RAID-II: A High-Bandwidth Network File Server

Ann L. Drapeau   Ken W. Shirriff   John H. Hartman   Ethan L. Miller
Srinivasan Seshan   Randy H. Katz   Ken Lutz   David A. Patterson[1]

Edward K. Lee[2]   Peter M. Chen[3]   Garth A. Gibson[4]

Abstract

In 1989, the RAID (Redundant Arrays of Inexpensive Disks) group at U. C. Berkeley built a prototype disk array called RAID-I. The bandwidth delivered to clients by RAID-I was severely limited by the memory system bandwidth of the disk array's host workstation. We designed our second prototype, RAID-II, to deliver more of the disk array bandwidth to file server clients. A custom-built crossbar memory system called the XBUS board connects the disks directly to the high-speed network, allowing data for large requests to bypass the server workstation. RAID-II runs Log-Structured File System (LFS) software to optimize performance for bandwidth-intensive applications.

The RAID-II hardware with a single XBUS controller board delivers 20 megabytes/second for large, random read operations and up to 31 megabytes/second for sequential read operations. A preliminary implementation of LFS on RAID-II delivers 21 megabytes/second on large read requests and 15 megabytes/second on large write operations.

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 databases and scientific visualization. Even in well-established application areas such as scientific computing, the size of data sets is growing rapidly due to 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 [13], [5], use a collection of relatively small, inexpensive disks to achieve high performance and high reliability in a 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 multiple disk 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 megabytes of memory, four dual-string SCSI controllers, 28 5.25-inch SCSI disks and specialized disk striping software. We were interested in the performance of a disk array constructed of commercially-available components on two workloads: small, random operations typical of file servers in a workstation environment and large transfers typical of bandwidth-intensive applications. We anticipated that there would be both hardware and software bottlenecks that would restrict the bandwidth achievable by RAID-I. The development of RAID-I was motivated by our desire to gain experience with disk array software, disk controllers and the SCSI protocol.

Experiments with RAID-I show that it performs well when processing small, random I/O's, achieving approximately 275 four-kilobyte random I/Os per second. However, RAID-I proved 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. The bandwidth of nearly 26 of 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 workstation during I/O operations. The copy operations that move data between 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 restricted 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 workstations that are designed to exploit large, fast caches for good processor performance, but fail to support adequate primary memory or I/O bandwidth [8], [12]. In such

[1] Computer Science Division, University of California, Berkeley, CA 94720

[2] DEC Systems Research Center, Palo Alto, CA 94301-1044

[3] Computer Science and Engineering Division, University of Michigan, Ann Arbor, MI 48109-2122

[4] School of Computer Science, Carnegie Mellon University, Pittsburgh, PA 15213-3891

workstations, the memory system is designed so that only the CPU has a fast, high-bandwidth path to memory. For busses or backplanes farther away from the CPU, available memory bandwidth drops quickly. Thus, file servers incorporating such workstations perform poorly on bandwidth-intensive applications.

The design of hardware and software for our second prototype, RAID-II [1], was motivated by a desire to deliver more of the disk array's bandwidth to the file server's clients. Toward this end, the RAID-II hardware contains two data paths: a high-bandwidth path that handles large transfers efficiently using a custom-built crossbar interconnect between the disk system and the high-bandwidth network, and a low-bandwidth path used for control operations and small data transfers. The software for RAID-II was also designed to deliver disk array bandwidth. RAID-II runs LFS, the Log-Structured File System [14], developed by the Sprite operating systems group at Berkeley. LFS treats the disk array as a log, combining small accesses together and writing large blocks of data sequentially to avoid inefficient small write operations. A single XBUS board with 24 attached disks and no file system delivers up to 20 megabytes/second for random read operations and 31 megabytes/second for sequential read operations. With LFS software, the board delivers 21 megabytes/second on sequential read operations and 15 megabytes/second on sequential write operations. Not all of this data is currently delivered to clients because of client and network limitations described in Section 3.4.

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 deliver bandwidth, architecture and implementation details, and hardware 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 storage server prototype spans three racks, with the two outer racks containing disk drives and the center rack containing custom-designed crossbar controller boards and commercial disk controller boards. The center rack also contains a workstation, called the host, which is shown in the figure as logically separate. Each XBUS controller board has a HIPPI connection to a high-speed Ultra Network Technologies ring network that may also connect supercomputers and client workstations. The host workstation has an Ethernet interface that allows transfers between the disk array and clients connected to the Ethernet network.

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

2.1 Delivering Disk Array Bandwidth

2.1.1 High and Low Bandwidth Data Paths

Disk array bandwidth on our first prototype was limited by low memory system bandwidth on the host workstation. RAID-II was designed to avoid similar performance limitations. It uses a custom-built memory system called the XBUS controller to create a high-bandwidth data path. This data path allows RAID-II to transfer data directly between the disk array and the high-speed, 100 megabytes/second HIPPI network without sending data through the host workstation memory, as occurs in traditional disk array storage servers like RAID-I. Rather, data are temporarily stored in memory on the XBUS board and transferred using a high-bandwidth crossbar interconnect between disks, the HIPPI network, the parity computation engine and memory.

