RAID-II: A High-Bandwidth Network File Server

Ann L. Drapeau Ken Shirriff Edward K. Lee
John H. Hartman Ethan L. Miller Srinivasan Seshan
Randy H. Katz Ken Lutz David A. Patterson
University of California
571 Evans Hall
Berkeley, CA 94720
{drapeau,shirriff,eklee,jhh,elm,ss,randy,lutz,pattrsn}@cs.berkeley.edu
Phone: (510) 642-1845,642-9669,642-1845,642-9669,642-8248,642-8248,642-6037,643-7803,642-6037
FAX: (510) 642-5775

Peter M. Chen
Computer Science and Engineering Dept.
University of Michigan
Ann Arbor, MI 48109-2122

Garth A. Gibson
School of Computer Science
Carnegie-Mellon University
Pittsburgh, PA 15213-3890
garth@cs.cmu.edu
Phone: (412) 268-5890
FAX: (412) 681-5739

Abstract

In 1989, the RAID group at U. C. Berkeley built a prototype disk array called RAID-I. The bandwidth achieved by RAID-I was severely limited by the memory system bandwidth limitations of the disk array's host workstation. As a result, most of the bandwidth available from the disks could not be delivered to clients of the disk array file server. We designed our second prototype, RAID-II, to deliver as much of the disk array bandwidth as possible to file server clients. A custom-built circuit-board disk array controller, called the XBUS board, connects the disks and the high-speed network directly, allowing data for large requests to bypass the server workstation. A single workstation may control several XBUS boards for increased bandwidth. RAID-II runs the Log-Structured File System (LFS) to optimize the performance of the disk array for bandwidth-intensive applications.

The RAID-II hardware with a single XBUS controller board delivers 20 megabytes/second of I/O between the disks and high-speed networks. This performance is an order of magnitude better than our first prototype, but somewhat lower than our performance goals because of lower-than-expected performance of the commercial disk controller boards and our disk system interfaces. A preliminary implementation of LFS delivers 13.4 megabytes/second to the clients.

Key words: high-bandwidth network file service, local area networks, data services, network storage, mass storage system, RAID.

1 Introduction

It is essential for future file servers to provide high-bandwidth I/O because of a trend toward bandwidth-intensive applications like multi-media, CAD, object-oriented data bases and scientific visualization. Even in well-established application areas such as scientific computing, the size of data sets is growing rapidly because of reductions in the cost of secondary storage and the introduction of faster supercomputers. These developments require faster I/O systems to transfer the increasing volume of data.

High performance file servers will increasingly incorporate disk arrays to provide greater disk bandwidth. RAIDs, or Redundant Arrays of Inexpensive Disks [10], replace large, expensive mainframe-type disks with a collection of small, inexpensive disks to deliver higher performance and greater reliability from the secondary storage system. RAIDs provide greater disk bandwidth to a file by striping or interleaving the data from a single file across a group of disk drives, allowing transfers to occur in parallel. RAIDs ensure reliability by calculating error correcting codes across a group of disks; this redundancy information can be used to reconstruct the data on disks that fail. In this paper, we examine the efficient delivery of bandwidth from a file server that includes a disk array.

In 1989, the RAID group at U.C. Berkeley built an initial RAID prototype, called RAID-I [2]. The prototype was constructed using a Sun 4/280 workstation with 128 MB of memory, four dual-string SCSI controllers, 28 5-1/4 inch SCSI disks and specialized disk striping software. The purpose of the prototype was to understand how well a disk array constructed of commercially-available components would perform on two workloads: small, rather random operations typical of file servers in a workstation environment, and large transfers typical of bandwidth-intensive applications.

Experiments with RAID-I show that it performs well when processing small, random I/O's, performing approximately 275 4 KB random I/Os per second. However, RAID-I has proven woefully inadequate at providing high-bandwidth I/O, sustaining at best 2.3 megabytes/second to a user-level application on RAID-I. By comparison, a single disk on RAID-I can sustain 1.3 megabytes/second. Most of the bandwidth available from the 28 disks in the array is effectively wasted, because it cannot be delivered to clients.

There are several reasons why RAID-I is ill-suited for high-bandwidth I/O. The most serious is the memory contention experienced on the Sun 4/280 server during I/O operations. The copy operations performed to move data between the kernel DMA buffers and buffers in user space saturate the memory system when I/O bandwidth reaches 2.3 megabytes/second. Second, because all I/O on the Sun 4/280 goes through the CPU's virtually addressed cache, data transfers experience interference from cache flushes. Finally, disregarding the workstation's memory bandwidth limitation, high-bandwidth performance is also limited by the low backplane bandwidth of the Sun 4/280's VME system bus, which becomes saturated at 9 megabytes/second.

The problems RAID-I experienced are typical of many "CPU-centric" workstations that are designed for good processor performance but fail to support adequate I/O bandwidth. In such workstations, the memory system is designed so that the CPU has the fastest and highest-bandwidth path to memory. For busses or backplanes farther away from the CPU, the available memory bandwidth drops quickly. In general, today's workstations are poorly suited for supporting high-bandwidth I/O because they achieve high performance using large fast caches, without significantly improving the performance of the primary memory and I/O systems [5], [9].

The design of our second prototype, RAID-II, was motivated by a desire to preserve as much bandwidth from the disk array as possible for delivery to the file server's clients. One way we preserve disk array bandwidth is to implement separate low- and high-bandwidth data paths. The high-bandwidth path allows data to be transferred directly between the disks and a high-speed network, without passing through the memory of the file server workstation, as occurs in traditional file servers and RAID-I. To achieve this we built a custom storage controller board called the XBUS board. The XBUS board contains a high-speed crossbar, internal memory, and interfaces to the network and disks. In response to control instructions from the workstation, the XBUS board manages transfers between the disks and network. The separate low- and high-bandwidth paths make it possible to achieve good utilization of the high-bandwidth path by sending small data transfers and control operations over the low-bandwidth path. The separate paths also allow the server to support clients on a local Ethernet as well as across the high-speed network. Each XBUS board can support approximately 20 megabytes/second of disk activity. To achieve greater bandwidth, several XBUS boards can be attached to a single workstation.

Efficient file server software is also essential in preserving the bandwidth of the disk array. RAID-II runs LFS [11], the Log-Structured File System, developed by the Sprite operating system group at Berkeley. LFS is a file system specially optimized to support high-bandwidth I/O by treating the disks or disk array as a log, and writing large segments of data sequentially. Small operations are grouped together to avoid wasting disk bandwidth. This makes LFS especially well-suited for use with disk arrays. As is well documented [13], small writes to a disk array require multiple disk accesses to fetch the old data and parity and to write the new data and parity. By eliminating small write operations, LFS eliminates this most costly mode of access for disk arrays. Another benefit of LFS is fast crash recovery. A preliminary implementation of LFS running on RAID-II with a single XBUS board delivers 13.4 megabytes/second through the XBUS board.

This paper describes the design, implementation and performance of the RAID-II prototype. Section 2 describes the RAID-II hardware, including the design choices made to preserve bandwidth, architecture and implementation details, and benchmark performance measurements. Section 3 discusses the implementation and performance of LFS running on RAID-II. Section 4 compares other high performance I/O systems to RAID-II, and Section 5 discusses future directions. Finally, we summarize the contributions of the RAID-II prototype.

