamlyn's *sender-based memory management* [4], but we have not found a need for these features in our current uses of Trapeze.

3 Page Transfers in GMS/Trapeze

This section outlines a Trapeze-based kernel-kernel RPC-like messaging layer designed to support cooperative cluster services. The package is derived from the original RPC package for the Global Memory Service [14] (*gms_net*), extended to use Trapeze and to support a richer set of communication styles, primarily for asynchronous prefetching at high bandwidth [24]. Although the package is generic, we draw on GMS examples to motivate its features and to illustrate their use.

Since many aspects of RPC and messaging systems are well-understood, we focus on those aspects that benefit from the Trapeze features discussed in the previous section. In particular, we explain the features for transferring pages (or file blocks) efficiently within the RPC framework, and their use by the protocol operations most critical for GMS performance: page fetches (*getpage*) from the global page cache to a local page cache, and page pushes or evictions (*putpage* or *movepage*) from a local cache to the global cache.

Section 3.2 discusses the zero-copy handling of fetched pages using the Trapeze incoming payload table (IPT); Sections 3.3 and 3.4 extend the zero-copy reply scheme to delegated and nonblocking RPC variants useful in GMS and other peer-to-peer network services. We illustrate use of nonblocking RPC to extend standard read-ahead for files and virtual memory to GMS; this allows processes to access data from network memory or storage servers at close to network bandwidth.

3.1 Basic Mechanisms

The *gms_net* messaging layer includes basic support for typed messages, stub procedures, dispatching to service procedures based on message types, and matching replies with requests. The Trapeze receiver interrupt handler directs incoming messages to *gms_net* by upcalling a registered service routine; the service routine

Explore Litigation Insights

Docket Alarm provides insights to develop a more informed litigation strategy and the peace of mind of knowing you're on top of things.

Real-Time Litigation Alerts

Keep your litigation team up-to-date with **real-time alerts** and advanced team management tools built for the enterprise, all while greatly reducing PACER spend.

Our comprehensive service means we can handle Federal, State, and Administrative courts across the country.

Advanced Docket Research

With over 230 million records, Docket Alarm's cloud-native docket research platform finds what other services can't. Coverage includes Federal, State, plus PTAB, TTAB, ITC and NLRB decisions, all in one place.

Identify arguments that have been successful in the past with full text, pinpoint searching. Link to case law cited within any court document via Fastcase.

Analytics At Your Fingertips

Learn what happened the last time a particular judge, opposing counsel or company faced cases similar to yours.

Advanced out-of-the-box PTAB and TTAB analytics are always at your fingertips.

API

Docket Alarm offers a powerful API (application programming interface) to developers that want to integrate case filings into their apps.

LAW FIRMS
Build custom dashboards for your attorneys and clients with live data direct from the court.

Automate many repetitive legal tasks like conflict checks, document management, and marketing.

FINANCIAL INSTITUTIONS
Litigation and bankruptcy checks for companies and debtors.

E-DISCOVERY AND LEGAL VENDORS
Sync your system to PACER to automate legal marketing.

WHAT WILL YOU BUILD? | sales@docketalarm.com | 1-866-77-FASTCASE

fastcase® Smarter legal research.