There is also a low-bandwidth data path in RAID-II that is similar to the data path in RAID-I. This path transfers data across the workstation's backplane between the XBUS board and host memory. The data sent along this path include file metadata (data associated with a file other than its contents) and small data transfers. Metadata are needed by file system software for file management, name translation and for maintaining consistency between file caches on the host workstation and the XBUS board. The low-bandwidth data path is also used for servicing data requests from clients on the 10 megabits/second Ethernet network attached to the host workstation.

We refer to the two access modes on RAID-II as the *high-bandwidth mode* for accesses that use the high-performance data path and *standard mode* for requests serviced by the low-bandwidth data path. Any client request can be serviced using either access mode, but we maximize utilization and performance of the high-bandwidth data path if smaller requests use the Ethernet network and larger requests use the HIPPI network.

2.1.2 Scaling to Provide Greater Bandwidth

The bandwidth of the RAID-II storage server can be scaled by adding XBUS controller boards to a host workstation. 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.

Eventually, adding XBUS controllers to a host workstation will saturate the host's CPU, since the host manages all disk and network transfers. However, since the high-bandwidth data path of RAID-II is independent of the host workstation, we can use a more powerful host to continue scaling storage server bandwidth.

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 connects the high-bandwidth network interfaces, several XBUS controllers and a host workstation that controls the operation of the XBUS controller boards. Each XBUS board contains interfaces to the

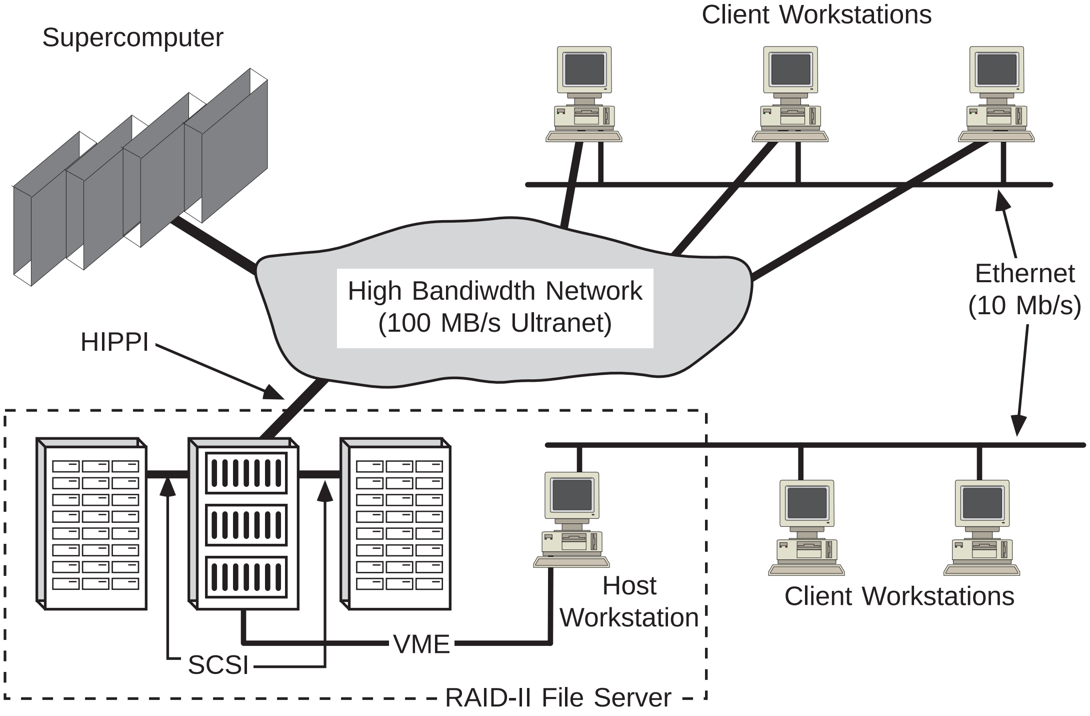

Figure 1: The RAID-II File Server and its Clients. RAID-II, shown by dotted lines, is composed of three racks. The host workstation is connected to one or more XBUS controller boards (contained in the center rack of RAID-II) over the VME backplane. Each XBUS controller has a HIPPI network connection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

HIPPI backplane and VME control bus, memory, a parity computation engine and four VME interfaces that connect to disk controller boards.

Figure 3 illustrates the physical packaging of the RAID-II file server, which spans 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 DRAM. The center rack is composed of three chassis. On top 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 3.5-inch, 320 megabyte IBM SCSI disks and their power supplies. Each of these racks contains eight shelves of nine disks. RAID-II has a total capacity of 46 gigabytes, although the performance results in the next section are for a single XBUS board controlling 24 disks. We will achieve full array connectivity by adding XBUS boards and increasing the number of disks per SCSI string.

Figure 4 is a block diagram of the XBUS disk array controller board, which implements a 4x8, 32-bit wide crossbar interconnect called the XBUS. The crossbar connects four memory modules to eight system components called *ports*. The XBUS ports include two interfaces to HIPPI network boards, four VME interfaces to Interphase Cougar disk controller boards, a parity computation engine, and a VME interface to the host workstation. Each XBUS transfer involves one of the four memory modules and one of the eight XBUS ports. Each port was intended to support 40 megabytes/second of data transfer for a total of 160 megabytes/second of sustainable XBUS bandwidth.