2 RAID-II Hardware

Figure 1 illustrates the RAID-II storage architecture. The RAID-II file server spans three racks. It includes a workstation, also referred to as the host, that controls one or more XBUS controller boards. This host workstation has an Ethernet connection that allows transfers between the host

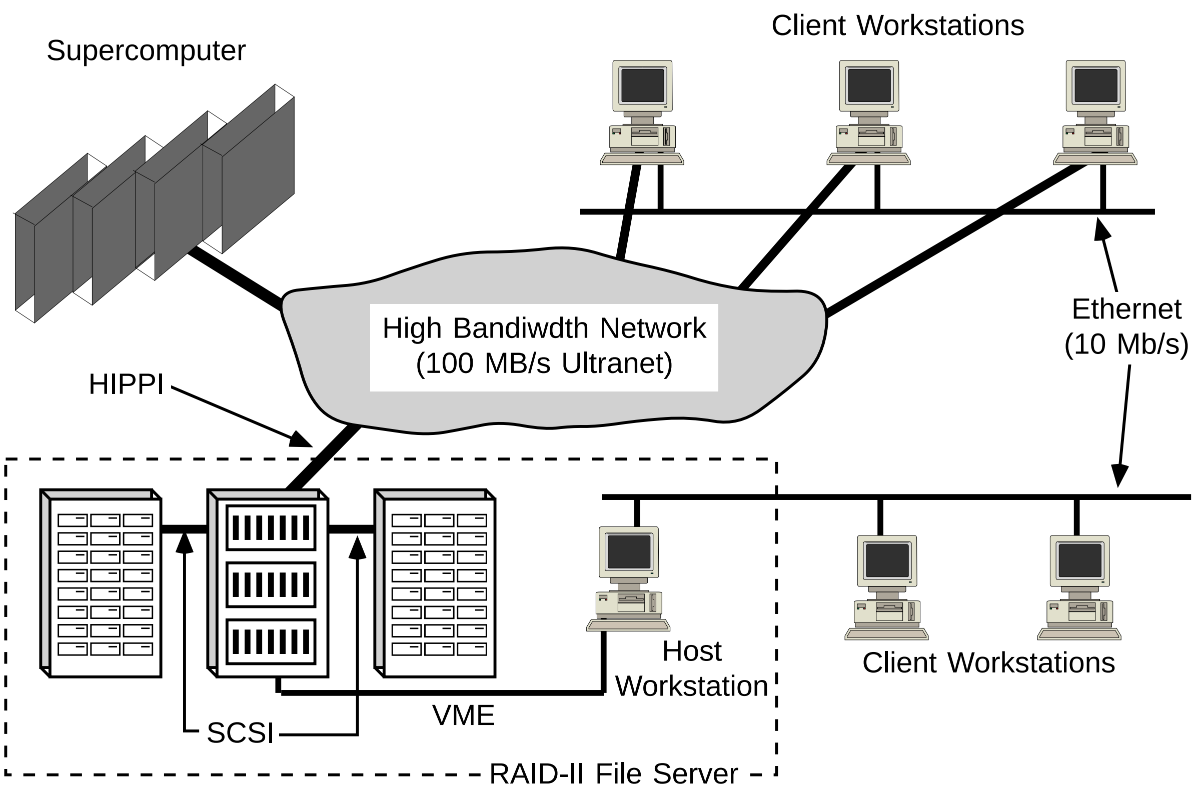

Figure 1: RAID-II File Server Architecture. RAID-II is composed of three racks, including a host workstation, shown by dotted lines. The host workstation is connected by Ethernet to some client workstations. The host is also connected to one or more XBUS controller boards (contained in the center rack of RAID-II) over the VME backplane. Each XBUS controller has a HIPPI network connection, which connects to the Ultranet high-speed ring network; the Ultranet may have connections to client workstations and supercomputers. The XBUS boards also have connections to SCSI disk controller boards.

and clients on the Ethernet network. Each XBUS controller board has a HIPPI connection to a high-speed Ultranet ring network that may also connect supercomputers and client workstations.

In this section, we discuss the prototype hardware. First, we describe the design decisions that were made to preserve as much of the disk array bandwidth as possible. Next, we describe the architecture and implementation, followed by microbenchmark measurements of hardware performance.

2.1 Design Choices That Preserve Bandwidth

Several important design decisions in the RAID-II architecture reflect our desire to deliver high bandwidth from the disks to the clients.

Disk array bandwidth on our first prototype was limited by low memory system bandwidth on the host. To avoid a similar limitation in RAID-II, we bypass the host's memory during large data transfers. We avoid connecting the disk array, a high-bandwidth secondary storage system, to the high-bandwidth HIPPI network via the host's low-bandwidth backplane and memory bus. Instead, our XBUS controller board connects the disk array and HIPPI network directly using a high-bandwidth crossbar interconnect. The host still controls the movement of data; the XBUS merely responds to commands sent by the host when transferring data between disks and the network. The host also manages the file metadata (data associated with the file other than its contents); translates hierarchical, human-readable names to logical device addresses; and maintains a file cache of recently-accessed files in host memory.

Besides the 100 megabytes/second HIPPI interface connection to each XBUS board, the host workstation has a low-latency 10 megabytes/second Ethernet interface. Any request can go over either datapath, but we maximize utilization and performance of the high-bandwidth datapath if smaller requests use the Ethernet network and larger requests use the HIPPI network.

Thus, there are two modes of data access in RAID-II, which we call *standard mode* and *high-bandwidth mode*. Standard mode is used for smaller data transfers and file system operations such as `open`, `close` and `fstat`. During such accesses, data are transferred from the disks through the XBUS board and host memory before being sent over the Ethernet attached to the host. RAID-II's standard mode transmits data and control messages together to reduce the number of network messages and the software overhead.

High-bandwidth mode, by contrast, is optimized to provide a high rate of data transfer for large file requests. This mode uses the high-performance data path just described. The XBUS board bypasses host memory and transfers data from the disks, through the XBUS board's crossbar interconnect and memory, and over the high-performance HIPPI network. In high-bandwidth mode, the host processes each data request by setting up data transfers directly between the XBUS controller and client. Implementation details for each data transfer mode are given in Section 3.

A host workstation may control multiple XBUS controller boards. Thus, RAID-II can scale to provide greater bandwidth by using several XBUS boards. To some extent, adding XBUS boards is like adding disks to a conventional file server. An important distinction is that adding an XBUS board to RAID-II increases the I/O bandwidth available to the *network*, whereas adding a disk to a conventional file server only increases the I/O bandwidth available to that particular file server. The latter is less effective, since the file server's memory system and backplane will soon saturate if it is a typical workstation.

2.2 Architecture and Implementation of RAID-II

Figure 2 illustrates the architecture of the RAID-II file server. The file server's backplane consists of two high-bandwidth (HIPPI) data busses and a low-latency (VME) control bus. The backplane interconnects the high-bandwidth network interfaces, several XBUS controllers and a host workstation operating that controls the operation of the XBUS controller boards. We use the high-bandwidth HIPPI datapath for large data transfers, and the VME control bus for sending control information and small data transfers between the host workstation and the XBUS board. Each XBUS board contains interfaces to the HIPPI backplane and VME control bus, memory, a parity computation engine and four VME interfaces that connect to disk controller boards. The Ethernet interface on the host workstation is used for data transfer to client workstations on that Ethernet network.

