Performance and Design Evaluation of the RAID-II Storage Server

Peter M. Chen, Edward K. Lee, Ann L. Drapeau, Ken Lutz, Ethan L. Miller, Srinivasan Seshan, Ken Shirriff, David A. Patterson, Randy H. Katz

Abstract: *RAID-II is a high-bandwidth, network-attached storage server designed and implemented at the University of California at Berkeley. In this paper, we measure the performance of RAID-II and evaluate various architectural decisions made during the design process. We first measure the end-to-end performance of the system to be approximately 20 MB/s for both disk array reads and writes. We then perform a bottleneck analysis by examining the performance of each individual subsystem and conclude that the disk subsystem limits its performance. By adding a custom interconnect board with a high-speed memory and bus system and parity engine, we are able to achieve a performance speedup of 8 to 15 over a comparative system using only off-the-shelf hardware.*

1 Introduction

RAID-II is a high-bandwidth, network file server designed and implemented at the University of California at Berkeley as part of a project to study high-performance, large-capacity, highly-reliable storage systems. RAID-II is designed for the heterogeneous computing environments of the future, consisting of diskless supercomputers, visualization workstations, multi-media platforms, and UNIX workstations.

Other papers [Lee92, Katz93] discuss in detail the architecture and implementation of RAID-II. The goal of this paper is to evaluate the decisions made in designing RAID-II. We evaluate those decisions by measuring the performance of RAID-II and its components and comparing it to the performance of RAID-I, a prototype network file server assembled at U.C. Berkeley using off-the-shelf parts. We are particularly interested in evaluating the novel architectural features of RAID-II, which are the crossbar-based interconnect, the high-speed data path between the network and the disk system, the separate network path for small and large file accesses, and the exclusive-or unit for fast parity computation.

The rest of the paper is organized as follows. Section 1 provides motivation and reviews previous work; Section 2 describes RAID-II's architecture and highlights its novel features; Section 3 reports the performance of RAID-II at the component and system level and draws conclusions about our design decisions.

1.1 Motivation

The development of RAID-II is motivated by three key observations. First, we notice a trend toward bandwidth-intensive applications: multi-media, CAD, large-object databases, and scientific visualization. Even in well established application areas such as scientific computing, reductions in the cost of secondary storage and the introduction of faster supercomputers have caused a rapid growth in the size of datasets, requiring faster I/O systems to transfer the increasing amounts of data.

The second observation is that most of today's workstation-based file servers are incapable of supporting high-bandwidth I/O. This was demonstrated in experience with our first prototype, RAID-I. Moreover, the future I/O performance of server workstations is likely to degrade relative to the overall performance of their client workstations even if applications do not become more I/O-intensive. This is because today's workstations achieve high performance by using large, fast caches without significantly improving the performance of the primary memory and I/O systems. This problem is mentioned by Hennessy and Jouppi [Hennessy91] in their paper discussing how interactions between technology and computer architecture affect the performance of computer systems.

Third, recent technological developments in networks and secondary storage systems make it possible to build high-bandwidth, supercomputer file servers at workstation prices. Until recently, anyone wishing to build a high-bandwidth, supercomputer I/O system had to invest millions of dollars in proprietary, high-bandwidth network technology and expensive parallel-transfer disks. But with the standardization of high-performance interconnects and network, such as HIPPI and FDDI, and the commercialization of the disk arrays, high-bandwidth networks and secondary storage systems have suddenly become affordable. What is lacking, and the point that RAID-II addresses, is a storage architecture that can exploit these developments.

1.2 Related Work

There are currently many existing file server architectures. We examine a few of them to serve as a background for the discussion of various aspects of RAID-II. First we examine RAID-I, a workstation-based file server with off-the-shelf disk controllers and disks. Next we look at the Auspex NS5000 file server, which provides scalable high-performance NFS file service. Finally, we examine several mass storage systems (MSS) currently used by supercomputing centers for high-capacity, shared storage.

1.2.1 RAID-I

We constructed RAID-I to see how well a workstation-based file server could provide access to the high data and I/O rates supported by disk arrays. The prototype was constructed using a Sun 4/280 workstation with 128 MB of memory, 28 5-1/4 inch SCSI disks and four dual-string SCSI controllers.

Experiments with RAID-I show that it is good at sustaining small random I/Os, performing approximately 300 4 KB random I/Os per second [Chervenak91]. However, RAID-I has proven woefully inadequate for high-bandwidth I/O, sustaining at best 2.3 MB/s to a user-level application on RAID-I. In comparison, a single disk on RAID-I can sustain 1.3 MB/s. 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 in moving data between the kernel DMA buffers and buffers in user space saturate the memory system when I/O bandwidth reaches 2.3 MB/s. 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, high-bandwidth performance is limited by the low bandwidth of the Sun 4/280's VME system bus. Although nominally rated at 40 MB/s, the bus becomes saturated at 9 MB/s.

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 systems, 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 bandwidth to memory drops quickly. Our experience with RAID-I indicates that the memory systems of workstations are, in general, poorly suited for supporting high-bandwidth I/O.

1.2.2 Auspex NS5000

The Auspex NS5000 [Nelson90] is designed for high-performance, NFS file service [Sandberg85]. NFS is the most common network file system protocol for workstation-based computing environments. It is primarily designed to support operations on small and medium sized files. Because NFS transfers files in small individual packets, it is inefficient for large files.

In NFS, as with most network file systems, each packet requires a relatively constant CPU overhead to process device interrupts and manage the network protocol. Because of this per-packet overhead, current workstations lack sufficient processing power to handle the large numbers of small NFS packets that are generated at high data transfer rates. Additionally, NFS is designed to support many clients making small file requests independent of each other. Although an individual client's request stream may exhibit locality, the sequence of requests seen by the server will include interleaved requests from many clients and thus show less locality. Finally, NFS was designed for workstation clients that typically handle small files. As a result, the protocols are optimized for files that can be transferred in relatively few packets.

The NS5000 handles the network processing, file system management, and disk control using separate dedicated processors. This *functional multiprocessing*, in contrast to symmetric multiprocessing, makes synchronization between processes explicit and allows the performance of the file server to scale by adding processors, network attachments and disks. Typical NFS file servers, on the other hand, perform all functions on a single processor. In such systems, performance can be scaled to only a very limited degree by adding network attachments and disks because the processor will quickly become a bottleneck. Due to this functional multiprocessing, the NS5000 can support up to 770 NFS I/Os per second with 8 Ethernets and 16 disks [Horton90].