The XBUS is a synchronous, multiplexed (address/data), crossbar-based interconnect that uses a centralized, priority-based arbitration scheme. Each of the eight 32-bit XBUS ports operates at a cycle time of 80 nanoseconds. The memory is interleaved in sixteen-word blocks. The XBUS supports only two types of bus transactions: reads and writes. Our implementation of the crossbar interconnect was fairly expensive, using 192 16-bit transceivers. Using surface mount packaging, the implementation required 120 square inches or approximately 20% of the XBUS controller's board area. An advantage of the implementation is that the 32-bit XBUS ports are relatively inexpensive.

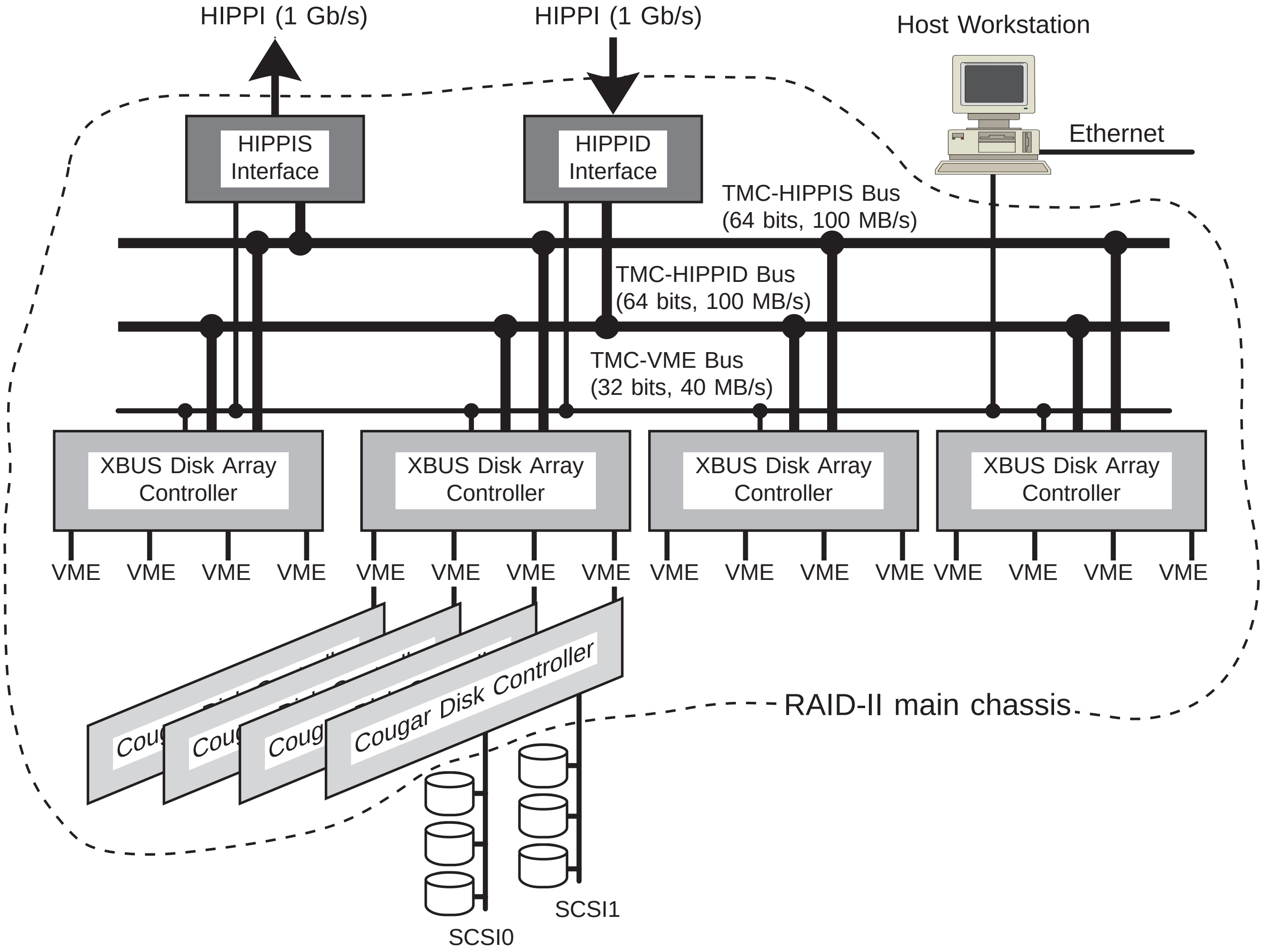
HIPPI (1 Gb/s)
HIPPI (1 Gb/s)
Host Workstation
Ethernet
HIPPIS Interface
HIPPID Interface
TMC-HIPPIS Bus (64 bits, 100 MB/s)
TMC-HIPPID Bus (64 bits, 100 MB/s)
TMC-VME Bus (32 bits, 40 MB/s)
XBUS Disk Array Controller
XBUS Disk Array Controller
XBUS Disk Array Controller
XBUS Disk Array Controller
VME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

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, buffer memory, a high-bandwidth crossbar, a parity engine, and interfaces to four SCSI disk controller boards.