Figure 3 illustrates the physical packaging of the RAID-II file server, which is implemented using three racks. To minimize the design effort, we used commercially available components whenever possible. Thinking Machines Corporation (TMC) provided a board set for the HIPPI interface to the Ultranet ring network; Interphase Corporation provided VME-based, dual SCSI, Cougar disk controllers; Sun Microsystems provided the Sun 4/280 file server; and IBM donated disk drives and

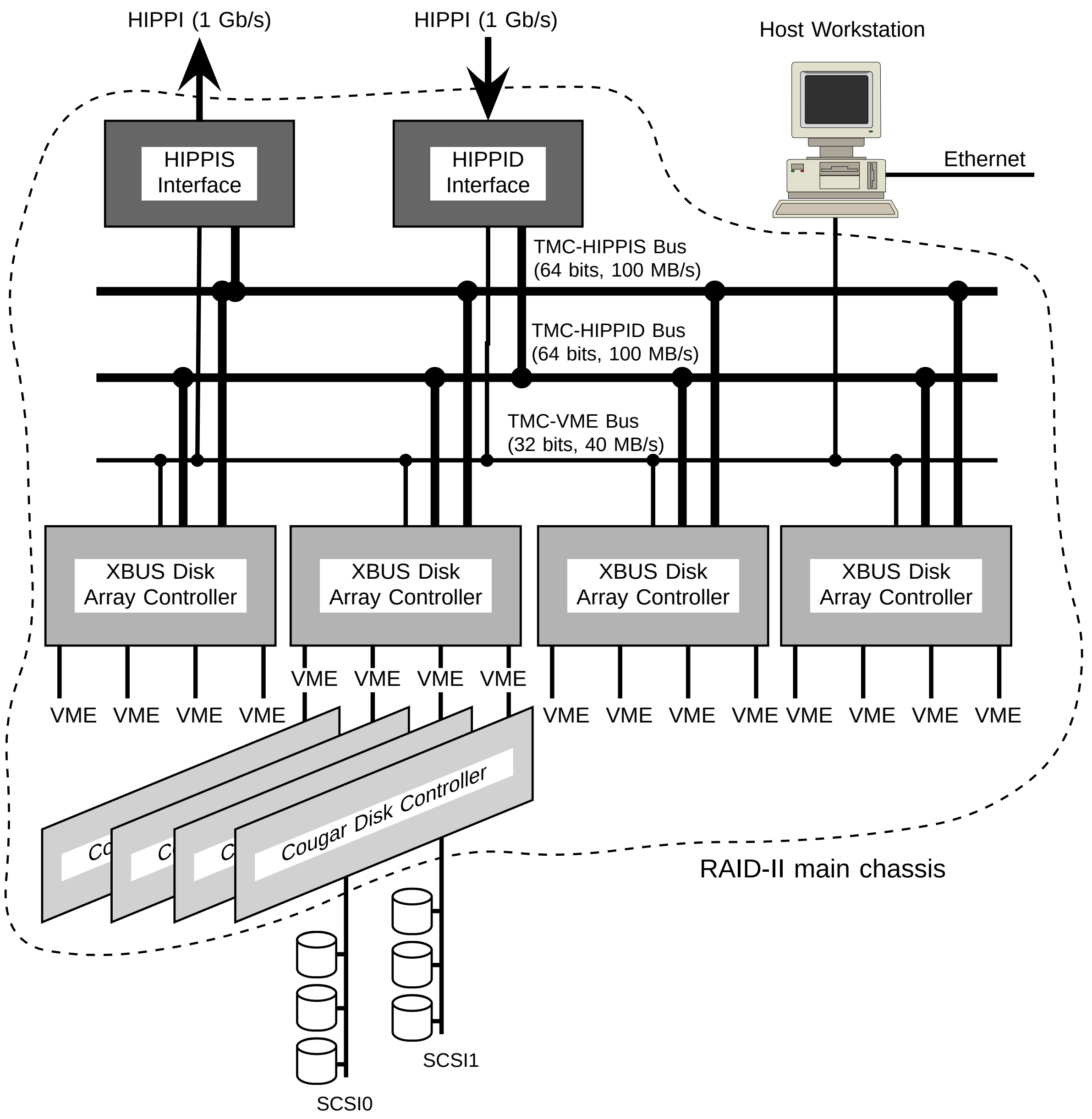
HIPPI (1 Gb/s)
HIPPI (1 Gb/s)
Host Workstation
Ethernet
HIPPIS Interface
HIPPID Interface
TMC-HIPPIS Bus (64 bits, 100 MB/s)
TMC-HIPPID Bus (64 bits, 100 MB/s)
TMC-VME Bus (32 bits, 40 MB/s)
XBUS Disk Array Controller
XBUS Disk Array Controller
XBUS Disk Array Controller
XBUS Disk Array Controller
VME
Cougar Disk Controller
SCSI0
SCSI1
RAID-II main chassis

Figure 2: Architecture of RAID-II File Server. The host workstation may have several XBUS controller boards attached to its VME backplane. Each XBUS controller board contains interfaces to HIPPI network source and destination boards, internal memory, high-bandwidth crossbar, parity engine, and interfaces to four SCSI disk controller boards.

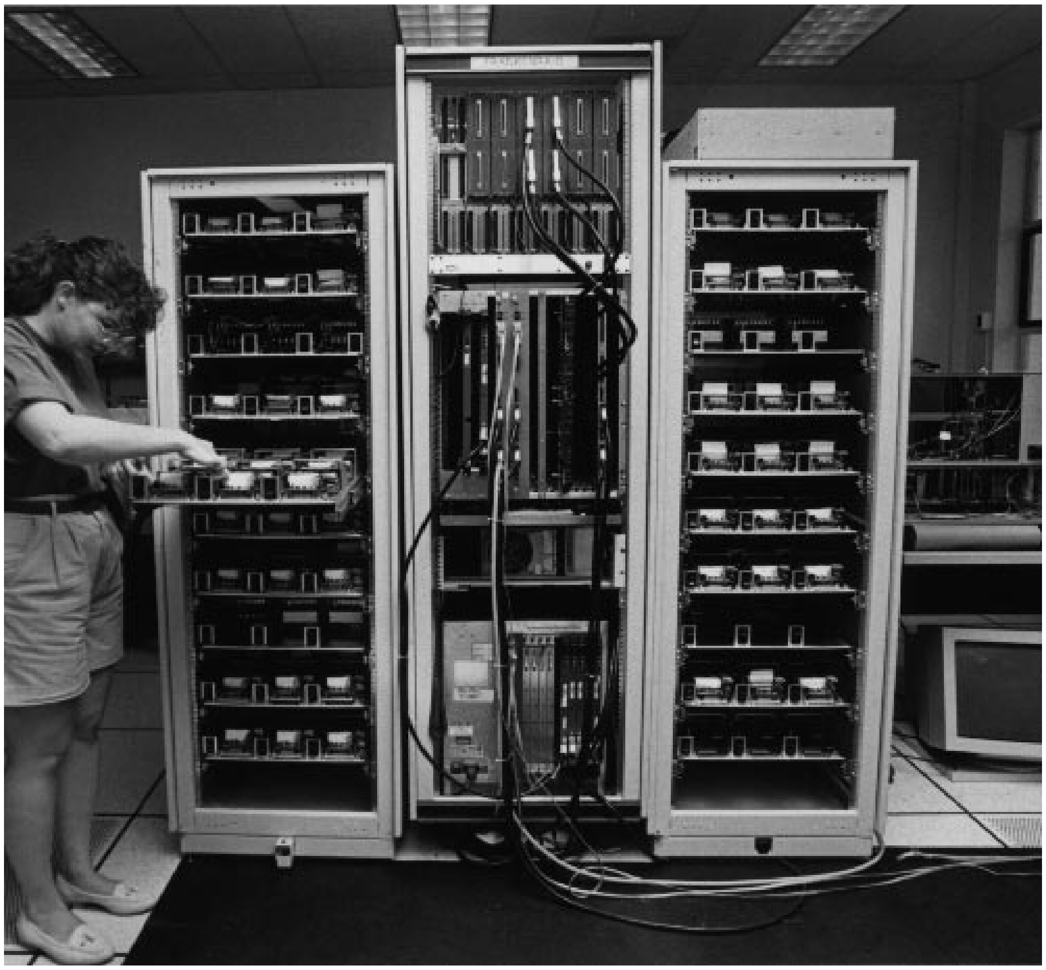

Figure 3: The physical packaging of the RAID-II File Server. Two outer racks contain 144 disks and their power supplies. The center rack contains three chassis: the top chassis holds VME disk controller boards; the center chassis contains XBUS controller boards and HIPPI interface boards, and the bottom VME chassis contains the Sun4/280 workstation.

