High Performance Network and Channel-Based Storage

Randy H. Katz

Computer Science Division
Department of Electrical Engineering and Computer Sciences
University of California
Berkeley, California 94720

Abstract: In the traditional mainframe-centered view of a computer system, storage devices are coupled to the system through complex hardware subsystems called I/O channels. With the dramatic shift towards workstation-based computing, and its associated client/server model of computation, storage facilities are now found attached to file servers and distributed throughout the network. In this paper, we discuss the underlying technology trends that are leading to high performance network-based storage, namely advances in *networks*, *storage devices*, and *I/O controller* and *server architectures*. We review several commercial systems and research prototypes that are leading to a new approach to high performance computing based on network-attached storage.

Key Words and Phrases: High Performance Computing, Computer Networks, File and Storage Servers, Secondary and Tertiary Storage Device

1. Introduction

The traditional mainframe-centered model of computing can be characterized by small numbers of large-scale mainframe computers, with shared storage devices attached via I/O channel hardware. Today, we are experiencing a major paradigm shift away from centralized mainframes to a distributed model of computation based on workstations and file servers connected via high performance networks.

What makes this new paradigm possible is the rapid development and acceptance of the *client-server model* of computation. The client/server model is a message-based protocol in which *clients* make requests of service providers, which are called *servers*. Perhaps the most successful application of this concept is the widespread use of file servers in networks of computer workstations and personal computers. Even a high-end workstation has rather limited capabilities for data storage. A distinguished machine on the network, customized either by hardware, software, or both, provides a *file service*. It accepts network messages from client machines containing open/close/read/write file requests and processes these, transmitting the requested data back and forth across the network.

This is in contrast to the pure *distributed storage model*, in which the files are dispersed among the storage on workstations rather than centralized in a server. The advantages of a distributed organization are that resources are placed near where they are needed, leading to better performance, and that the environment can be more autonomous because individual machines continue to perform useful work even in the face of network failures. While this has been the more popular approach over the last few years, there has emerged a growing awareness of the advantages of the centralized view. That is, every user sees the same file system, independent of the machine they are currently using. The view of storage is pervasive and transparent. Further, it is much easier to administer a centralized system, to provide software updates and archival backups. The resulting organization combines distributed processing power with a centralized view of storage.

Admittedly, centralized storage also has its weaknesses. A server or network failure renders the client workstations unusable and the network represents the critical performance bottleneck. A highly tuned remote file system on a 10 megabit (Mbit) per second Ethernet can provide perhaps 500K bytes per second to remote client applications. Sixty 8K byte I/Os per second would fully utilize this bandwidth. Obtaining the right balance of workstations to servers depends on their relative processing power, the amount of memory dedicated to file caches on workstations and servers, the available network bandwidth, and the I/O bandwidth of the server. It is interesting to note that today's servers are not I/O limited: the Ethernet bandwidth can be fully utilized by the I/O bandwidth of only two magnetic disks!

Meanwhile, other technology developments in processors, networks, and storage systems are affecting the relationship between clients to servers. It is well known that processor performance, as measured in MIPS ratings, is increasing at an astonishing rate, doubling on the order of once every eighteen months to two years. The newest generation of RISC processors have performance in the 50 to 60 MIPS range. For example, a recent workstation announced by Hewlett-Packard Corporation, the HP 9000/730, has been rated at 72 SPECMarks (1 SPECMark is roughly the processing power of a single Digital Equipment Corporation VAX 11/780 on a particular benchmark set). Powerful shared memory multiprocessor systems, now available from companies such as Silicon Graphics and Solborne, provide well over 100 MIPS performance. One of Amdahl's famous laws equated one MIPS of processing power with one megabit of I/O per second. Obviously such processing rates far exceed anything that can be delivered by existing server, network, or storage architectures.

Unlike processor power, network technology evolves at a slower rate, but when it advances, it does so in order of magnitude steps. In the last decade we have advanced from 3 Mbit/second Ethernet to 10 Mbit/second Ethernet. We are now on the verge of a new generation of network technology, based on fiber optic interconnect, called FDDI. This technology promises 100 Mbits per second, and at least initially, it will move the server bottleneck from the network to the server CPU or its storage system. With more powerful processors available on the horizon, the performance challenge is very likely to be in the storage system, where a typical magnetic disk can service thirty 8K byte I/Os per second and can sustain a data rate in the range of 1 to 3 MBytes per second. And even faster networks and interconnects, in the gigabit range, are now commercially available and will become more widespread as their costs begin to drop [UltraNet 90].

To keep up with the advances in processors and networks, storage systems are also experiencing rapid improvements. Magnetic disks have been doubling in storage capacity once every three years. As disk form factors shrink from 14" to 3.5" and below, the disks can be made to spin faster, thus increasing the sequential transfer rate. Unfortunately, the random I/O rate is improving only very slowly, due to mechanically-limited positioning delays. Since I/O and data rates are primarily disk actuator limited, a new storage system approach called *disk arrays* addresses this problem by replacing a small number of large format disks by a very large number of small format disks. Disk arrays maintain the high capacity of the storage system, while enormously increasing the system's disk actuators and thus the aggregate I/O and data rate.