Two XBUS ports implement an interface between the XBUS and the TMC HIPPI boards. Each port is unidirectional, designed to sustain 40 megabytes/second of data transfer and bursts of 100 megabytes/second into 32 kilobyte 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. The Cougar disk controllers can transfer data at 8 megabytes/second, for a maximum disk bandwidth to a single XBUS board of 32 megabytes/second in our present configuration.

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, file data and metadata in the XBUS memory.

2.3 Hardware Performance

In this section, we present raw performance of the RAID-II [1] hardware, that is, the performance of the hardware without the overhead of a file system. In Section 3.4, we show how much of this raw hardware performance can be delivered by the file system to clients.

Figure 5 shows RAID-II performance for random reads and writes. We refer to these performance measurements as hardware 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

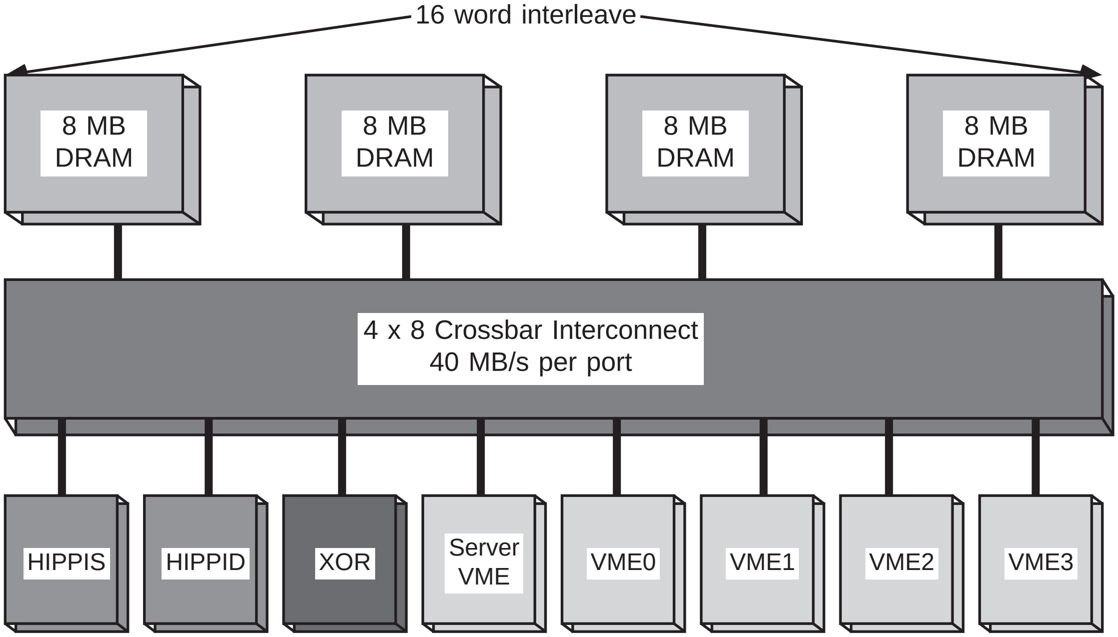

Figure 4: Structure of XBUS controller board. The board contains four memory modules connected by a 4x8 crossbar interconnect to eight XBUS ports. Two of these XBUS ports are interfaces to the HIPPI network. Another is a parity computation engine. One is a VME network interface for sending control information,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

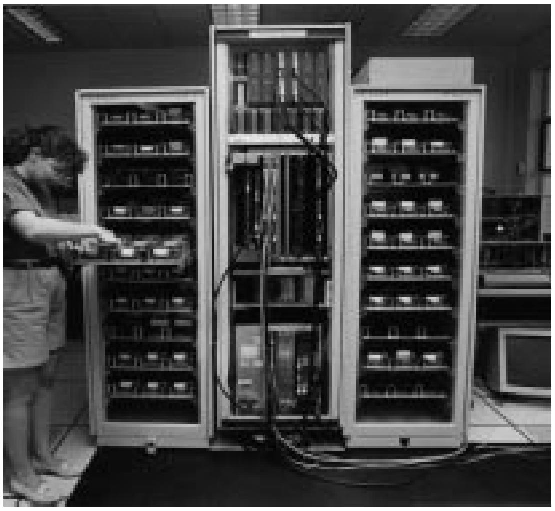

Figure 3: The physical packaging of the RAID-II File Server. Two outer racks contain 144 disks and their power supplies. The center rack contains three chassis: the top chassis holds VME disk controller boards; the center chassis contains XBUS controller boards and HIPPI interface boards, and the bottom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

configured as a RAID Level 5 [13] with one parity group of 24 disks. (This scheme delivers high bandwidth but exposes the array to data loss during dependent failure modes such as a SCSI controller failure. Techniques for maximizing reliability are beyond the scope of this paper [4], [16], [6].)