DRAM. The center rack is composed of three chassis. Above is a VME chassis containing eight Interphase Cougar disk controllers. The center chassis was provided by TMC and contains the HIPPI interfaces and our custom XBUS controller boards. The bottom VME chassis contains the Sun4/280 host workstation. Two outer racks each contain 72 three-and-a-half inch, 320 MB IBM SCSI disks and their power supplies. There are eight shelves of nine disks per rack. RAID-II has a total capacity of 46 GB.

We designed the custom printed-circuit XBUS controller board called the XBUS board. The

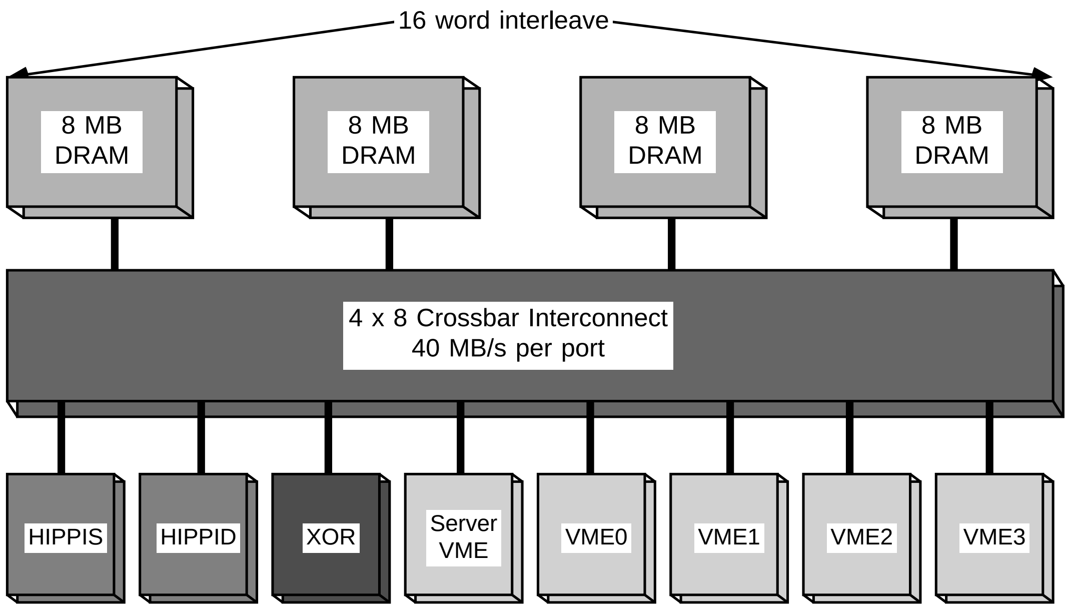

Figure 4: Structure of XBUS controller board. The board contains four memory modules connected by a 4x8 crossbar interconnect to eight XBUS ports. Two of these XBUS ports are interfaces to the HIPPI network. Another is a parity computation engine. One is a VME network interface for sending control information, and the remaining four are VME interfaces connecting to commercial SCSI disk controller boards.

main purpose of the XBUS board is to provide a high-bandwidth path between the major system components: the HIPPI network interface, four VME interfaces that connect to Cougar disk controller boards, a parity computation engine, and an interleaved, multiported semiconductor memory.

Figure 4 is a block diagram of the XBUS disk array controller board. The XBUS board implements a 4x8, 32-bit wide crossbar interconnect, which we call the XBUS. The crossbar connects four memory modules and eight XBUS ports. All XBUS transfers involve one of the four memory modules as either the source or the destination of the transfer. Each memory port is designed to transfer bursts of data at 50 megabytes/second and sustain transfers at 40 megabytes/second, for a total sustainable memory bandwidth on the XBUS board of 160 megabytes/second.

The XBUS is a synchronous multiplexed (address/data) crossbar-based interconnect that uses a centralized strict priority-based arbitration scheme. Each of the eight 32-bit XBUS ports operates at a cycle time of 80 ns. The memory is interleaved in sixteen-word blocks. The XBUS supports only two types of bus transactions: reads and writes.

An important concern in the design of the XBUS was contention for memory modules. In practice, contention is infrequent because most XBUS ports perform large sequential accesses. When XBUS ports conflict, the loser of the arbitration ends up following the winner around the memory modules in a deterministic manner, avoiding further conflicts. Also, each XBUS port can buffer at least a kilobyte of data to/from the XBUS to even out fluctuations caused by memory conflicts. Measurements of the XBUS under heavy load [1] show that contention is not a problem.

The main advantage of the crossbar-based memory system is that high aggregate memory bandwidth is made available using relatively inexpensive 32-bit ports. One disadvantage is that a single port cannot utilize the full 160 megabytes/second of memory bandwidth but is limited to 40 megabytes/second. In our case, this is not a serious problem since only the HIPPI ports could have sustained more than 40 megabytes/second. Another disadvantage is that, although the ports are inexpensive, the crossbar itself is expensive. We implemented the crossbar using 192 16-bit transceivers. Using surface mount packaging we are able to implement the crossbar in 120 square inches or approximately 20 % of the XBUS controller's board area.

Of the eight XBUS ports, two interface to the HIPPI source and destination busses. Each TMC HIPPI board also interfaces to one of these busses. The XBUS HIPPI ports are unidirectional and can sustain transfers of up to 40 megabytes/second, with bursts of up to 100 megabytes/second into 32 KB FIFO interfaces.

Four of the XBUS ports are used to connect the XBUS board to four VME busses, each of which may connect to one or two dual-string Interphase Cougar disk controllers. In our current configuration, we connect three disks to each SCSI string, two strings to each Cougar controller, and one Cougar controller to each XBUS VME interface for a total of 24 disks per XBUS board. Up to 96 disks could be connected to an XBUS board using six disks per string, two strings per Cougar controller, and two controllers per XBUS VME port. The Cougar disk controllers can

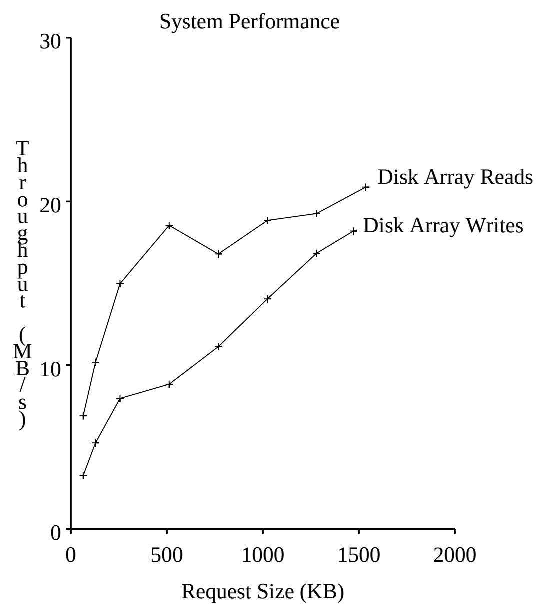

Figure 5: System Level Read and Write Performance. RAID-II achieves approximately 20 megabytes/second for both reads and writes.

transfer data at 8 megabytes/second, for a total maximum bandwidth to the disk array of 32 megabytes/second in our present configuration. As will be shown in the next section, the four XBUS VME ports interfacing to the Cougar disk controllers have the lowest bandwidth of any of the XBUS ports, about 7 megabytes/second each. This low performance is largely due to the difficulty of synchronizing the asynchronous VME bus for interaction with the XBUS.