The confluence of developments in processors, networks, and storage offers the possibility of extending the client-server model so effectively used in workstation environments to higher performance environments, which integrate supercomputer, near supercomputers, workstations, and storage services on a very high performance network. The technology is rapidly reaching the point where it is possible to think in terms of *diskless supercomputers* in much the same way as we think about diskless workstations. Thus, the network is emerging as the future "backplane" of high performance systems. The challenge is to develop the new hardware and software architectures that will be suitable for this world of network-based storage.

The emphasis of this paper is on the integration of storage and network services, and the challenges of managing the complex storage hierarchy of the future: file caches, on-line disk storage, near-line data libraries, and off-line archives. We specifically ignore existing mainframe I/O architectures, as these are well described elsewhere (for example, in [Hennessy 90]. The rest of this paper is organized as follows. In the next three sections, we will review the recent advances in interconnect, storage devices, and distributed software, to better understand the underlying changes in network, storage, and software technologies. Section 5 contains detailed case studies of commercially available high performance networks, storage servers, and file servers, as well as a prototype high performance network-attached I/O controller being developed at the University of California, Berkeley. Our summary, conclusions, and suggestions for future research are found in Section 6.

2. Interconnect Trends

2.1. Networks, Channels, and Backplanes

Interconnect is a generic term for the "glue" that interfaces the components of a computer system. Interconnect consist of high speed hardware interfaces and the associated logical protocols. The former consists of physical wires or control registers. The latter may be interpreted by either hardware or software. From the viewpoint of the storage system, interconnect can be classified as high speed networks, processor-to-storage channels, or system backplanes that provide ports to a memory system through direct memory access techniques.

Networks, channels, and backplanes differ in terms of the interconnection distances they can support, the bandwidth and latencies they can achieve, and the fundamental assumptions about the inherent unreliability of data transmission. While no statement we can make is universally true, in general, backplanes can be characterized by parallel wide data paths, centralized arbitration, and are oriented towards read/write "memory mapped" operations. That is, access to control registers is treated identically to memory word access. Networks, on the other hand, provide serial data, distributed arbitration, and support more message-oriented protocols. The latter require a more complex handshake, usually involving the exchange of high-level request and acknowledgment messages. Channels fall between the two extremes, consisting of wide datapaths of medium distance and often incorporating simplified versions of network-like protocols.

These considerations are summarized in Table 2.1. Networks typically span more than 1 km, sustain 10 Mbit/second (Ethernet) to 100 Mbit/second (FDDI) and beyond, experience latencies measured in several ms, and the network medium itself is considered to be inherently unreliable. Networks include extensive data integrity features within their protocols, including CRC checksums at the packet and message levels, and the explicit acknowledgment of received packets.

Channels span small 10's of meters, transmit at anywhere from 4.5 MBytes/s (IBM channel interfaces) to 100 MBytes/second (HiPPI channels), incur latencies of under 100 μs per transfer, and have medium reliability. Byte parity at the individual transfer word is usually supported, although packet-level checksumming might also be supported.

Backplanes are about 1 m in length, transfer from 40 (VME) to over 100 (FutureBus) MBytes/second, incur sub μs latencies, and the interconnect is considered to be highly reliable. Backplanes typically support byte parity, although some backplanes (unfortunately) dispense with parity altogether.

	Network	Channel	Backplane
Distance	>1000 m	10 - 100 m	1 m
Bandwidth	10 - 100 Mb/s	40 - 1000 Mb/s	320 - 1000+ Mb/s
Latency	high (>ms)	medium	low (<µs)
Reliability	low Extensive CRC	medium Byte Parity	high Byte Parity

Table 2.1: *Comparison of Network, Channel, and Backplane Attributes*
The comparison is based upon the interconnection distance, transmission bandwidth, transmission latency, inherent reliability, and typical techniques for improving data integrity.

In the remainder of this section, we will look at each of the three kinds of interconnect, network, channel, and backplane, in more detail.

2.2. Communications Networks and Network Controllers

An excellent overview of networking technology can be found in [Cerf 91]. For a futuristic view, see [Tesla 91] and [Negraponte 91]. The decade of the 1980's has seen a slow maturation of network technology, but the 1990's promise much more rapid developments. 10 Mbit/second Ethernets are pervasive today, with many environments advancing to the next generation of 100 Mbit/second networks based on the FDDI (Fiber Distributed Data Interface) standard [Joshi 86]. FDDI provides higher bandwidth, longer distances, and reduced error rates, due largely to the introduction of fiber optics for data transmission. Unfortunately cost, especially for replacing the existing copper wire network with fiber, coupled with disappointing transmission latencies, have slowed the acceptance of these higher speed networks. The latency problems have more to do with FDDI's protocols, which are based on a token passing arbitration scheme, than anything intrinsic in fiber optic technology.

A network system is decomposed into multiple protocol layers, from the application interface down to the method of physical communication of bits on the network. Figure 2.1 summarizes the popular seven layer ISO protocol model. The *physical* and *link* levels are closely tied to the underlying transport medium, and deal with the physical attachment to the network and the method of acquiring access to it. The *network*, *transport*, and *session* levels focus on the detailed formats of communications packets and the methods for transmitting them from one program to another. The *presentation* and *applications* layers define the formats of the data embedded within the packets and the application-specific semantics of that data.