Although the NS5000 is good at supporting small low-latency NFS requests, it is unsuitable for high-bandwidth applications. The use of a single 55 MB/s VME bus to connect the networks, disks and memory limits the aggregate I/O bandwidth of the system. NFS is also very inefficient for large files because it always breaks up files into small packets which are sent individually over the network. This results in fragmentation of the available network bandwidth and forces the receiving system to handle a large number of interrupts.

1.2.3 Supercomputer Mass Storage Systems

Almost all supercomputer mass storage systems use a mainframe as a high-performance file server. The mainframe runs the file system and provides a high-bandwidth data path between its channel-based I/O system and supercomputer clients via a high-performance channel or network interface. There are several problems with today's supercomputer mass storage system. First, most supercomputer mass storage systems are designed primarily for capacity, so very few support data transfer rates over 10 MB/s. For performance, supercomputer applications rely on locally attached parallel-transfer disks. Second, supercomputer mass storage systems are not designed to service a large number of small file requests and are rarely used as primary storage systems for large numbers of client workstations. Third, mainframes are very expensive, costing millions of dollars. The following briefly describes the MSS-II, NCAR, LSS, and Los Alamos National Labs mass storage systems

MSS-II [Tweten90], the NASA Ames mass storage system, uses an Amdahl 5880 as a file server. MSS-II achieves data transfer rates up to 10 MB/s by striping data over multiple disks and transferring data over multiple network channels. The practice of striping data across disks to provide higher data transfer rates has been used for some time in supercomputer environments.

The mass storage system at NCAR [Nelson87], the National Center for Atmospheric Research, is implemented using Hyperchannel and an IBM mainframe running MVS. The NCAR mass storage system is unique in that it provides a direct data path between supercomputers and the IBM mainframe's channel-based storage controllers. On a file access, data can bypass the mainframe and be transferred directly between the storage devices and the supercomputers.

The Lawrence Livermore National Laboratory's LINCS Storage System (LSS) [Foglesong90], is one of the systems upon which the Mass Storage System (MSS) Reference Model [Coleman90] is based. A notable aspect of LSS is that control and data messages are always transmitted independently. This allows the control and data messages to take different paths through the system. For example, a control message requesting a write might be sent to the bitfile server via an Ethernet but the data itself would be sent directly to the storage server via a high speed HIPPI channel, bypassing the bitfile server.

Los Alamos National Labs's High-Performance Data System [Collins91] is an experimental prototype designed to support high-bandwidth I/O for the LANL supercomputers. The LANL design is quite close to the RAID-II architecture. It directly connects an IBM RAID Level 3 disk array to a HIPPI network and controls the data movement remotely (over an Ethernet) from a IBM RISC/6000. Los Alamos has demonstrated data rates close to the maximum data rate of the IBM disk array, which is close to 60 MB/s. The main difference between LANL's High-Performance Data System and RAID-II is that LANL 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.

In summary, although most supercomputer mass storage systems can transfer data at rates up to 10 MB/s, this is still insufficient to support diskless supercomputers. Furthermore, they neglect the performance of small NFS-type requests in order to optimize the performance of high-bandwidth data transfers. Finally, even if the supercomputer mass storage systems were optimized to support NFS-type requests, it would be economically infeasible to use mainframes as file servers for workstations. The mass storage system closest to RAID-II is the Los Alamos High-Performance Data System, which uses a HIPPI-attached IBM disk array, but this disk array only supports one I/O at a time.

2 RAID-II Architecture

RAID-II is a high-performance file server that interfaces a SCSI-based disk array to a HIPPI network. Our goal in designing RAID-II is to provide high-bandwidth access from the network to a large disk array without transferring data through the relatively slow file server (a Sun4/280 workstation) backplane. To do this, we designed a custom printed-circuit board called the *XBUS card*. The main purpose of the XBUS card is to provide a high-bandwidth path between the major system components: the HIPPI network, four VME busses that connect to VME disk controllers, and an interleaved, multiported semiconductor memory. The XBUS card also contains a parity computation engine that generates parity for writes and reconstruction on the disk array. The entire system is controlled by an external Sun 4/280 file server through a memory-mapped control register interface. Figure 1 shows a block diagram for the controller. To minimize the design effort, we used commercially available components whenever possible. Thinking Machines (TMC) provided a board set for the HIPPI network interface; 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.

2.1 XBUS Card Architecture

The XBUS card implements a 4x8 (four memory ports and eight client ports), 32-bit wide crossbar, which we call the XBUS. All XBUS transfers involve one of the four memory ports as either the source or the destination of the transfer. Each memory port is designed to