Of the remaining two XBUS ports, one interfaces to a parity computation engine. The last port is the VME control interface linking the XBUS board to the host workstation. It provides the host with access to the XBUS board's memory as well as its control registers. This makes it possible for file server software running on the host to access network headers and file metadata in the XBUS memory.

2.3 Performance

In this section, we present the performance of the RAID-II hardware, running below the level of the operating system software. In Section 3.4, we show how much of this raw hardware performance is delivered by the file system.

Figure 5 shows system performance for reads and writes. We refer to these performance measurements as system-level experiments, since they involve all the components of the system from the disks to the HIPPI network. For these experiments, the disk system is configured as a RAID Level 5 [10] with one parity group of 24 disks. For reads, data are read from the disk array into the memory on the XBUS board and from there are sent over HIPPI, back to the XBUS board, and into XBUS memory. For writes, data originate in XBUS memory, are sent over the HIPPI and then back to the XBUS board to XBUS memory; parity is computed, and then both data and parity are written to the disk array. For both tests, the system is configured with four Interphase Cougar disk controllers, each with two strings of three disks. For both reads and writes, subsequent operations are at random locations. Figure 5 shows that, for large requests, system-level read and write performance reaches about 20 megabytes/second. Writes are slower than reads due to the increased disk and memory activity associated with computing and writing parity. While an order of magnitude faster than our previous prototype, RAID-I, this is still well below our target bandwidth of 40 megabytes/second. Below, we show that system performance is limited by that of the commercial disk controller boards and our disk interfaces.

Table 1 shows peak performance of the system when sequential read and write operations are performed. These measurements were obtained using the four Cougar boards attached to the XBUS VME interfaces, and in addition, having a fifth Cougar board attached to the XBUS VME control bus interface. Read performance is 31 megabytes/second, compared to 23 megabytes/second for writes. There are two reasons why read performance is so much better than write performance. First, extra accesses are required on writes to calculate and write parity. Second, sequential reads benefit from the read-ahead performed into track buffers on the disks; writes have no such advantage.

While one of the main goals in the design of RAID-II was to provide better performance than our first prototype on high-bandwidth operations, we also want the file server to perform well on small, random operations. Table 2 compares the I/O rates achieved using a test program that performed random 4 kilobyte reads. In each case, fifteen disks were accessed. The tests used Sprite

Operation	Peak Performance (megabytes/second)
Sequential reads	31
Sequential writes	23

Table 1: Peak read and write performance for an XBUS board on sequential operations. Obtained using the four vme interfaces to disk controllers, plus attaching a disk controller to the VME control bus interface.

System	1 disk I/O Rate (I/Os/sec)	15 disks I/O Rate (I/Os/sec)
RAID-I	27	274
RAID-II	36	422

Table 2: Peak I/O rates in operations per second for random, 4 kilobyte reads. Performance for a single disk, and for fifteen disks, with a separate process issuing random I/O operations to each disk. The IBM 0661 disk drives used in RAID-I have shorter seek and rotation times than the Seagate Wren IV disks used in RAID-II.

operating system read and write system calls; the RAID-II test did not use LFS. In each case, a single process issued 4 kilobyte, randomly distributed I/O requests to each active disk in the system. The performance with a single disk indicates the difference between the Seagate Wren IV disks used in RAID-I and the IBM 0661 disks used in RAID-II. The IBM disk drives have faster rotation and seek times, making random operations quicker. With fifteen disks active, both systems perform well. The small, random I/O rates are limited in both cses by the large number of context switches required on the Sun4/280 workstation.

The remaining performance measurements in this section are for specific components of the XBUS board. Figure 6 shows the performance of the HIPPI network and boards. Data are transferred from the XBUS memory to the HIPPI source board to the HIPPI destination board and back to XBUS memory. Because the network is configured as a loop, there is minimal network protocol overhead. This test focuses on the HIPPI network's raw hardware performance; unfortunately, this level of bandwidth cannot be achieved over the Ultranet network, which limits packet size to 32 kilobytes. In the loopback mode, the overhead of sending a HIPPI packet is about 1.1 ms, mostly due to setting up the HIPPI and XBUS control registers across the slow VME link (in comparison, an Ethernet packet takes approximately 0.5 ms to transfer). Due to this control overhead, small

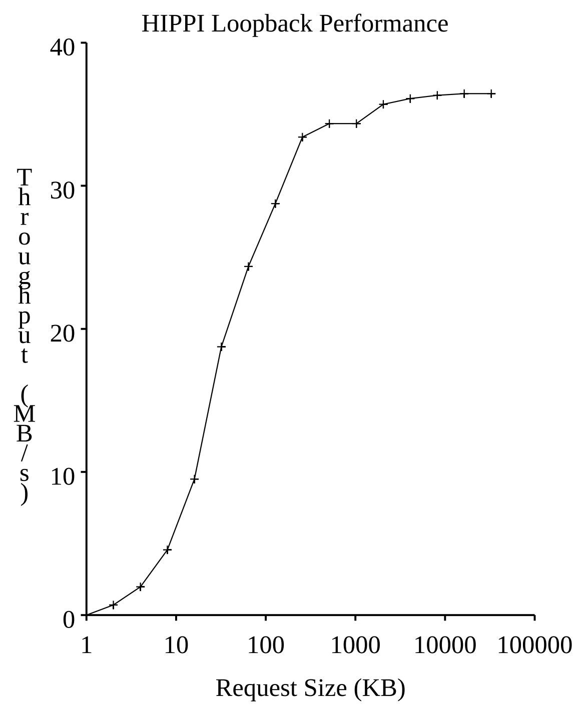

Figure 6: HIPPI Loopback Performance. RAID-II achieves 38.5 megabytes/second in each direction.

requests result in low performance. For large requests, however, the XBUS and HIPPI boards support 38 megabytes/second in both directions, which is very close to the maximum bandwidth of each XBUS port. During these large transfers, the XBUS boards are transferring a total of 76 megabytes/second, which is an order of magnitude faster than FDDI and two orders of magnitude faster than Ethernet. Clearly the HIPPI part of the XBUS is not the limiting factor in determining system level performance.

Disk performance is responsible for the lower-than-expected system-level performance of RAID-II. The performance limitations are the Cougar disk controllers, which only support about 3 megabytes/second on each of two SCSI strings, as shown in Figure 7, and our relatively slow, synchronous VME interface ports, which only support 6.9 megabytes/second on read operations and 5.9 megabytes/second on write operations.

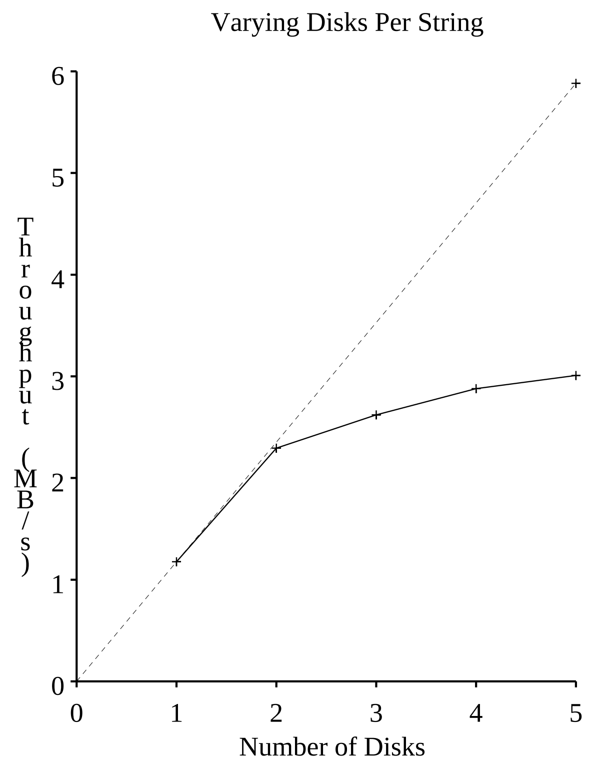

Figure 7: Disk Read Performance for varying number of Disks on a Single SCSI string. Cougar string bandwidth is limited to about 3 megabytes/second, less than that of three disks. Dashed line indicates the performance if bandwidth scaled linearly.