A number of performance measurements of network transmission services all point out that the significant overhead is not protocol interpretation (approximately 10% of instructions are spent in interpreting the network headers). The culprits are memory system overheads due to data movement and operating system overheads related to context switches and data copying [Clark 89, Heatly 89, Kanakia 90, Watson 87]. We will see this again and again in the sections to follow.

The network controller is the collection of hardware and firmware that implements the interface between the network and the host processor. It is typically implemented on a small printed circuit board, and contains its own processor, memory mapped control registers, interface to the network, and small memory to hold messages being transmitted and received. The on-board processor, usually in conjunction with VLSI components within the network interface, implements

Application	Detailed information about the data being exchanged
Presentation	Data representation
Session	Management of connections between programs
Transport	Delivery of packet sequences
Network	Format of individual packets
Link	Access to and control of transmission medium
Physical	Medium of transmission

Figure 2.1: *Seven Layer ISO Protocol Model*
The figure shows the seven layers of the ISO protocol model. The physical layer describes the actual transmission medium, be it coax cable, fiber optics, or a parallel backplane. The link layer describes how stations gain access to the medium. This layer deals with the protocols for arbitrating for and obtaining grant permission to the media. The network layer defines the format of data packets to be transmitted over the media, including destination and sender information as well as any checksums. The transport layer is responsible for the reliable delivery of packets. The session layer establishes communications between the sending program and the receiving program. The presentation layer determines the detailed formats of the data embedded within packets. The application layer has the responsibility of understanding how this data should be interpreted within an applications context.

the physical and link level protocols of the network.

The interaction between the network controller and the host's memory is depicted in Figure 2.2. Lists of blocks containing packets to be sent and packets that have been received are maintained in the host processor's memory. The locations of buffers for these blocks are made known to the network controller, and it will copy packets to and from the request/receive block areas using direct memory access (DMA) techniques. This means that the copy of data across the peripheral bus is under the control of the network controller, and does not require the intervention of the host processor. The controller will interrupt the host whenever a message has been received or sent.

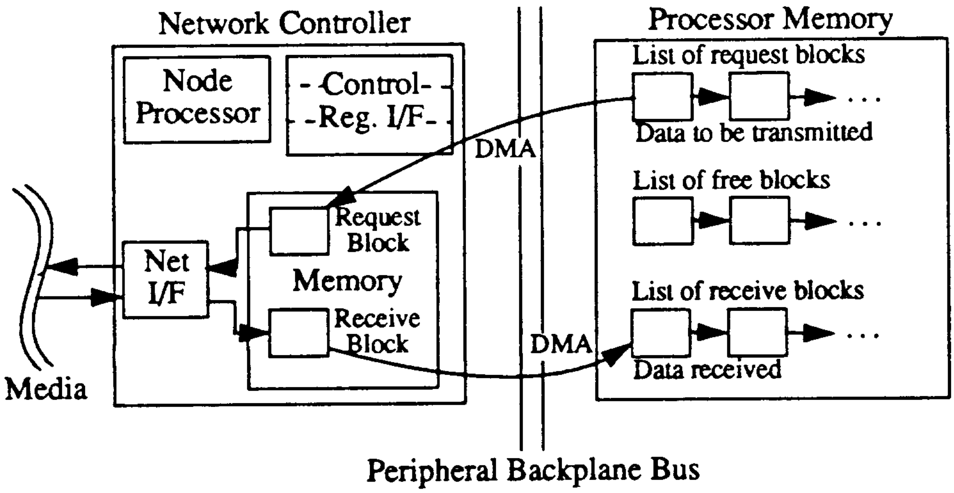


Figure 2.2: *Network Controller/Processor Memory Interaction*
The figure describes the interaction between the Network Controller and the memory of the network node. The controller contains an on-board microprocessor, various memory-mapped control registers through which service requests can be made and status checked, a physical interface to the network media, and a buffer memory to hold request and receive blocks. These contain network messages to be transmitted or which have been received respectively. A list of pending requests and messages already received reside in the host processor's memory. Direct memory operations (DMA), under the control of the node processor, copy these blocks to and from this memory.

While this presents a particularly clean interface between the network controller and the operating system, it points out some of the intrinsic memory system latencies that reduce network performance. Consider a message that will be transmitted to the network. First the contents of the message are created within a user application. A call to the operating system results in a process switch and a data copy from the user's address space to the operating system's area. A protocol-specific network header is then appended to the data to form a packaged network message. This must be copied one more time, to place the message into a request block that can be accessed by the network controller. The final copy is the DMA operation that moves the message within the request block to memory within the network controller.

Data integrity is the aspect of system reliability concerned with the transmission of correct data and the explicit flagging of incorrect data. An overriding consideration of network protocols is their concern with reliable transmission. Because of the distances involved and the complexity of the transmission path, network transmission is inherently lossy. The solution is to append checksum protection bits to all network packets and to include explicit acknowledgment as part of the network protocols. For example, if the checksum computed at the receiving end does not match the transmitted checksum, the receiver sends a negative acknowledgment to the sender.

2.3. Channel Architectures

Channels provide the logical and physical pathways between I/O controllers and storage devices. They are medium distance interconnect that carry signals in parallel, usually with some parity technique to provide data integrity. In this section, we will describe two alternative channel organizations that characterize the low end and high end respectively: SCSI (Small Computer System Interface) and HiPPI (High Performance Parallel Interface).