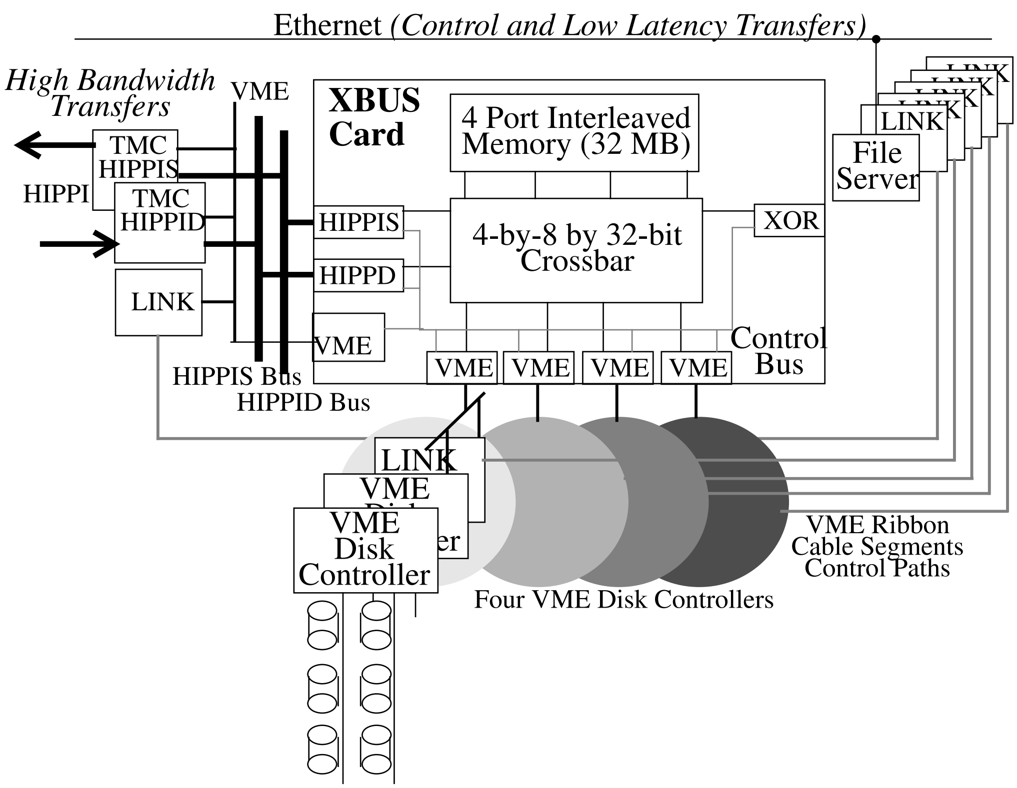


Figure 1: RAID-II Organization. A high-bandwidth crossbar interconnect ties the network interface (HIPPI), the disk controllers, a multiported memory system, and a parity computation engine. An internal control bus provides access to the crossbar ports, while external point-to-point VME links provide control paths to the surrounding SCSI and HIPPI interface boards. Up to two VME disk controllers can be attached to each of the four VME interfaces. The design originally had 8 memory ports and 128 MB of memory; however, we built a four memory port version to reduce manufacturing time.

transfer bursts of data at 50 MB/s and sustain transfers at 40 MB/s, for a total sustainable memory bandwidth on the XBUS card of 160 MB/s.

The XBUS is a synchronous multiplexed (address/data) crossbar-based interconnect that uses a centralized strict priority-based arbitration scheme. All paths to memory can be reconfigured on a cycle-by-cycle basis. Each of the eight 32-bit XBUS ports operates at a cycle time of 80 ns.

The XBUS supports reads and write transactions. Between 1 and 16 words are transferred over the XBUS during each transaction. Each transaction consists of an arbitration phase, an address phase, and a data phase. If there is no contention for memory, the arbitration and address phases each take a single cycle; data is then transferred at the rate of one word per cycle. The memory may arbitrarily insert wait cycles during the address and data cycles to compensate for DRAM access latencies and refreshes. The shortest XBUS transaction is a single word write, taking three cycles (one each for the arbitration, address, and data phases).

Data is interleaved across the four banks in 16-word interleave units. Although the crossbar is designed to move large blocks between memory, network, and disk interfaces, it is still possible to access a single word when necessary. The external file server can access the on-board memory through the XBUS card's VME control interface.

Of the eight client XBUS ports, two interface to the TMC I/O bus (HIPPIS/HIPPID busses). The HIPPI board set also interfaces to this bus. These XBUS ports are unidirectional and can sustain transfers of up to 40 MB/s, with bursts of up to 100 MB/s into 32 KB FIFO interfaces.

Four of the client ports are used to connect the XBUS card to four VME busses, each of which can connect to one or two dual-string disk controllers. Because of the physical packaging of the array, 6 to 12 disks can be attached to each disk controller in two rows of three to six disks each. Thus, up to 96 disk drives can be connected to each XBUS card. Each VME interface has a maximum transfer bandwidth of 40 MB/s; however, in our experience, the Sun 4 VME can usually realize at most 8-10 MB/s [Chervenak91]. The VME disk controllers that we use, Interphase Cougar disk controllers, can transfer data at 8 MB/s[1], for a total maximum bandwidth to the disk array of 32 MB/s.

Of the remaining two client ports, one interfaces to a parity computation engine. The last port links the XBUS card to the external file server. It provides access to the on-board memory as well as the board's control registers (through the board's control bus). This makes it possible for file server software, running on the controller, to access network headers and file meta-data in the controller cache.

2.2 Design Issues

The first design issue is whether memory/XBUS contention significantly degrades system performance in the XBUS design.

Before deciding upon the XBUS, we considered using a single, wide bus interconnect. Its main attraction was conceptual simplicity. However, a 128-bit wide bus would have required a huge number of FIFO and bus transceiver chips. While we could have used a small number of time-multiplexed 128-bit ports, interfaced to narrower 32-bit and 64-bit busses, the result would have been a more complex system.

A disadvantage of using a crossbar interconnect with multiple, independent memories is the possibility of contention for memory. For example, if there were four client ports (each transferring at 40 MB/s) accessing a random memory, then on average 1.26 memory ports would be idle and 50 MB/s of the available memory bandwidth would be wasted. The original XBUS design had eight memory ports to minimize this contention; however, we later reduced the number of memory ports to four due to problems routing the original design [Katz93].[2] The total sustainable demand of all client-ports is 150 MB/s (40 MB/s for each non-VME port and 6 MB/s for each VME), so the XBUS could be close to fully utilized and memory contention could seriously degrade system performance.

While contention for memory modules is a concern, actual contention should be infrequent. Most XBUS ports perform large accesses at least of at least a few KB so that when two accesses conflict, the loser of the arbitration deterministically follows the winner around the memory modules, avoiding further conflicts. Each XBUS port buffers 4-32 KB of data to/from the XBUS to even out fluctuations.

The second design issue is the necessity of two networks, one for high bandwidth, one for low latency.

The remote connection of the HIPPI network to the XBUS card increases the latency of sending a HIPPI packet. Thus, while the XBUS's HIPPI network provides high bandwidth, its latency is actually worse than if we had directly connected it to the Sun4 file server. Also, because it is a high-bandwidth network, even small latencies waste a large fraction of its available bandwidth and it is thus inefficient at transferring small packets. Thus, RAID-II also supports an Ethernet that is directly connected to the Sun4 file server for small transfers where network latency dominates service time. In this paper, we measure the latency of the high bandwidth HIPPI to determine if having the low latency Ethernet is necessary.

The third design issue is the performance benefit of including a special purpose parity computation engine.