3 The RAID-II File System

Although some applications can use RAID-II as a large raw disk, most applications will require a file system with abstractions such as directories and files. The file system should deliver to applications as much of the bandwidth of the RAID-II disk array hardware as possible. RAID-II's file system is a modified version of the Sprite Log Structured File System (LFS) [11]. LFS is particularly well-suited for use with disk arrays because it writes data in large segments to an append-only disk log. This eliminates inefficient small write accesses to the disk array. Changes to the original LFS implementation were required to manage two features of RAID-II: the separate low- and high-bandwidth datapaths, and the separate file system caches on the host computer and XBUS board. Currently the RAID-II file system can provide 13.4 megabytes/second of data for large read requests.

In this section, we first describe the Log Structured File System and its suitability for disk arrays. Second, we describe the unique features of RAID-II that require special attention in the LFS implementation. Next, we illustrate how RAID-II processes typical read requests from a client. Finally, we discuss the measured performance and implementation status of LFS on RAID-II.

3.1 The Log Structured File System

LFS [11] is a disk storage manager that writes all file data and metadata to a sequential append-only log; files are not assigned fixed blocks on disk. In LFS, there are no small write operations to disk; data for small writes are buffered in main memory, and are written to disk asynchronously when a log segment fills. This is in contrast to traditional file systems, which assign files to fixed blocks on disk. In traditional file systems, a sequence of random file writes results in inefficient small, random disk accesses.

Because it eliminates small write operations, the Log Structured File System is better suited to achieving high bandwidth from a disk array than traditional file systems. This is because small write operations in a Level 5 RAID are inefficient, requiring four disk accesses: reads of the old data and parity blocks, and writes of the new data and parity blocks. Large accesses to the disk array are much more efficient, because they don't require reading old data or parity. A traditional file system running on a RAID would perform inefficient small write operations, but LFS eliminates them, grouping small operations into efficient large, sequential write operations.

A secondary benefit of LFS is that crash recovery is very quick. To recover from a file system crash, the LFS server need only read the log from the position of the last checkpoint. By contrast, a UNIX file system consistency checker traverses the entire directory structure in search of lost data. Because a RAID has a very large storage capacity, a standard UNIX-style consistency check on boot would be unacceptably slow. For a 1 GB file system, it takes less than a second to perform an LFS file system check, compared with approximately 20 minutes to check the consistency of a typical UNIX file system of comparable size.

3.2 LFS on RAID-II

An implementation of LFS for RAID-II must efficiently handle two architectural features of the disk array: the separate low- and high-bandwidth datapaths, and the separate memories on the host workstation and the XBUS board.

Because of the existence of the two datapaths, we changed the original Sprite LFS software to separate the handling of data and metadata (information associated with a file other than its contents, such as inodes and directories). We use the high-bandwidth datapath to transfer data between the disks and the HIPPI network, without passing through host workstation memory. The relatively low-bandwidth VME connection between the XBUS memory and the host workstation is needed to send metadata to the host; the host uses metadata to look up file names and locate data on disk. The low-bandwidth path also transfers data to the host to service small requests made by clients connected to the host's Ethernet network. Small requests are handled with this slower datapath to allow efficient use of the high-bandwidth datapath for larger requests.

LFS on RAID-II also must manage separate memories on the host workstation and the XBUS board. The host memory cache contains metadata as well as files that have been read into workstation memory for transfer over the Ethernet. The cache is managed with a simple Least Recently Used replacement policy. Memory on the XBUS board is used for prefetch buffers, HIPPI network buffers, and as write buffers for LFS segments. The file system keeps the two caches consistent, and copies data between them as required.

To hide some of the latency required to service large requests, LFS performs prefetching into XBUS memory buffers. LFS on RAID-II uses several prefetch processes that request data in parallel to increase the total bandwidth available to the application. Latency is also minimized by pipelining network and disk operations.

From the user's perspective, the RAID-II file system looks like a standard file system to clients using the slow path, but requires relinking of applications for clients to use the fast path. To access data over the Ethernet, clients just use NFS or the Sprite RPC mechanism. The fast datapath across the Ultranet uses a special library of file system operations for RAID files: open, read, write, etc. The library converts file operations to operations on an Ultranet socket between the client and the RAID-II server. The advantage of this approach is that it doesn't require changes to the client operating system. Alternatively, these operations could be moved into the kernel on the client and would be transparent to the user-level application.

3.3 Typical Large and Small Requests

This section explains how the RAID-II file system handles typical open and read request from a client over the fast Ultranetwork and over the slow Ethernet network attached to the host workstation.

In the first example, the client is connected to RAID-II across the high-bandwidth Ultranet network. The client application is linked with a small library which converts raid file operations into operations on an Ultranet socket connection.

First, the application running on the client performs an open operation by calling the library routine `raid_open`. This call is transformed by the library into socket operations. The library opens a socket between the client and the RAID-II file server. Then the client sends an open command. Finally, the RAID-II file server opens the file and informs the client.

Now the application can perform read and write operations on the file. Suppose the application does a large read using the library routine `raid_read`. As before, the library converts this read operation into socket operations, sending RAID-II a read command that includes file position and request length.

RAID-II handles a read request by pipelining disk reads and network sends. First, the file system code allocates a buffer in XBUS memory, determines the position of the data on disk, and calls the RAID driver code to read the first block of data into XBUS memory. When the read has completed, the file system calls the network code to send the data from XBUS memory to the client. Meanwhile, the file system allocates another XBUS buffer and reads the next block of data. Network sends and disk reads continue in parallel until the transfer is complete. LFS may have several prefetch processes issuing read requests, allowing disk reads to get ahead of network send operations for efficient network transfers.

On the client side, the network data blocks are received and read into the application memory using socket read operations. When all the data blocks have been received, the library returns from the `raid_read` call to the application.

In the second example, we illustrate how the file server handles a read request sent over the relatively slow Ethernet. A client sends standard file system `open` and `read` calls to the host workstation across its Ethernet connection. The host sends the read commands over the VME link to the XBUS board. The XBUS board responds by reading the data from disk into XBUS memory, and transferring the data to the host workstation. Finally, the host workstation packages the data into Ethernet packets and sends the packets to the client.

3.4 Performance and Status

The file system for RAID-II is still under development and is being used to research various design parameters. It currently handles creation, reads, and writes of files. LFS cleaning (garbage collection of free disk space in the log) has not yet been implemented.

Currently, the LFS file system implementation provides about 67% of the 20 megabytes/second total bandwidth of RAID-II with a single XBUS board. With 24 disks active in the array, the file system delivers 13.4 megabytes/second for a workload of large read requests. This experiment involves a single application process issuing 30 megabyte read requests. LFS breaks down these requests, so that requests to the disk array are approximately 650 kilobytes. The raw hardware performance for 650 kilobyte requests is 14 megabytes/second, so the LFS implementation on RAID-II delivers 96% of the array bandwidth for this workload. During this experiment, LFS uses three processes for efficient data prefetching. Data are striped in the array in units of 16 kilobytes. In this experiment, data read from the disks are not sent across the HIPPI network; we currently do not have a client capable of receiving 13.4 megabytes/second of data. The data rate reported here is the rate at which data are read from the disk array and written to HIPPI network buffers in the XBUS board's memory.