2.3.1. Small Computer System Interface

SCSI is the channel interface most frequently encountered in small formfactor (5.25" diameter and smaller) disk drives, as well as a wide variety of peripherals such as tape drives, optical disk readers, and image scanners. SCSI treats peripheral devices in a largely device-independent fashion. For example, a disk drive is viewed as a linear byte stream; its detailed structure in terms of sectors, tracks, and cylinders is not visible through the SCSI interface.

A SCSI channel can support up to 8 devices sharing a common bus with an 8-bit wide datapath. In SCSI terminology, the I/O controller counts as one of these devices, and is called the *host bus adapter* (HBA). Burst transfers at 4 to 5 MBytes/second are widely available today. In SCSI terminology, a device that requests service from another device is called the master or the *initiator*. The device that is providing the service is called the slave or the *target*.

SCSI provides a high-level message-based protocol for communications among initiators and targets. While this makes it possible to mix widely different kinds on devices on the same channel, it does lead to relatively high overheads. The protocol has been designed to allow initiators to manage multiple simultaneous operations. Targets are intelligent in the sense that they explicitly notify the initiator when they are ready to transmit data or when they need to throttle a transfer.

It is worthwhile to examine the SCSI protocol in some detail, to clearly distinguish what it does from the kinds of messages exchanged on a computer network. The SCSI protocol proceeds in a series of phases, which we summarize below:

Bus Free: No device currently has the bus allocated

Arbitration: Initiators arbitrate for access to the bus. A device's physical address determines its priority.

Selection: The initiator informs the target that it will participate in an I/O operation.

Reselection: The target informs the initiator that an outstanding operation is to be resumed. For example, an operation could have been previously suspended because the I/O device had to obtain more data.

Command: Command bytes are written to the target by the initiator. The target begins executing the operation.

Data Transfer: The protocol supports two forms of the data transfer phase, *Data In* and *Data Out*. The former refers to the movement of data from the target to the initiator. In the latter, data moves from the initiator to the target.

Message: The message phase also comes in two forms, *Message In* and *Message Out*. Message In consists of several alternatives. *Identify* identifies the reselected target. *Save Data Pointer* saves the place in the current data transfer if the target is about to disconnect. *Restore Data Pointer* restores this pointer. *Disconnect* notifies the initiator that the target is about to give up the data bus. *Command Complete* occurs when the target tells the initiator that the operation has completed. Message Out has just one form: *Identify*. This is used to identify the requesting initiator and its intended target.

Status: Just before command completion, the target sends a status message to the initiator.

To better understand the sequencing among the phases, see Figure 2.3. This illustrates the phase transitions for a typical SCSI read operation. The sequencing of an I/O operation actually begins when the host's operating system establishes data and status blocks within its memory. Next, it issues an I/O command to the HBA, passing it pointers to command, status, and data blocks, as well as the SCSI address of the target device. These are staged from host memory to device-specific queues within the HBA's memory using direct memory access techniques.

Now the I/O operation can begin in ernest. The HBA arbitrates for and wins control of the SCSI bus. It then indicates the target device it wishes to communicate with during the selection phase. The target responds by identifying itself during a following message out phase. Now the actual command, such as "read a sequence of bytes," is transmitted to the device.

We assume that the target device is a disk. If the disk must first seek before it can obtain the requested data, it will disconnect from the bus. It sends a disconnect message to the initiator, which in turn gives up the bus. Note that the HBA can communicate with other devices on the SCSI channel, initiating additional I/O operations. Now the device will seek to the appropriate track and will begin to fill its internal buffer with data. At this point, it needs to reestablish communications with the HBA. The device now arbitrates for and wins control of the bus. It next enters the reselection phase, and identifies itself to the initiator to reestablish communications.

The data transfer phase can now begin. Data is transferred one byte at a time using a simple request/acknowledgment protocol between the target and the initiator. This continues until the need for a disconnect arises again, such as when the target's buffer is emptied, or perhaps the command has completed. If it is the first case, the data pointer must first be saved within the HBA, so we can restart the transfer at a later time. Once the data transfer pointer has been saved, the target sequences through a disconnect, as described above.

When the disk is once again ready to transfer, it rearbitrates for the bus and identifies the initiator with which to reconnect. This is followed by a restore data pointer message to restablish the current position within the data transfer. The data transfer phase can now continue where it left