Disk arrays inherently require extra computation for error correction codes. Several popular disk array architectures, such as RAID Levels 3 and 5, use parity as the error correction code [Patterson88, Lee91]. Parity is computed when writing data to the disk array and when reconstructing the contents of a failed disk. Parity computation is a simple but bandwidth intensive operation that workstations, with their limited memory bandwidth, perform too slowly. To address this concern, we provide a simple parity computation engine on the XBUS card to speed up disk writes and reconstructions.

1. This is the transfer rate for SCSI-1 drives; with SCSI-2, the Cougar disk controllers can transfer at higher rates.

2. We have since routed the full 8x8 version of the XBUS card.

We measure the performance of this parity engine and compare it to the performance of computing parity on the workstation file server.

The fourth design issue this paper addresses is the success or failure of connecting the network and disks to the XBUS instead of to the file server.

A key feature of RAID-II's architecture is the minimal use of the file server's backplane. Data never goes over the file server's backplane; instead, data travels directly from the HIPPI network to the XBUS memory to the disks. The file server still controls this data movement but never touches the data.[3] By doing this, we hope to achieve end-to-end performance that is limited by XBUS's performance rather than the file server backplane's performance.

3 Performance and Design Evaluation

This section is organized in two parts. The first part reports a series of system-level performance measurements by sending data from the disk subsystem to/from the network and analyzes the performance of each system component to locate system bottlenecks. The second part examines the design issues raised in Section 2.

3.1 System Performance

For all system-level experiments, the disk system is configured as a RAID Level 5 [Patterson88] with one parity group of 24 disks. We break down system tests into two separate experiments, reads and writes; Figure 2 shows the results. For reads, data is read from the disk array into the memory on the XBUS card and from there is sent over HIPPI back to the XBUS card into XBUS memory. For writes, data originates in XBUS memory, is sent over the HIPPI back to the XBUS card to XBUS memory, parity is computed, then both data and parity are written to the disk array. We use the XBUS card for both sending and receiving the data because of our current lack of another system that can source or sink data at the necessary bandwidth.[4] For both tests, the system is configured with four Interphase Cougar disk controllers with six disks on each disk controller. Figure 2 shows that, for large requests, system-level read and write performance tops out at about 20 MB/s. 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 of 40 MB/s. In the next sections, we measure the performance of each component to determine what limits performance.

3.2 HIPPI

Figure 3 shows the performance of the HIPPI network and boards. Data is 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 network's raw hardware performance. 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 requests result in low performance. For large requests, however, the XBUS and HIPPI boards support 38 MB/s in both directions, which is very close to the maximum bandwidth of each XBUS port. During these large transfers, the XBUS card is transferring a total of 76 MB/s, 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 liming factor in determining system level performance.

3.3 Parity Engine

Figure 4 shows the performance of the parity computation engine, which is used in performing writes to the disk array. After the file server writes control words into the parity engine's DMA, the parity engine reads words from XBUS memory, computes the exclusive-or, and then writes the resulting parity blocks back into XBUS memory. Figure 4 graphs performance for all combinations of buffer sizes that are powers of 2 between 1 KB and 1 MB and {1, 2, 4, 8, and 16} data buffers. The amount of data transferred by the parity engine is the size of each buffer times the number of buffers plus one (for the result buffer). Performance depends mainly on the total amount of data transferred

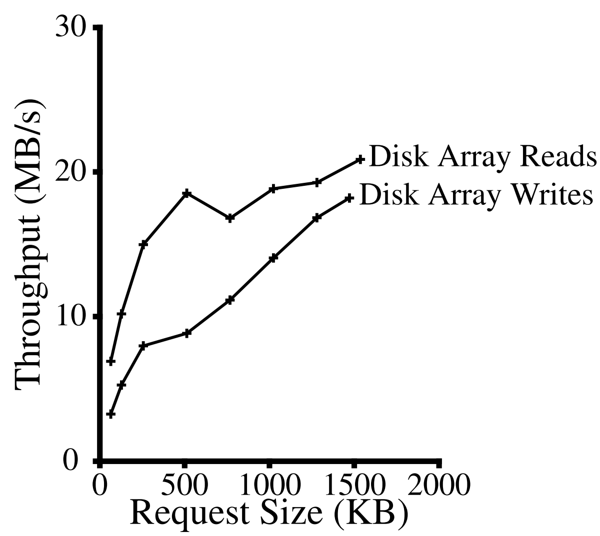


Figure 2: System Level Read and Write Performance. This figure shows the overall performance of the RAID-II storage server for disk reads and writes. For disk reads, data is read from the disk array into XBUS memory, transferred over the HIPPI network and back to XBUS memory. For disk writes, data starts in XBUS memory, is transferred over the HIPPI network back to XBUS memory, parity is computed, then the data and parity are written to the disk array. CPU utilization limits performance for small requests; disk system throughput limits performance for large requests. Request size refers to the total size of the request before it is broken into individual disk requests.

3. The server does store and manipulate the meta-data, such as inodes and directory structures.

4. We are in the process of connecting to a 100 MB/s video microscope at Lawrence Berkeley Laboratories.

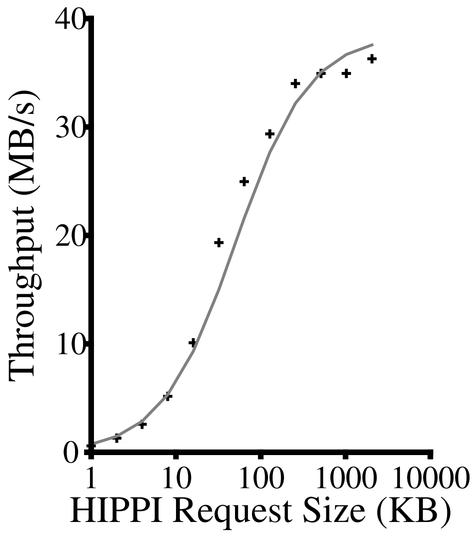


Figure 3: HIPPI Loopback This figure shows the performance of the HIPPI network. Data is transferred from the XBUS memory to the HIPPI source board to the HIPPI destination board and back to XBUS memory. The overhead of sending a HIPPI packet is about 1.1 ms, mostly due to setting up the HIPPI and XBUS control registers. The measured data is shown as symbols; the dashed line represents performance derived from a simple model with a constant overhead of 1.1 ms and a maximum throughput of 38.5 MB/s.