For reads, data are read from the disk array into the memory on the XBUS board; from there, data 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 fixed size operations are at random locations. Figure 5 shows that, for large requests, hardware system level read and write performance reaches about 20 megabytes/second. The dip in read performance for requests of size 768 kilobytes occurs because at that size the striping scheme in this experiment involves a second string on one of the controllers; there is some contention on the controller that results in lower performance when both strings are used. 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

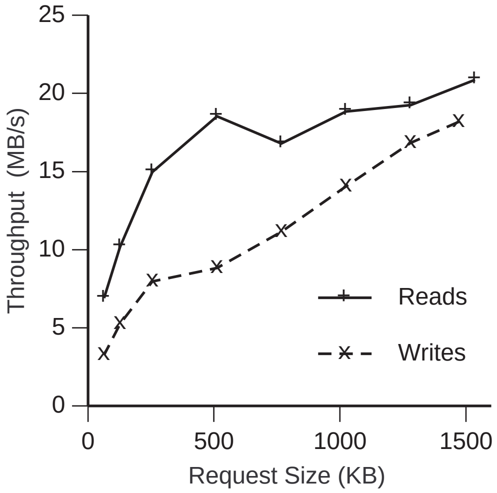

Figure 5: Hardware System Level Read and Write Performance. RAID-II achieves approximately 20 megabytes/second for both random reads and writes.

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. This experiment was run using the four XBUS VME interfaces to disk controllers boards, with a fifth disk controller attached to the XBUS 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-II can perform more I/Os per second than the Seagate Wren IV disks used in RAID-I because they have shorter seek and rotation times. Superior I/O rates on RAID-II are also the result of the architecture, which does not require data to be transferred through the host workstation.

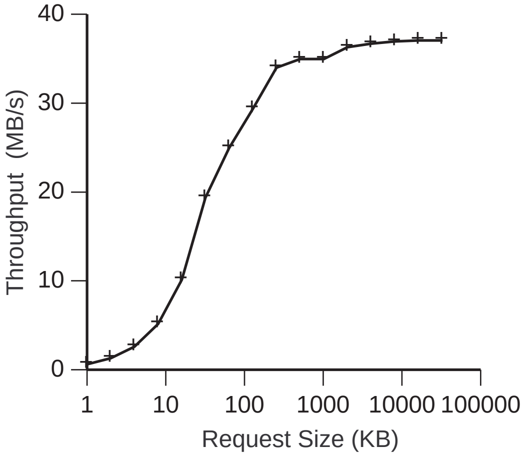

Figure 6: HIPPI Loopback Performance. RAID-II achieves 38.5 megabytes/second in each direction.

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, using a fifth Cougar board attached to the XBUS VME control bus interface. For requests of size 1.6 megabytes, read performance is 31 megabytes/second, compared to 23 megabytes/second for writes. Write performance is worse than read performance because of the extra accesses required during writes to calculate and write parity and because sequential reads benefit from the read-ahead performed into track buffers on the disks; writes have no such advantage on these disks.

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 on our two disk array prototypes, RAID-I and RAID-II, using a test program that performed random 4 kilobyte reads. In each case, fifteen disks were accessed. The tests used Sprite operating system read and write system calls; the RAID-II test did not use LFS. In each case, a separate 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, with RAID-II achieving over 400 small accesses per second. In both prototypes, the small, random I/O rates were limited by the large number of context switches required on the Sun4/280 workstation to handle request completions. RAID-I's performance was also limited by the memory bandwidth on the Sun4/280, since all data went into host memory. Since data need not be transferred to the host workstation in RAID-II, it delivers a higher percentage (78%) of the potential I/O rate from its fifteen disks than does RAID-I (67%).

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, and then 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 interfaces' raw hardware performance; it does not include measurements for sending data over the Ultranet. In the loopback mode, the overhead of sending a HIPPI packet is about 1.1 milliseconds, mostly due to setting up the HIPPI and XBUS control registers

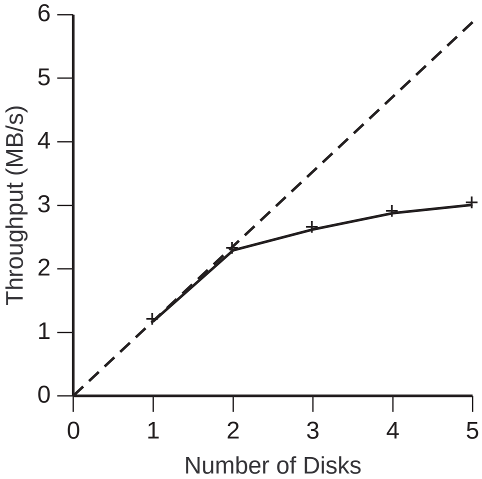

Figure 7: Disk read performance for varying number of disks on a single SCSI string. Cougar string bandwidth is limited to about 3 megabytes/second, less than that of three disks. The dashed line indicates the performance if bandwidth scaled linearly.

across the slow VME link. (By comparison, an Ethernet packet takes approximately 0.5 millisecond to transfer.) Due to this control overhead, small requests result in low performance on the HIPPI network. 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 the XBUS ports. The HIPPI loopback bandwidth of 38 megabytes/second is three times greater than FDDI and two orders of magnitude greater than Ethernet. Clearly the HIPPI part of the XBUS is not the limiting factor in determining hardware system level performance.

Disk performance is responsible for the lower-than-expected hardware system level performance of RAID-II. One performance limitation is the Cougar disk controller, which only supports about 3 megabytes/second on each of two SCSI strings, as shown in Figure 7. The other limitation is 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. This low performance is largely due to the difficulty of synchronizing the asynchronous VME bus for interaction with the XBUS.