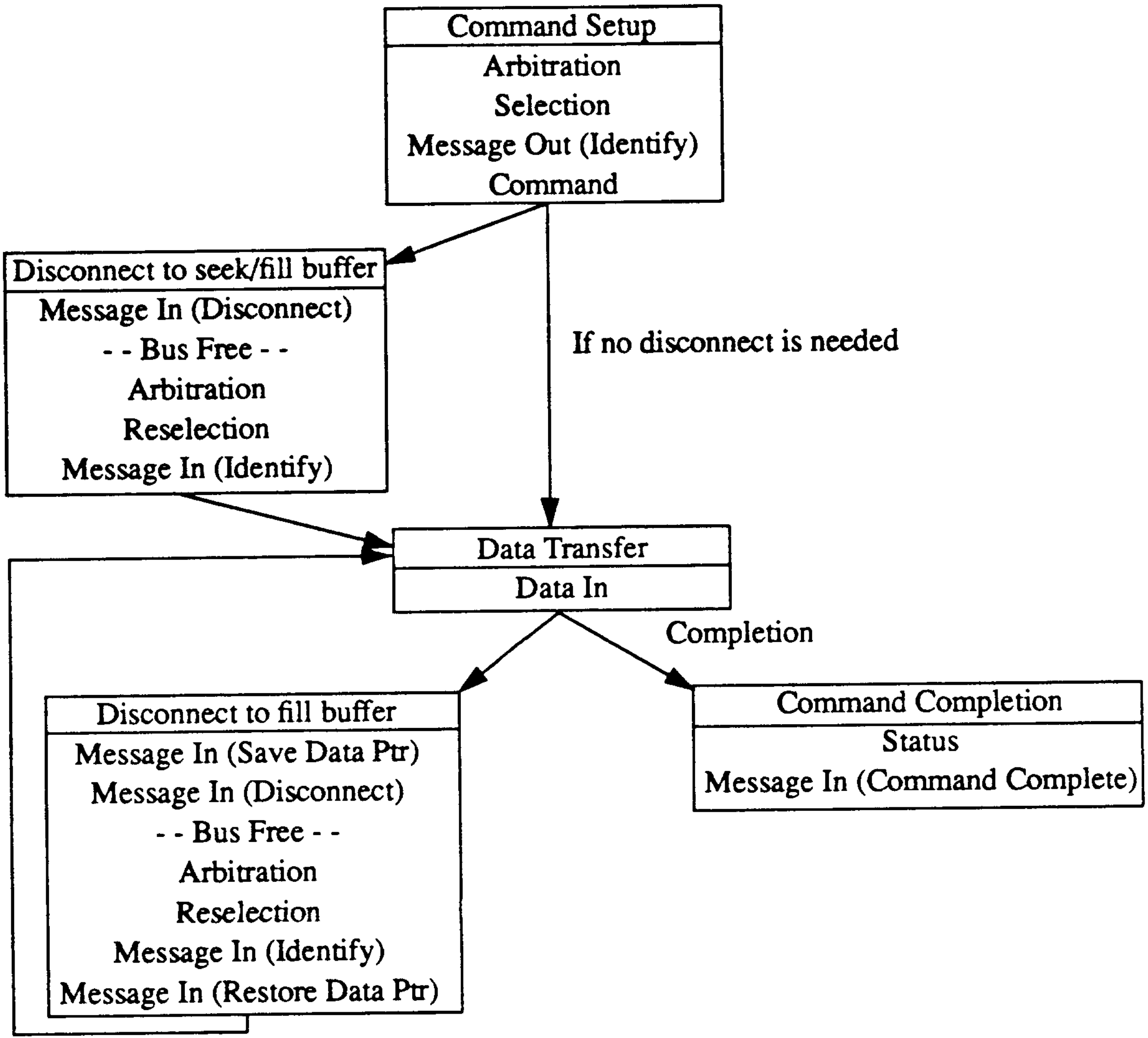


Figure 2.3: *SCSI Phase Transitions on a Read*
The basic phase sequencing for a read (from disk) operation is shown. First the initiator sets up the read command and sends it to the I/O device. The target device disconnects from the SCSI bus to perform a seek and to begin to fill its internal buffer. It then transfers the data to the initiator. This may be interspersed with additional disconnects, as the transfer gets ahead of the internal buffering. A command complete message terminates the operation. This figure is adapted from [Chervenak90].

off.

The command completion phase is entered once the data transfer is finished. The target device sends a status message to the initiator, describing any errors that may have been encountered during the operation. The final command completion message completes the I/O operation.

The SCSI protocol specification is currently undergoing a major revision for higher performance. In the so-called "SCSI-1," the basic clock rate on the channel is 10 Mhz. In the new SCSI-2, "fast SCSI" increases the clock rate to 20 Mhz, doubling the channel's bandwidth from 5 MByte/second to 10 MByte/second. Recently announced high performance disk drives, such as those from Fujitsu, support fast SCSI. The revised specification also supports an alternative method of doubling the channel bandwidth, called "wide SCSI." This provides a 16-bit data path on the channel rather than SCSI-1's 8-bit width. By combining wide and fast SCSI-2, the channel bandwidth quadruples to 20 MByte/second. Some manufacturers of high performance disk controllers have begun to use SCSI-2 to interface their controllers to a computer host.

2.3.2. High Performance Parallel Interface

The High Performance Parallel Interface, HiPPI, was originally developed at the Los Alamos National Laboratory in the mid-1980s as a high speed unidirectional (simplex) point-to-point interface between supercomputers [Ohrenstein 90]. Thus, two-way communications requires two HiPPI channels, one for commands and write data (the *write channel*) and one for status and read data (the *read channel*). Data is transmitted at a nominal rate of 800 Mbits/second (32-bit wide datapath) or 1600 Mbit/second (64-bit wide datapath) in each direction.

The physical interface of the HiPPI channel was standardized in the late 1980s. Its data transfer protocol was designed to be extremely simple and fast. The source of the transfer must first assert a request signal to gain access to the channel. A connection signal grants the channel to the source. However, the source cannot send until the destination asserts ready. This provides a simple flow control mechanism.