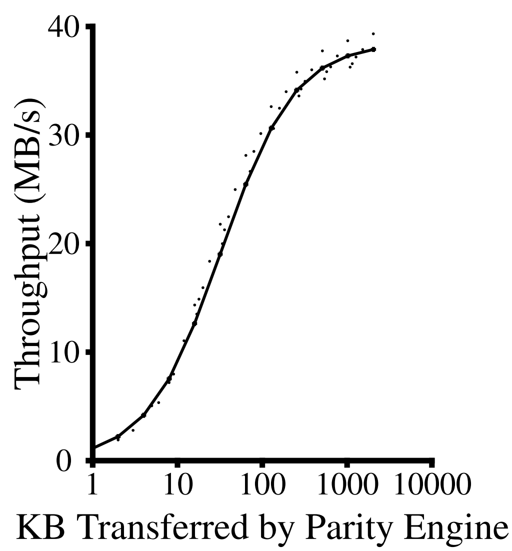


Figure 4: Parity Engine Performance. This figure shows the performance of the parity computation engine. The symbols represent measured data, gathered from workloads with buffer sizes that are powers of 2 between 1 KB and 1 MB, and {1, 2, 4, 8, and 16} data buffers. The line represents performance derived from a simple model with a constant overhead of 0.9 ms and a maximum throughput of 38.5 MB/s. Throughput measures the total number of bytes transferred (including both data and parity) divided by elapsed time.

rather than the number of data buffers. The line in Figure 4 represents performance derived from a simple model with a constant overhead of 0.9 ms and a maximum throughput of 38.5 MB/s. As with the HIPPI, the parity engine is not a bottleneck, though the fixed overhead of 0.9 ms degrades performance when less than 100 KB is transferred.

3.4 Disk Subsystem

Figure 5 and Figure 6 show how well RAID-II performance scales with the number of disks. All graphs use 64 KB requests. The dashed lines indicate the performance under perfect, linear scaling. Figure 5a and Figure 6a show how the performance of one string on one Interphase Cougar disk controller scales as the number of disks varies from 1 to 5. Each string can support the full bandwidth of two disks; with more than two disks, the maximum string throughput of 3.1 MB/s limits its performance. For all other graphs, we use three disks per string.

Figure 5b and Figure 6b show how well each Cougar disk controller supports multiple strings. In these figures, we graph how performance increases as we go from one string per controller to two. We see that using two strings per controller almost doubles performance over using one string per controller, topping out at 5.3 MB/s for disk reads and 5.6 MB/s for disk writes. For the rest of the graphs, we use two strings per controller.

Figure 5c and Figure 6c vary the number of disk controllers per VME bus from one to two. Performance per VME bus is limited to 6.9 MB/s for disk reads and 5.9 MB/s for disk writes due to a relatively slow synchronous design of the XBUS VME interface [Katz93][5]. For the remainder of the paper, we use one disk controller per VME bus.

Figure 5d and Figure 6d increase the number of VME busses active on the XBUS from one to five[6]. Performance of the XBUS card scales linearly with the number of controllers for both reads and writes.

It is clear from these graphs that the limiting factor to overall system performance is the disk subsystem. In designing RAID-II, we expected each string to support 4 MB/s. Thus, with four dual-string controllers, the total RAID-II disk bandwidth should be 32 MB/s. However, with three disks per string, each string transfers only 3.1 MB/s for disk writes and 2.6 MB/s for disk reads, so the maximum total string bandwidth is 25 MB/s for disk writes and 21 MB/s for disk reads. Other than string performance, the system scales linearly with one disk controller on each VME bus.

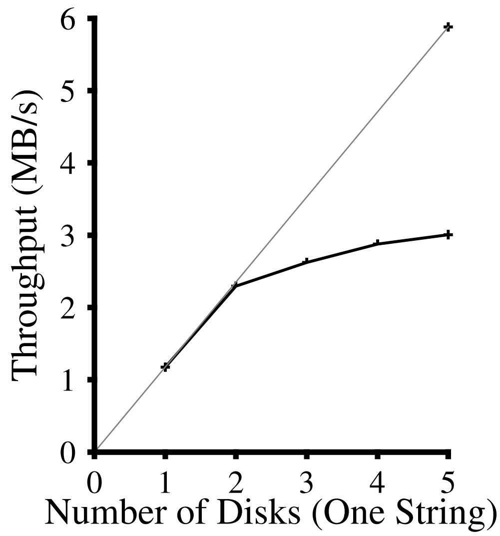
3.5 Evaluation of Design

In this section, we examine the four design questions raised in Section 2:

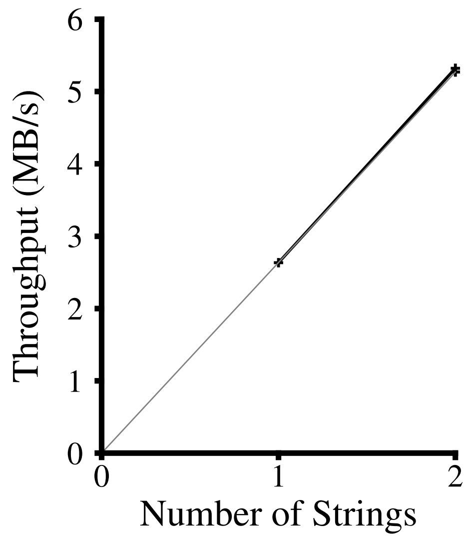
5. To implement the VME interface as quickly as possible, we designed a simple, synchronous VME interface. This interface takes approximately 5 XBUS cycles per VME word. By pipelining data words and using a faster VME clock rate, this interface could be sped up significantly.
6. The disk controller on the fifth VME bus used only one string due to lack of sufficient cabling.

- Does memory/XBUS contention significantly degrade system performance?
- Is it necessary to have two networks: one (HIPPI) attached to the XBUS card for high bandwidth, one (Ethernet or FDDI) attached to the file server for low latency?
- How much does the parity computation engine improve performance?
- Does attaching the network to the XBUS card instead of the file server increase performance?