LFS performance is fairly poor across the host workstation's Ethernet path. Data read and write rates are about 350 kilobytes/second. The low bandwidth is due to the file system, which breaks Ethernet requests into 4 kilobyte blocks; RAID-II has low performance on these small blocks. The Ethernet path performance would increase dramatically with a larger block size. Due to time constraints, we have not yet tuned the Ethernet datapath.

We are exploring several design issues with the RAID-II file system, including the best use of XBUS memory for caching or prefetching, the best size of data blocks for efficient transfers, and the performance of the file system under high loads.

4 RAID-II and Existing File Server Architectures

RAID-II is not the only system offering high I/O performance. In this section, we highlight some important characteristics of several existing high-performance I/O systems, and explain how they relate to the design of RAID-II.

4.1 Auspex NS5000

The Auspex NS5000 [8] is a file server designed to support a large number of client workstations using NFS [12], the most common network file system protocol for workstation-based computing environments. The design of the Auspex NS5000 was motivated by the lack of scalability of conventional workstation file servers. Each NFS packet processed by a file server requires an approximately constant CPU overhead to process device interrupts and manage the network protocol. In a conventional network file server, a single processor performs all these functions; a large number of clients generate many small NFS packets, causing the file server processor to become a bottleneck. The NS5000 supports more clients than conventional file servers by dividing the functions normally performed by a single CPU over several processors. This *functional multiprocessing* design uses dedicated processors to perform network processing, file system management and disk control. As a result, the NS5000 performs better than a comparably configured Sun 4/490 server by a ratio ranging from 1.4, when both are configured with a single disk and single Ethernet, to 10.0, when both have two Ethernets and four disks [6]. RAID-II utilizes a similar degree of functional multiprocessing, with the host workstation handling metadata, name translation, and small accesses over the local Ethernet network; the XBUS board responsible for storage and retrieval of disk data; and an AMD29000 on the TMC HIPPI boards as well as the Ultranet hardware handling most of the HIPPI protocol handling.

Although the NS5000 is good at supporting small, low-latency NFS requests, it does not perform as well for high-bandwidth applications. Currently, the system contains eight Ethernet networks, limiting aggregate bandwidth to around 8 megabytes/second. Also, the current version of NFS limits packet size to 8 kilobytes, limiting the bandwidth available to individual users. Once the packet size limit is removed in future versions of NFS, file server performance may reach 25 megabytes/second. Bandwidth is constrained by the 55 megabytes/second VME bus connecting networks and disks.

4.2 Supercomputer File Systems

Supercomputers have long used high performance I/O systems. These systems fall into two categories — high-speed disks attached directly to the supercomputer, and mainframe-managed storage connected to the supercomputer via a network.

The first type of supercomputer file system, consisting of many fast parallel transfer disks directly attached to a supercomputer's I/O channels, is used primarily to hold data being actively used by the supercomputer. This file system is usually composed of forty or more high-speed disks, each capable of transferring up to 10 megabytes/second. Data are often striped across these disks, although no parity is computed. A typical system might stripe data across eight disks and transfer a single file at tens of megabytes per second. The local file system on the Cray Y/MP at the San Diego Supercomputer Center (SDSC), for example, averages 19 megabytes/second sustained over the course of a day, and allows individual transfers at a higher rate [7]. While these file systems can transfer data at very high speeds to their host supercomputer, they are not designed to service a large number of small file requests, and are rarely used as primary storage systems for a large number of client workstations. Instead, files for a particular application are copied onto the high-speed disks from the mass storage system before the application is run.

Mainframe-managed mass storage, on the other hand, provides large amounts of storage to both supercomputers and workstations. A typical system, such as the NASA Ames mass storage system MSS-II [14], provides both disk and tertiary storage to clients over a network. MSS-II uses an Amdahl 5880 as a file server, and distributes data across its disks using a scheme based on RAID level 5. Data from these disks may be transferred over one of several network channels. Like the supercomputer file system, however, MSS-II is not designed to service many small requests; rather, it is best suited for transferring entire files to and from its clients. Additionally, its bandwidth is lower than that of the file system managing high-speed disks; the goal for MSS-II was to provide 10 megabytes/second for individual files.

In RAID-II, we take advantage of recent technological developments that make it possible to build network file servers that can provide high bandwidth without the use of mainframes or expensive high-speed disks. Instead, we can use workstations, disk arrays, and standard interconnects and networks such as HIPPI and FDDI to build high performance network file servers at much lower cost. Like previous mainframe-based file servers for supercomputers, RAID-II provides high performance for large files. However, it can also service a large number of small file requests, and serve as a primary file server for client workstations.

The Los Alamos National Laboratory's High-Performance Data System (HPDS) [3] is an experimental prototype similar in design philosophy to RAID-II. Like RAID-II, it attempts to provide low latency on small transfers and control operations, as well as high bandwidth on large transfers. HPDS directly connects an IBM RAID Level 3 disk array to a HIPPI network and controls the data movement remotely (over an Ethernet) from an IBM RISC/6000. Los Alamos has demonstrated data rates close to the maximum data rate of the IBM disk array, approximately 60 megabytes/second. The main difference between LANL's High-Performance Data System and

RAID-II is that HPDS uses a bit-striped, or RAID Level 3, disk array, whereas RAID-II uses a flexible, crossbar interconnect that can support many different RAID architectures. In particular, RAID-II supports RAID Level 5, which supports many independent I/Os in parallel. RAID Level 3, on the other hand, supports only one I/O at a time, which will hurt the performance of small operations.

5 Future Directions

5.1 Planned Use of RAID-II

We plan to use RAID-II as a storage system for two new research projects at Berkeley. As a part of the Gigabit Test Bed project being conducted by the U.C. Berkeley Computer Science Division and Lawrence Berkeley Laboratory (LBL), RAID-II will act as a high-bandwidth video storage and playback server. Located at the LBL is an electron microscope with an attached video digitizer. The digitizer has a HIPPI interface that is connected into a switched HIPPI network. This network spans the two sites via fiber based HIPPI extenders. On this network along with RAID-II are Sun workstations and a MASPAR multiprocessor. We will capture the video stream from the microscope and store it on RAID-II. The image sequence can be enhanced on the MASPAR, stored again on RAID-II, and then played back from the disk array for frame by frame analysis of the molecular motion being studied.

The InfoPad project at U.C. Berkeley will use the RAID-II disk array as an information server. In conjunction with the real time protocols developed by Prof. Domenico Ferrari's research group, we will provide video storage and play back from the disk array to a network of base stations. These base stations in turn drive the radio transceivers of a pico-cellular network to which hand-held display devices, called InfoPads, will connect. The disk array will need to serve simultaneous streams to a number of users. Typical bandwidth per user is 1 megabyte/second over the radio link, and more over the backbone network.

5.2 Striping Across File Servers: The Zebra File System

The scalable nature of RAID-II's disk array allows the prototype's performance to be easily improved by increasing the number of disks in the array across which data are striped. There is a limit to the benefit of wider striping, however, since the XBUS board limits the rate at which data can be transferred between the network and the disk array. It is also possible to scale performance by adding more XBUS boards to the server, but doing so will not necessarily improve the performance of individual I/O requests. This is because each XBUS board has its own disk array and may have a separate HIPPI network interface. Clients must be careful to access data on a particular disk array via its corresponding network interface. Thus, a single RAID-II server with multiple XBUS boards appears to its clients as multiple independent servers, each with its own network address and storage system. Striping across XBUS boards implies that the clients are aware of the internal configuration of the RAID-II server and use the appropriate network interfaces to access data.