The minimum unit of data transfer is the *burst*. A burst consists of 1 to 256 words (the width is determined by the physical width of the channel; for a 32-bit channel, a burst is 1024 bytes), sent as a continuous stream of words, one per clock period. A burst is in progress as long as the channel's burst signal is asserted. When the burst signal goes unasserted, a CRC (cyclic redundancy check) word computed over the transmitted data words is sent down the channel. Because of the way the protocol is defined, when the destination asserts ready, it means that it must be able to accept a complete burst.

Unfortunately, the Upper Level Protocol (ULP) for performing operations over the channel is still under discussion within the standardization committees. To illustrate the concepts involved in using HiPPI as an interface to storage devices, we restrict our description to the proposal to layer the IPI-3 Device Generic Command Set on top of HiPPI, put forward by Maximum Strategies and IBM Corporation [Maximum Strategies 90].

A logical unit of data, sent from a source to a destination, is called a *packet*. A packet is a sequence of bursts. A special channel signal delineates the start of a new packet. Packets consist of a header, a ULP (Upper Layer Protocol) data set, and fill. The ULP data consists of a command/response field and read/write data field.

Packets fall into three types: *command*, *response*, or *data-only*. A command packet can contain a header burst with an IPI-3 device command, such as read or write, followed by multiple data bursts if the command is a write. A response packet is similar. It contains an IPI-3 response within a header burst, followed by data bursts if the response is a read transfer notification. Data-only packets contain header bursts without command or response fields.

Consider a read operation over a HiPPI channel using the IPI-3 protocol. On the write-channel, the slave peripheral device receives a header burst containing a valid read command from the master host processor. This causes the slave to initiate its read operation. When data is available, the slave must gain access to the read-channel. When the master is ready to receive, the slave will transmit its response packet. If the response packet contains a transfer notification status, this indicates that the slave is ready to transmit a stream of data. The master will pulse a ready signal to receive subsequent data bursts.

2.4. Backplane Architecture

Backplanes are designed to interconnect processors, memory, and peripheral controllers (such as network and disk controllers). They are relatively wide, but short distance. The short distances make it possible to use fast, centralized arbitration techniques and to perform data transfers at a higher clock rate. Backplane protocols make use of addresses and read/write operations, rather

Metric	VME	FutureBus	MultiBus II	SCSI-I
Bus Width (signals)	128	96	96	25
Address/Data Multiplexed?	No	Yes	Yes	na
Data Width	16 - 32	32	32	8
Xfer Size	Single/Multiple	Single/Multiple	Single/Multiple	Single/Multiple
# of Bus Masters	Multiple	Multiple	Multiple	Multiple
Split Transactions	No	Optional	Optional	Optional
Clocking	Async	Async	Sync	Either
Bandwidth, Single Word (0 ns mem)	25	37	20	5, 1.5
Bandwidth, Single Word (150 ns mem)	12.9	15.5	10	5, 1.5
Bandwidth Multiple Word (0 ns mem)	27.9	95.2	40	5, 1.5
Bandwidth Multiple Word (150 ns mem)	13.6	20.8	13.3	5, 1.5
Max # of devices	21	20	21	7
Max Bus Length	.5 m	.5 m	.5 m	25 m
Standard	IEEE 1014	IEEE 896	ANSI/IEEE 1296	ANSI X3.131

Table 2.2: *Comparisons Among Popular Backplane and Channel Interconnects*

than the more message-oriented protocols to be found on networks and channels.

Table 2.2 gives some of the metrics of three popular backplane busses, VME, FutureBus, and Multibus II (this table is adapted from [Hennessy 90]). For the purposes of comparison, we include the same metrics for the SCSI-I channel specification. The table includes the width of the interconnect (including control and data signals), whether the address and data lines are multiplexed, the data width, whether the transfer size is a single or multiple word, the number of bus masters supported, whether split transactions are supported (these are network-like request and acknowledgment messages), the clocking scheme, the interconnect's bandwidth under a variety of assumptions (single vs. multiple word transfers, 0 ns access time memories vs. 150 ns access time), the maximum number of controllers or devices per bus, the maximum bus length, and the relevant ANSI or IEEE standard that defines the interconnect.

The most dramatic differences are in the interconnect width and the maximum bus width. In general, channel interconnects are narrow and long distance while backplanes are wide but short distance.

However, some of the distinctions are being to blur. The SCSI channel has many of the attributes of a bus, FutureBus has certain aspects that make it behave more like a channel than a bus, and nobody could describe a 64-bit HiPPI channel as being narrow! For example, let's consider FutureBus in a little more detail. The bus supports distributed arbitration, asynchronous signaling (that is, no global clocks), single source/muliple destination "broadcast" messages, and request/acknowledge split bus transactions [Borrill 84]. The latter are very much like SCSI disconnect/reconnect phases. A host issues a read request message to a memory or I/O controller, and then detaches from the bus. Later on, the memory sends a response message to the host, containing the requested data.