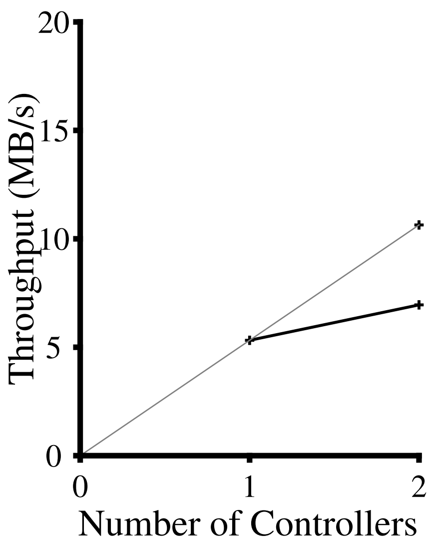
First, we examine memory/XBUS contention. Table 1 shows that when the HIPPID, HIPPIS and parity engine ports are simultaneously active, the XBUS expe-



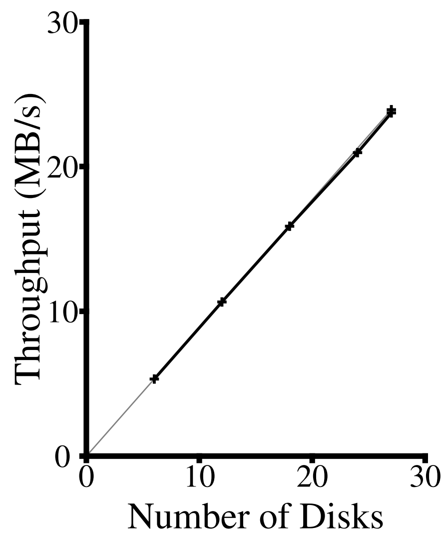
(a) Varying Disks Per String



(b) Varying Strings Per Controller



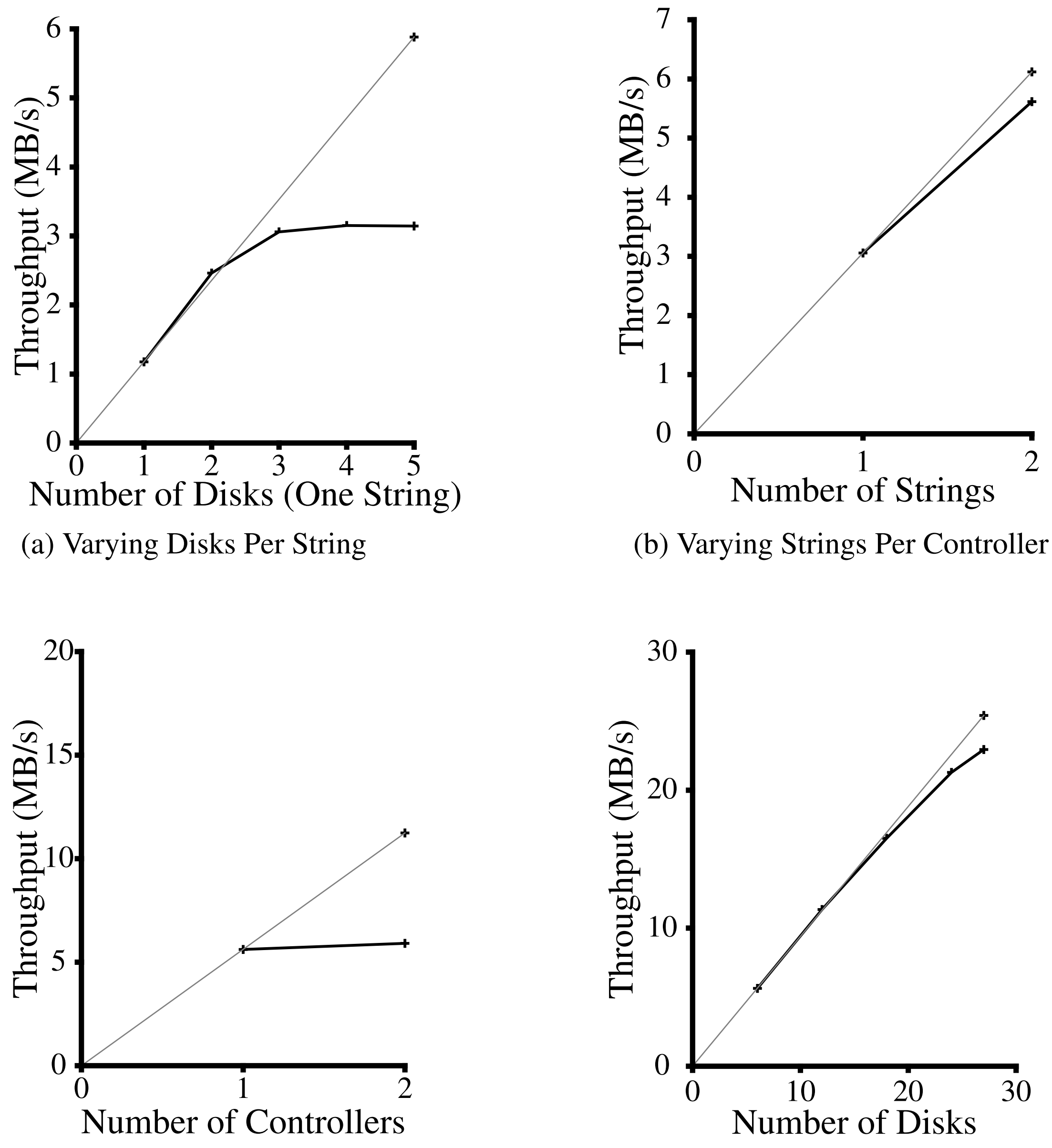
(c) Varying Controllers Per VME Bus



(d) Varying Number of Disks in System

Figure 5: Disk Read Performance. This figure shows how disk read performance scales with increasing numbers of disks. Figure (a) uses one string on one Interphase Cougar and varies the number of disks from 1 to 5. Figure (b) fixes the number of disks per string at three and varies the number of strings per Cougar from 1 to 2. Figure (c) varies the number of Cougars per VME bus from 1 to 2, fixing the number of disks per string at three and the number of strings per Cougar at two. Figure (d) varies the number of VME busses attached to the XBUS card from one to four, fixing the number of disks per string at three, the number of strings per Cougar at two, and the number of Cougars per VME bus at one. Dotted lines show linear speedup.