A storage architecture whose bandwidth for individual requests would be incrementally scalable is highly desirable because of the trend toward higher bandwidth workloads. Zebra [4] is a network file system designed to provide high-bandwidth file access by striping files across multiple file servers. Its use with RAID-II would provide a mechanism for striping high-bandwidth file accesses over multiple network connections, and therefore across multiple XBUS boards. Zebra incorporates ideas from both RAID and LFS: from RAID, the ideas of combining many relatively low-performance devices into a single high-performance logical device, and using parity to survive device failures; and from LFS the concept of treating the storage system as a log, so that small writes and parity updates are avoided.

There are two aspects of Zebra that make it particularly well-suited for use with RAID-II. First, the servers in Zebra perform very simple operations, merely storing blocks of the logical log of files without examining the content of the blocks. Little communication is needed between the XBUS board and the host CPU, allowing data to flow between the network and the disk array efficiently. The second benefit of using Zebra on RAID-II is that the log-structured nature of Zebra allows a client to write to the servers in large transfers, even if the application programs are doing small writes. Thus, Zebra avoids costly small writes to the disk array, allowing the disk array to be used efficiently without any extra effort on the part of the server.

6 Summary and Conclusions

Our first disk array prototype, RAID-I, performed well for small, random I/O operations. However, it was unable to deliver much of the available bandwidth from the disk array on large transfers because of limited memory bandwidth on the host workstation. We designed our second prototype with the goal of delivering much more of the available disk array bandwidth to file server clients.

RAID-II shares several features with existing high performance I/O systems. Like the Auspex NS5000, RAID-II divides functionality between the host workstation, XBUS board, and other boards like the TMC HIPPI interfaces. Data bypass the file server on large transfers and are transferred via a high-performance data path directly between the disk array and clients, as in the Los Alamos HPDS system. Also like HPDS, RAID-II separates data and control messages for high-bandwidth data accesses, sending control over a slow network, and data over a fast one.

The architectural features of the RAID-II Storage Architecture that allow it to perform well on high-bandwidth applications are:

- RAID-II uses a custom-built XBUS controller board to connect the disks and high-bandwidth network directly, unlike conventional network file servers that connect disks, memory and network through the host workstation's backplane and low-bandwidth memory bus. This allows RAID-II to scale in performance as network performance scales.

- RAID-II has two network attachments, and thus, two modes of access: *standard mode*, which sends data through the host workstation memory and over the Ethernet network, and *high-bandwidth mode*, which uses the XBUS board to bypass host memory and send data over the HIPPI network.

- The host workstation continues to control all data transfers, handles file metadata, translates human-readable file names to logical device addresses, and maintains a file cache in host memory.
- The host may control several XBUS disk array controller boards, so that bandwidth is not limited to that of a single controller board.

The distinctive implementation features of RAID-II are:

- RAID-II runs LFS, the Log-Structured File System, which lays out files in large contiguous segments to provide high-bandwidth access to large files, groups small write requests into large write requests to avoid wasting disk bandwidth, and provides fast crash recovery.
- The RAID-II file server uses two high-bandwidth HIPPI busses for data large transfers and a low-latency VME bus for small data transfers and control.
- The XBUS controller uses a 4×8 crossbar-based interconnect with a peak bandwidth of 160 megabytes/second. The crossbar connects four memory modules and eight XBUS ports, including two interfaces to the HIPPI network, four interaces to disk controller boards, a parity computation engine, and a VME control bus used to send commands between the host workstation and the XBUS board.

The main performance results are:

- RAID-II using a single XBUS controller board achieves about 20 megabytes/second for read and write operations using all the components of the hardware. This performance is an order of magnitude improvement compared to our first prototype, but disappointing compared to our performance goal of 40 megabytes/second. The performance is limited by the SCSI disk controller boards and by the bandwidth capabilities of our disk interface ports. Other components of the system achieve their bandwidth goal of 40 megabytes/second, including the HIPPI interfaces, memory modules and parity engine.

- A preliminary implementation of LFS achieves 13.4 megabytes/second from RAID-II configured with a single XBUS board into HIPPI network buffers on the XBUS board.

The RAID-II disk array prototype succeeds in delivering much higher bandwidth than its predecessor, RAID-I. We accomplish this using hardware and software specifically designed to preserve the bandwidth available from the disks. Until workstation memory bandwidth improves considerably, we believe that future workstation-based file servers designed to provide high bandwidth I/O over networks should include a separate high-bandwidth access path like the XBUS board to connect the disks and the network directly. The use of LFS also optimizes the performance of the disk array for bandwidth-intensive applications.

7 Acknowledgments

We would like to thank the other members of the RAID and Sprite groups at Berkeley for their contributions to RAID-II. In particular we would like to thank John Ousterhout, Mendel Rosenblum, Rob Pfile, Rob Quiros, Richard Drewes and Mani Varadarajan.

This research was supported by Array Technologies, DARPA/NASA (NAG2-591), DEC, Hewlett-Packard, IBM, Intel Scientific Computers, California MICRO, NSF (MIP 8715235), Seagate, Storage Tek, Sun Microsystems and Thinking Machines Corporation.

References

[1] Peter M. Chen, Edward K. Lee, Ann L. Drapeau, Ken Lutz, Ethan L. Miller, Srinivasan Seshan, Ken Shirriff, David A. Patterson, and Randy H. Katz. Performance and Design Evaluation of the RAID-II Storage Server. *International Parallel Processing Symposium Workshop on I/O in Parallel Computer Systems*, April 1993. invited for submission to the Journal of Distributed and Parallel Databases.

[2] Ann L. Chervenak and Randy H. Katz. Perfomance of a disk array prototype. In *Proceedings SIGMETRICS*, May 1991.

[3] Bill Collins. High-performance data systems. In *Proc. IEEE Symposium on Mass Storage Systems*, October 1991.

[4] John H. Hartman and John K. Ousterhout. The zebra striped network file system. In *to appear in Proceedings of the 14th ACM Symposium on Operating System Principles*, December 1993.

[5] John L. Hennessy and Norman P. Jouppi. Computer technology and architecture: An evolving interaction. *IEEE Computer*, 24:18–29, September 1991.

[6] William A. Horton and Bruce Nelson. The Auspex NS 5000 and the SUN SPARCserver 490 in One and Two Ethernet NFS Performance Comparisons. Technical Report Auspex Performance Report 2, Auspex, May 1990.

[7] Reagan W. Moore. File servers, networking, and supercomputers. Technical Report GA-A20574, San Diego Supercomputer Center, July 1991.

[8] Bruce Nelson. An overview of functional multiprocessing for NFS network servers. Technical Report 1, Auspex Engineering, 1990.

[9] John K. Ousterhout. Why aren't operating systems getting faster as fast as hardware. In *Proceedings USENIX Technical Conference*, June 1990.

[10] David A. Patterson, Garth Gibson, and Randy H. Katz. A case for redundant arrays of inexpensive disks (RAID). In *Proceedings ACM SIGMOD*, pages 109–116, June 1988.

[11] Mendel Rosenblum and John Ousterhout. The design and implementation of a log-structured file system. In *Proc. ACM Symposium on Operating Systems Principles*, pages 1–15, October 1991.

[12] Russel Sandberg, David Goldbert, Steve Kleiman, Dan Walsh, and Bob Lyon. Design and Implementation of the Sun Network Filesystem. In *Summer 1985 Usenix Conference*, 1985.

[13] Daniel Stodolsky, Garth Gibson, and Mark Holland. Parity logging overcoming the small write problem in redundant disk arrays. In *Proc. International Symposium on Computer Architecture*, pages 64–75, May 1993.

[14] David Tweten. Hiding mass storage under Unix: NASA's MSS-II architecture. In *Proc. IEEE Symposium on Mass Storage Systems*, pages 140–145, May 1990.