3 The RAID-II File System

In the last section, we presented measurements of the performance of the RAID-II hardware without the overhead of a file system. Although some applications may use RAID-II as a large raw disk, most 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) [14]. In this section, we first describe LFS and explain why it is particularly well-suited for use with disk arrays. Second, we describe the features of RAID-II that require special attention in the LFS implementation. Next, we illustrate how LFS on RAID-II handles 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 [14] 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. LFS minimizes the number of 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 avoids small write operations, the Log-Structured File System avoids a weakness of redundant disk arrays that affects traditional file systems. Under a traditional file system, disk arrays that use large block interleaving (Level 5 RAID [13]) perform poorly on small write operations because each small write requires four disk accesses: reads of the old data and parity blocks and writes of the new data and parity blocks. By contrast, large write operations in disk arrays are efficient since they don't require the reading of old data or parity. LFS eliminates small writes, grouping them into efficient large, sequential write operations.

A secondary benefit of LFS is that crash recovery is very quick. LFS periodically performs *checkpoint* operations that record the current state of the file system. To recover from a file system crash, the LFS server need only process 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 gigabyte file system, it takes a few seconds 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 data paths, and the separate memories on the host workstation and the XBUS board.

To handle the two data paths, we changed the original Sprite LFS software to separate the handling of data and metadata. We use the high-bandwidth data path to transfer data between the disks and the HIPPI network, bypassing host workstation memory. The low-bandwidth VME connection between the XBUS and the host workstation is used to send metadata to the host for file name lookup and location of data on disk. The low-bandwidth path is also used for small data transfers to clients on the Ethernet attached to the host.

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, pipelining buffers, HIPPI network buffers, and 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 XBUS memory to pipeline network and disk operations. For read operations, while one block of data is being sent across the network, the next blocks are being read off the disk. For write operations, full LFS segments are written to disk while newer segments are being filled with data. We are also experimenting with prefetching techniques so small sequential reads can also benefit from overlapping disk and network operations.

From the user's perspective, the RAID-II file system looks like a standard file system to clients using the slow Ethernet path (standard mode), but requires relinking of applications for clients to use the fast HIPPI path (high-bandwidth mode). To access data over the Ethernet, clients simply use NFS or the Sprite RPC mechanism. The fast data path 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. Finally, some operating systems offer dynamically loadable device drivers that would eliminate the need for relinking applications; however, Sprite does not allow dynamic loading.

3.3 Typical High and Low Bandwidth Requests

This section explains how the RAID-II file system handles typical open and read requests from a client over the fast Ultra Network Technologies network 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 that 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 to RAID-II. Finally, the RAID-II host 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 pipeline 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 the handling of 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 workstation sends the read command 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

This section describes the performance of LFS running on RAID-II. These measurements show that LFS performs efficiently on large random read and write operations, delivering about 65% of the raw bandwidth of the hardware. LFS also achieves good performance on small random write operations by grouping small writes into larger segments.

All the measurements presented in this section use a single XBUS board with 16 disks. The LFS log is interleaved or "striped" across the disks in units of 64 kilobytes. The log is written to the disk array in units or "segments" of 960 kilobytes. The file system for RAID-II is still under development; it currently handles creation, reads, and writes of files, but LFS cleaning (garbage collection of free disk space in the log) has not yet been implemented. Because we don't have sufficiently fast network clients, many of the file server measurements in this section don't actually send data over the network; rather, data are written to or read from network buffers in XBUS memory.

Figure 8 shows performance for random read and write operations for RAID-II running LFS. For each request type, a single process issued requests to the disk array. For both reads and writes, data are transferred to/from network buffers, but do not actually go across the network. The figure shows that for random read requests larger than 10 megabytes, which are effectively sequential in nature, the file system delivers up to 20 megabytes/second, approximately 68% of the raw sequential hardware bandwidth described in Table 1. Smaller random reads have lower bandwidth due to an average overhead of 23 milliseconds per operation: 4 milliseconds of file system overhead and 19 milliseconds of disk overhead. The file system overhead should be reduced substantially as LFS performance is tuned.

The graph also shows random write performance. For random write requests above approximately 512 kilobytes in size, the file system delivers close to its maximum value of 15 megabytes/second. This value is approximately 65% of the raw sequential write performance reported in Table 1. Random writes perform as well as sequential writes for much lower request sizes than was the case for read operations. This is because LFS groups together random write operations and writes them to the log sequentially. The bandwidth for small random write operations is better than bandwidth for small random reads, suggesting that LFS has been successful in mitigating the penalty for small writes on

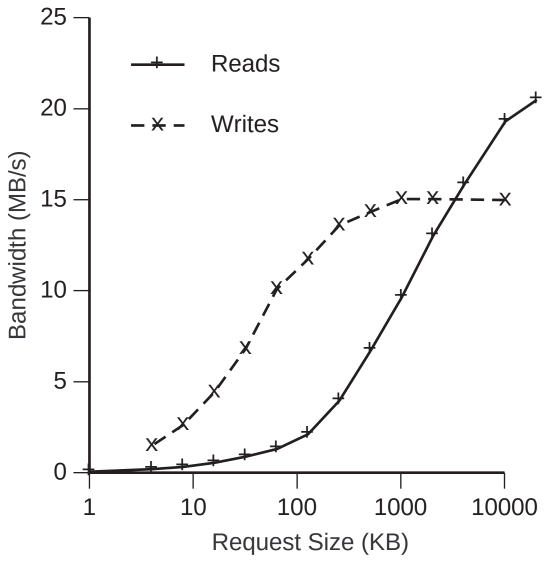

Figure 8: Performance of RAID-II running the LFS file system.

a RAID Level 5 architecture. Small write performance is, however, limited because of approximately 3 milliseconds of network and file system overhead per request.