riences no contention for memory and can support the full bandwidth of each (38 MB/s). This lack of contention is due to two reasons. First, memory is interleaved on a fine-grained basis (16 words), so the memory ports are evenly load balanced; that is, no memory port is more heavily loaded than any other memory port. Second, if two accesses collide by trying to access the same memory port, one access must wait for the other. But, after this initial waiting period, the loser follows the winner around the memory banks deterministically. Because the average access is many times larger than



(c) Varying Controllers Per VME Bus

(d) Varying Number of Disks in System

Figure 6: Disk Write Performance. This figure shows how disk write performance scales with increasing numbers of disks. Figure (a) uses one string on one Interphase Cougar and varies the number of disks from 1 to 5. Figure (b) fixes the number of disks per string at three and varies the number of strings per Cougar from 1 to 2. Figure (c) varies the number of Cougars per VME bus from 1 to 2, fixing the number of disks per string at three and the number of strings per Cougar at two. Figure (d) varies the number of VME busses attached to the XBUS card from one to four, fixing the number of disks per string at three, the number of strings per Cougar at two, and the number of Cougars per VME bus at one. These tests measure raw disk write performance—the disk array is configured without redundancy. Dotted lines show linear speedup.

the unit of interleaving, the initial waiting period is insignificant relative to the total transfer time.

Performance suffers, however, when the disk subsystem begins to read data from the disks to the memory. There are two factors leading to this degradation. First, each VME port can sustain less than 8 MB/s, which is less than one-fourth the bandwidth of a single memory port. Memory requests from other XBUS ports must hence wait behind the slow VME access Second, we have configured the disk controllers to transfer only 32 32-bit words per VME access. These two factors cause the access pattern generated by a disk port is thus similar to small, random memory accesses, resulting in a much higher degree of contention for memory. Since the VME bus supports transfer sizes of up to 128 32-bit words, XBUS and memory contention could be decreased by increasing the size of VME accesses. This may decrease overall performance, however, by increasing the time control words wait for the VME bus. We are currently tuning performance by varying the size of VME accesses.

Table 1 shows XBUS contention when all four ports (HIPPID, HIPPIS, parity engine, and disk subsystem) are active. Real application environments will not have the network configured to loop back, so only one of the network ports would be active for disk reads or disk writes. In such an environment, the XBUS could easily support the full port bandwidth (38 MB/s) for both disk reads (exercising the HIPPIS and disk subsystem) or disk writes (exercising the HIPPID, parity engine, and disk subsystem).

Next, we examine the benefit of using two networks, one for high bandwidth and one for low latency. We saw in Figure 3 that the overhead of initiating a HIPPI access was a surprisingly low 1.1 ms, comparable tothe Ethernet latency of about 0.5 ms. In this short time, however, the HIPPI could have transferred 40 KB.

	Disk Array Read Performance	**Disk Array Write Performance**	**Write Performance Degradation**
RAID-I	2.4 MB/s	1.2 MB/s	50%
RAID-II	20.9 MB/s	18.2 MB/s	13%
RAID-II speedup	8.7	15.2	

Table 1: Performance Comparison between RAID-II and RAID-I. This table compares the performance of RAID-II to that of RAID-I. Because RAID-II has a special purpose parity engine, disk array write performance is comparable to disk array read performance. All writes in this test are full-stripe writes [Lee91]. For RAID-II reads, data is read from the disk array into XBUS memory then sent over the HIPPI network back to XBUS memory. For RAID-I reads, data is read from the disk array into Sun4 memory, then copied again into Sun4 memory. This extra copy equalized the number of memory accesses per data word. For RAID-II writes, data starts in XBUS memory, is sent over HIPPI back into XBUS memory, parity is computed, and the data and parity are written to the disk subsystem. For RAID-I writes, data starts in Sun4 memory, gets copied to another location in Sun4 memory, then is written to disk. Meanwhile, parity is computed on the Sun4. RAID-I uses a 32 KB striping unit with 8 disks; RAID-II uses a 64 KB striping unit with 24 disks.

Test	**Aggregate XBUS Bandwidth**	**Average Bandwidth Per Active Subsystem**
HIPPID + HIPPIS + Parity Engine	114 MB/s	38 MB/s
HIPPID + HIPPIS + Parity Engine + Disk Subsystem	110 MB/s	28 MB/s

Table 2: XBUS contention. When using only the HIPPID, HIPPIS, and parity engine, the XBUS experiences no contention for the memory banks, supporting the full bandwidth of each port. However, performance suffers when the disk subsystem is simultaneously reading data from the disks to the memory. There are two factors leading to this degradation. First, each VME port can sustain less than 8 MB/s, which is less than one-fourth the bandwidth of a single memory port. Memory requests from other XBUS ports must hence wait behind the slow VME access Second, we have configured the disk controllers to transfer only 32 32-bit words per VME access. These two factors cause the access pattern generated by a disk port is thus similar to small, random memory accesses, resulting in a much higher degree of contention for memory. Even with this degradation, the XBUS can still support 37 MB/s of disk write (using only the HIPPID, parity engine, and disk subsystem) or disk read activity (using the HIPPIS and disk subsystem).

Because of the 1.1 ms overhead and the high transfer rate of HIPPI, network packet sizes of 120 KB are necessary to achieve 75% utilization of the HIPPI. In such a situation, small packets (less than 40 KB) are best transferred over a lower latency network for two reasons. First, small packets waste a disproportionately large amount of the available HIPPI bandwidth. Second, since large average HIPPI packets are needed to efficiently utilize the HIPPI, small packets which encounter a large packet currently being transferred must wait many milliseconds before they can use the network. We thus conclude that having two networks benefits performance both for small and large packets, small packets because they need not wait while large packets use the network, large packets because the available bandwidth of the HIPPI is not lost due to inefficient small packets. In addition, adding an Ethernet connection costs essentially nothing, since all workstations already come with an Ethernet connection.