3. Storage Trends

3.1. The Storage Hierarchy and Storage Technology

3.1.1. Concept of Storage Hierarchy

The storage hierarchy is traditionally modeled as a pyramid, with a small amount of expensive, fast storage at the pinnacle and larger capacity, lower cost, and lower performance storage as we

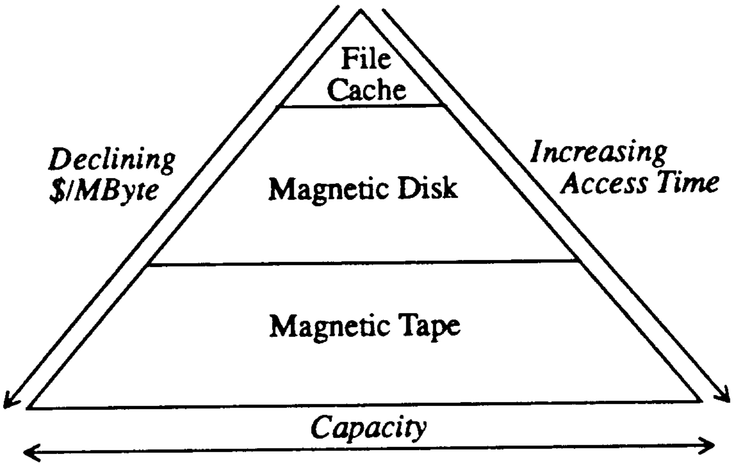


Figure 3.1: *Typical Storage Hierarchy, Circa 1980*
The microsecond access is provided by the file cache, a small number of bytes stored in semiconductor memory. Medium capacity, denominated in several hundred megabytes with tens of millisecond access, is provided by disk. Tape provides unlimited capacity, but access is restricted to tens of seconds to minutes.

move towards the base. In general, there are order of magnitude differences in capacity, access time, and cost among the layers of the hierarchy. For example, main memory is measured in megabytes, costing approximately $50/MByte, and can be accessed in small numbers of microseconds. Secondary storage, usually implemented by magnetic disk, is measured in gigabytes, costs below $5/MByte, and is accessed in tens of milliseconds. The operating system can create the illusion of a large fast memory by judiciously staging data among the levels. However, the organization of the storage hierarchy must adapt as magnetic and optical recording methods continue to improve and as new storage devices become available.

Figure 3.1 depicts the storage hierarchy of a typical minicomputer of 1980. (It should be noted that large mainframe and supercomputer storage hierarchies were more complex than what is depicted here.) A small file cache (or buffer), allocated by the operating system from the machine's semiconductor memory, provides the fastest but most expensive access. The job of the cache is to hold data likely to be accessed in the near future, because it is near data recently accessed (*spatial locality*) or because it has recently been accessed itself (*temporal locality*). Prefetching is a strategy that accesses larger chunks of file data than requested by an application, in the hope that it will soon access spatially local data.

Either a buffer or a cache can be used to decouple application accesses in small units from the larger units needed to efficiently utilize secondary storage devices. It is not efficient to amortize the millisecond latency cost to access secondary storage for a small number of bytes. Accesses in the range of 512 to 8192 bytes are more appropriate. The primary distinction between application memory and a cache is the latter's ability to keep resident certain data. For example, frequently accessed file directories can be held in an cache, thus avoiding slow accesses to the lower levels of the hierarchy.

Secondary storage is provided by magnetic disk. Data is recorded on concentric tracks on stacked platters, which have been coated with magnetic materials. The same track position across the platters is called a cylinder. A mechanical actuator positions the read-write heads to the desired recording track, while a motor rotates the platters containing the data under the heads.

Tertiary storage, provided primarily for archive/back-up, is implemented by magnetic tape. A spool of magnetic tape is drawn across the read-write mechanism in a sequential fashion. A

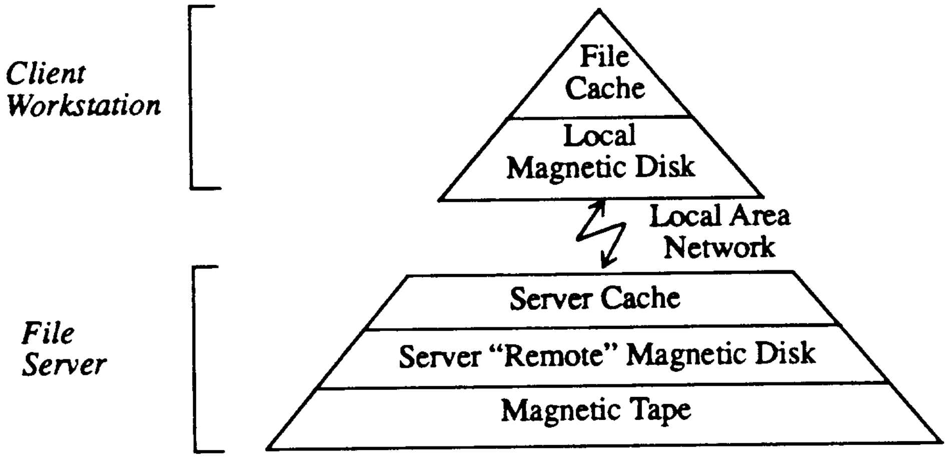


Figure 3.2: *Typical Storage Hierarchy, Circa 1990*
The file cache has become substantially larger, and may be partially duplicated at the client in addition to the server. Secondary storage is split between local and remote disk. Tape continues to provide the third level of storage.

good rule of thumb for a unit of tertiary storage media, such as a tape spool, is that it should have as much capacity as the secondary storage devices it is meant to back up. As disk devices continue to improve in capacity, tertiary storage media are driven to keep pace.

In 1980, a typical machine of this class would have one to two megabytes of semiconductor memory, of which only a few thousand bytes might be allocated for input/output buffers or file system caches. The secondary storage level might include a few hundred megabytes of magnetic disk. The tape storage level is limited only by the amount of shelf space in the machine room.