Figure 8 showed RAID-II performance without sending data over the network. Next we consider network performance for reads and writes to a single client workstation; we have not yet run these tests for multiple clients. Performance on write operations to RAID-II is limited by the client; we currently don't have a client that can deliver enough bandwidth to saturate the disk array. A SPARCstation 10/51 client on the HIPPI network writes data to RAID-II at 3.1 megabytes per second. Bandwidth is limited on the SPARCstation because its user-level network interface implementation performs many copy operations. RAID-II is capable of scaling to much higher bandwidth; utilization of the Sun4/280 workstation due to network operations is close to zero with the single SPARCstation client writing to the disk array.

Performance on network reads from the disk array is limited by our initial network device driver implementation. To perform a data transfer, we must synchronize operation of the HIPPI source board and the host. Although the source board is able to interrupt the host, this capability was not used in the initial device driver implementation. Rather, for reasons of simplicity, the host workstation waits while data are being transmitted from the source board to the network, checking a bit that indicates when the data transfer is complete. This rather inefficient implementation limits RAID-II read operations for a single SPARCstation client to 3.2 megabytes/second. In the implementation currently being developed, the source board will interrupt the CPU when a transfer is complete. With this modification, we expect to deliver the full file system bandwidth of RAID-II to clients on the network, although the network implementation on the SPARCstation will still limit the bandwidth to an individual workstation client.

4 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 File Servers for Workstations

File servers for a workstation computing environment are not designed to provide high-bandwidth file access to individual files. Instead, they are designed to provide low-latency service to a large number of clients via NFS [15]. A typical NFS file server is a UNIX workstation that is transformed into a server by adding memory, network interfaces, disk controllers, and possibly non-volatile memory to speed up NFS writes. All workstation vendors offer NFS servers of this type. The problem with using a workstation as a file server is that all data transfers between the network and the disks must pass across the workstation's backplane, which can severely limit transfer bandwidth. It has been shown that the bandwidth improvements in workstation memory and I/O systems have not kept pace with improvements in processor speed [12].

One way of achieving higher performance than is possible from a UNIX workstation server is to use special-purpose software and hardware. An example is the FAServer NFS file server [9]. The FAServer does not run UNIX; instead it runs a special-purpose operating system and file system tuned to provide efficient NFS file service. This reduces the overhead of each NFS operation and allows the FAServer to service many more clients than a UNIX-based server running on comparable hardware. The underlying hardware for the FAServer is a standard workstation, however, so the FAServer suffers from the same backplane bottleneck as a UNIX workstation when handling large transfers.

Another special-purpose NFS file server is the Auspex NS6000 [11]. The Auspex is a functional multi-processor, meaning that the tasks required to process an NFS request are divided among several processors. Each processors runs its own special-purpose software, tailored to the task it must perform. The processors are connected together and to a cache memory by a high-speed bus running at 55 megabytes/second. The Auspex does contain a host workstation that runs UNIX, but it is not involved in serving of NFS requests. File data are transferred between the network and the disks over the high-speed bus without being copied into the workstation memory. In this sense the Auspex architecture is similar to that of RAID-II, but it differs in the use of special-purpose processors to handle requests.

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 consists of many fast parallel transfer disks directly attached to a supercomputer's I/O channels. Each high-speed disk might transfer at a rate of 10 megabytes/second; typically, data might be striped across 40 of these disks. Data are usually copied onto these high-speed disks before an application is run on the supercomputer [10]. These disks do not provide permanent storage for users. The file system is not designed to provide efficient small transfers.

The other type of supercomputer file system is the mainframe-managed mass storage system. A typical system, such as the NASA Ames mass storage system NAStore [17], provides both disk and tertiary storage to clients over a network. The bandwidth of such systems is typically lower than that of the parallel transfer disk systems. NAStore, for example, sustains 10 megabytes/second to individual files.

RAID-II was designed to perform efficiently on small file accesses as well as on large transfers. A system similar in design philosophy to RAID-II is Los Alamos National Laboratory's High Performance Data System (HPDS) [3]. Like RAID-II, HPDS attempts to provide low latency on small transfers and control operations as well as high bandwidth on large transfers. HPDS connects an IBM RAID Level 3 disk array to a HIPPI network; like RAID-II, it has a high-bandwidth data path and a low-bandwidth control path. The main difference between HPDS and RAID-II is that HPDS uses a bit-interleaved, 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 can execute several small, independent I/Os in parallel. RAID Level 3, on the other hand, supports only one small I/O at a time.

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 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. Data collected from an electron microscope at LBL will be sent from a video digitizer across an extended HIPPI network for storage on RAID-II and processing by a MASPAR multiprocessor.