Third, we examine how much the parity engine enhances disk array write performance. To do so, we compare read and write performance of RAID-II, which has a parity engine, to the read and write performance of our first prototype, RAID-I, which has no parity engine and must use its Sun4 file server to compute parity [Chervenak91]. Both RAID-II and RAID-I tests use the same number of memory transfers.[7] Table 2 shows RAID-II full-stripe writes [Lee91] are only 13% slower than reads, where RAID-I full-stripe writes are 50% slower than reads. Without the parity engine, computing parity on RAID-II would have been even slower than on RAID-I, because the data over which parity is computed resides on the XBUS card and would have to first be transferred over a VME link into the Sun4's memory before parity could be computed. Even once data resides in Sun4 memory, parity can only be computed at a few megabytes per second. Clearly hardware parity is necessary for high-performance in RAID Level 5 systems.

Last, we examine our decision to attach the HIPPI network to the XBUS card instead of to the Sun4 file server. To do so, we again compare RAID-II performance to RAID-I performance. Table 2 shows that RAID-II achieves 8.7 to 15.2 times the performance of RAID-I. If we had simply connected the HIPPI to the Sun4's VME bus instead of to the XBUS card, RAID-II's performance would have been comparable to RAID-I's performance, because RAID-I's performance is limited by the speed of its memory bus, so using more disks and a fast network would not have increased performance [Chervenak91].

4 Conclusions

In this paper, we have evaluated the performance of RAID-II, a network-attached storage server. We have demonstrated end-to-end system performance of 20 MB/s for both disk array reads and writes. We have also examined four major design issues in the design of RAID-II. We first measured contention for the main interconnect of RAID-II, the XBUS, and found marginal levels of contention, due to the fine-grained interleaving of the memory ports and the relatively large accesses made by the HIPPI and parity ports. We next examined the necessity of using two networks, one for high bandwidth and one for low latency. We concluded that, due to the high bandwidth of the HIPPI network, even the moderate 1.1 ms network overhead per request would waste a large fraction of network bandwidth for small requests, thus justifying our use of a second, low-latency network. We then examined the performance benefit of the parity computation engine for disk array writes and concluded that it allowed full-stripe writes to achieve almost the same performance as full-stripe reads. Last, we examined the wisdom of attaching the HIPPI directly to the XBUS card by comparing performance to RAID-I. We concluded that this allowed system performance to scale with the XBUS rather than being limited by the speed of the Sun4 memory bus.

We have been quite pleased by the functionality and performance of RAID-II. Using identical Sun4 file servers and essentially the same disks and controllers as RAID-I, we achieved a speedup of 8 to 15 over RAID-I. The use of the XBUS card, with its custom XBUS interconnect and parity engine, enabled performance to scale well beyond the limits of the Sun4 memory bus. The XBUS card fully supported the bandwidth requirements of 24 disks and achieved end-to-end performance of 20 MB/s.

5 Acknowledgments

This work was supported in part by NASA/-DARPA grant number NAG 2-591, NSF MIP 8715235, and NFS Infrastructure Grant No. CDA-8722788. RAID-II was made practical through donations by IBM, Thinking Machines Corporation, IDT, and Sun Microsystems. We gratefully acknowledge contributions in the design of RAID-II by Garth Gibson, Rob Pfile, Rob Quiros, and Mani Varadarajan. Lance Lee at Interphase and Steve Goodison at Andataco were instrumental in providing quick delivery of five Interphase Cougar VME disk controllers for evaluation.

6 References

[Chervenak91] Ann L. Chervenak and Randy H. Katz. Performance of a Disk Array Prototype. In *Proceedings of the 1991 ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems*, volume 19, pages 188–197, May 1991. Performance Evaluation Review.

[Coleman90] Sam Coleman and Steve Miller. Mass Storage System Reference Model: Version 4, May 1990.

[Collins91] Bill Collins, Lynn Jones, Granville Chorn, Ronald Christman, Danny Cook, and Christina Mercier. Los Alamos High-Performance Data System: Early Experiences. Technical Report La-UR-91-3590, Los Alamos, November 1991.

7. We compensate for the lack of a fast network (HIPPI) on RAID-I by replacing the network send with a memory to memory copy (kernel to user space).

[Foglesong90] Joy Foglesong, George Richmond, Loellyn Cassell, Carole Hogan, John Kordas, and Michael Nemanic. The Livermore Distributed Storage System: Implementation and Experiences. *Mass Storage Symposium*, May 1990.

[Hennessy91] John L. Hennessy and Norman P. Jouppi. Computer Technology and Architecture: An Evolving Interaction. *IEEE Computer*, pages 18–29, September 1991.

[Horton90] 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.

[Katz93] Randy H. Katz, Peter M. Chen, Ann L. Drapeau, Edward K. Lee, Ken Lutz, Ethan L. Miller, Srinivasan Seshan, and David A. Patterson. RAID-II: Design and Implementation of a Large Scale Disk Array Controller. *1993 Symposium on Integrated Systems*, 1993. University of California at Berkeley UCB/CSD 92/705.

[Lee91] Edward K. Lee and Randy H. Katz. Performance Consequences of Parity Placement in Disk Arrays. In *Proceedings of the 4th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS-IV)*, pages 190–199, April 1991.

[Lee92] Edward K. Lee, Peter M. Chen, John H. Hartman, Ann L. Chervenak Drapeau, Ethan L. Miller, Randy H. Katz, Garth A. Gibson, and David A. Patterson. RAID-II: A Scalable Storage Architecture for High-Bandwidth Network File Service. Technical Report UCB/CSD 92/672, University of California at Berkeley, February 1992.

[Nelson87] Marc Nelson, David L. Kitts, John H. Merrill, and Gene Harano. The NCAR Mass Storage System. *IEEE Symposium on Mass Storage*, pages 12–20, November 1987.

[Nelson90] Bruce Nelson. An Overview of Functional Multiprocessing for NFS Network Servers. Technical Report Technical Report 1, Auspex Engineering, July 1990.

[Patterson88] David A. Patterson, Garth Gibson, and Randy H. Katz. A Case for Redundant Arrays of Inexpensive Disks (RAID). In *International Conference on Management of Data (SIGMOD)*, pages 109–116, June 1988.

[Sandberg85] Russel Sandberg, David Goldberg, Steve Kleiman, Dan Walsh, and Bob Lyon. Design and Implementation of the Sun Network Filesystem. In *Summer 1985 Usenix Conference*, 1985.

[Tweten90] David Tweten. Hiding Mass Storage Under Unix: NASA's MSS-II Architecture. *Tenth IEEE Symposium on Mass Storage*, pages 140–145, May 1990.