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 Tenet research group, RAID-II 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 will connect.

5.2 Striping Across File Servers: The Zebra File System

Zebra [7] 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 would be needed between the XBUS board and the host workstation, allowing data to flow between the network and the disk array efficiently. Second, Zebra, like LFS, treats the disk array as a log, eliminating inefficient small writes.

6 Conclusions

The RAID-II disk array prototype was designed to deliver much more of the available disk array bandwidth to file server clients than was possible in our first prototype. The most important feature of the RAID-II hardware is the high-bandwidth data path that allows data to be transferred between the disks and the high-bandwidth network without passing through the low-bandwidth memory system of the host workstation. We implement this high-bandwidth data path with a custom-built controller board called the XBUS board. The XBUS controller uses a 4×8 crossbar-based interconnect with a peak bandwidth of 160 megabytes/second. File system software is also essential to delivering high bandwidth. 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. We are pleased with the reliability of the XBUS controller and disk subsystem. The system has been in continuous operation for six months with no failures.

Performance results for the RAID-II hardware and the Log Structure File System are good. The RAID-II hardware with a single XBUS controller board achieves about 20 megabytes/second for random read and write operations and 31 megabytes/second for sequential read operations. This performance is an order of magnitude improvement compared to our first prototype, but somewhat 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. RAID-II performs well on small, random I/O operations, achieving approximately 400 four-kilobyte random I/Os per second to fifteen disks. A preliminary implementation of LFS delivers 21 megabytes/second on large random read operations and 15 megabytes/second for moderately large random write operations into HIPPI network buffers on the XBUS board from RAID-II configured with a single XBUS board. Preliminary network performance shows that RAID-II can read and write 3 megabytes/second to a single SPARCstation client. This network interface is currently being tuned; we soon expect to deliver the full RAID-II file system bandwidth to clients.

If the RAID-II project were developed with the network hardware available today, ATM network interfaces would likely be a better choice than HIPPI. The use of HIPPI in RAID-II had several disadvantages. Connections to HIPPI remain very expensive, especially when the cost of switches needed to implement a network is included. While the bandwidth capabilities of HIPPI are a good match for supercomputer clients, the overhead of setting up transfers makes HIPPI a poor match for supporting a large number of client workstations.

Finally, unless workstation memory bandwidth improves considerably, we believe a separate high-bandwidth data path like the XBUS board that connects the disks and the network directly is necessary to deliver high bandwidth I/O from a workstation-based network file server. The use of Log Structured File System software maximizes the performance of the disk array for bandwidth-intensive applications.

Acknowledgments

We would like to thank John Ousterhout for his suggestions for improving this paper and his participation in the development of RAID-II. We would also like to thank Mendel Rosenblum, Mary Baker, Rob Pfile, Rob Quiros, Richard Drewes and Mani Varadarajan for their contributions to the success of RAID-II. Special thanks to Theresa Lessard-Smith and Bob Miller.

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 Technology Corporation, 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. Also invited for submission to the *Journal of Distributed and Parallel Databases*, to appear.

[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 *Digest of Papers*. Eleventh IEEE Symposium on Mass Storage Systems, October 1991.

[4] Garth A. Gibson, R. Hugo Patterson, and M. Satyanarayanan. Disk Reads with DRAM Latency. *Third Workshop on Workstaion Operating Systems*, April 1992.

[5] Garth Alan Gibson. *Redundant Disk Arrays: Reliable, Parallel Secondary Storage*. PhD thesis, U. C. Berkeley, April 1991. Technical Report No. UCB/CSD 91/613.

[6] Jim Gray, Bob Horst, and Mark Walker. Parity Striping of Disc Arrays: Low-Cost Reliable Storage with Acceptable Throughput. In *Proceedings Very Large Data Bases*, pages 148–161, 1990.

[7] John H. Hartman and John K. Ousterhout. The Zebra Striped Network File System. In *Proceedings of the 14th ACM Symposium on Operating Systems Principles*, December 1993.

[8] John L. Hennessy and Norman P. Jouppi. Computer Technology and Architecture: An Evolving Interaction. *IEEE Computer*, 24:18–29, September 1991.

[9] David Hitz. An NFS File Server Appliance. Technical Report Technical Report TR01, Network Appliance Corporation, August 1993.

[10] Reagan W. Moore. File Servers, Networking, and Supercomputers. Technical Report GA-A20574, San Diego Supercomputer Center, July 1991.

[11] Bruce Nelson. An Overview of Functional Multiprocessing for NFS Network Servers. Technical Report Technical Report 1, Auspex Engineering, July 1990.

[12] John K. Ousterhout. Why Aren't Operating Systems Getting Faster as Fast as Hardware. In *Proceedings USENIX Technical Conference*, June 1990.

[13] 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.

[14] 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.

[15] 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.

[16] Martin Schulze, Garth Gibson, Randy H. Katz, and David A. Patterson. How Reliable is a RAID? In *Proceedings IEEE COMPCON*, pages 118–123, Spring 1989.

[17] David Tweten. Hiding Mass Storage Under UNIX: NASA's MSS-II Architecture. In *Digest of Papers*. Tenth IEEE Symposium on Mass Storage Systems, May 1990.