3.1.2. Evolution of the Storage Hierarchy

Figure 3.2 shows the storage hierarchy distributed across a workstation/server environment of today. Most of the semi-conductor memory in the server can be dedicated to the cache function because a server does not host conventional user applications. The file system "meta-data," that is, the data structures describing how logical files are mapped onto physical disk blocks, can be held in fast semiconductor memory. This represents much of the active portion of the file system. Thus, disk latency can be avoided while servicing user requests.

The critical challenge for workstation/server environments is the added latency of network communications. These are comparable to those of magnetic disk, and are measured in small tens of milliseconds. The figure shows one possible solution, which places small high performance disks in the workstation, with larger potentially slower disks at the server.

If most accesses can be serviced by the local disks, the network latencies can be avoided altogether, improving client performance and responsiveness. However, there are several choices for how to partition the file system between the clients and the servers. Each of these partitionings represents a different tradeoff between system cost, the number of clients per server, and the ease of managing the clients' files.

A *swapful* client allocates the virtual memory swap space and temporary files to its local disk. The operating system's files and user files remain on the server. This reduces some of the network traffic to the server, leaving the issues of system management relatively uncomplicated. For example, in this configuration, the local disk does not need to be backed up. However, to execute an operating system command still requires an access to the remote server.

A *dataless* client adds the operating system's files to the client's local disk. This further reduces the client's demand on the server, thus making it possible for a single server and network to support more clients. While it is still not necessary to back up the local disk, the system is more difficult to administer. For example, system updates must now be distributed to all of the workstations.

A *diskfull* client places all but some frequently shared files on the client. This yields the lowest demands on the server, but represents the biggest problems for system management. Now the personal files on the local disk need to be backed up, leading to significant network traffic during backup operations.

An alternative approach leverages the lower cost semiconductor memory to make feasible large file caches (approximately 25% of the available memory within the workstation) in the client workstation. These "client" caches provide an effective way to circumvent network latencies, if the network protocols allow file writes to be decoupled from communications with the server (see the discussion of NFS protocols in the next section). The approach, called *diskless* clients, has been used with great success in the Sprite Network Operating System [Nelson 88], where they report an ability to support 5-10 times as many clients per server as more conventional client/server organizations.

Figure 3.3 depicts one possible scenario for the storage hierarchy of 1995. Three major technical innovations shape the organization: *disk arrays*, *near-line storage subsystems* based on optical disk or automated tape libraries, and *network distribution*. We concentrate on disk arrays and near-line storage system technology in the remainder of this subsection. Network distribution is covered in Section 4.

Disk Arrays

Because of the rapidly decreasing formfactor of magnetic disks, it is becoming attractive to replace a small number of large disk drives with very many small drives. The resulting secondary storage system can have much higher capacity since small format drives traditionally obtain the highest areal densities. And since the performance of both large and small disk drives is limited by

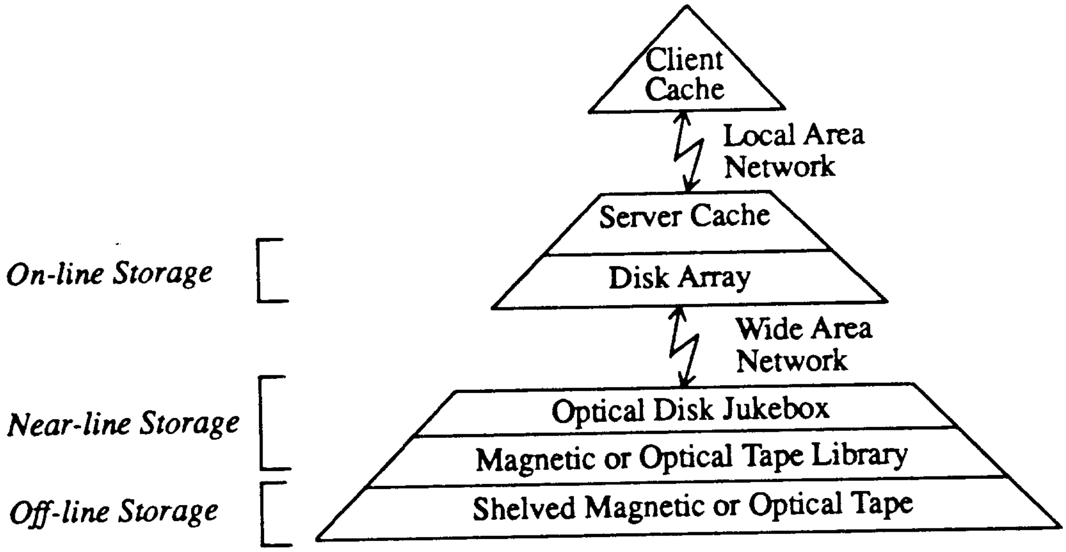


Figure 3.3: *Typical Storage Hierarchy, Circa 1995*
Conventional disks have been replaced by disk arrays, a method of obtaining much higher I/O bandwidth by striping data across multiple disks. A new level of storage, "near line", emerges between disk and tape. It provides very high capacity, but at access times measured in seconds.

mechanical delays, it is no surprise that performance can be dramatically improved if the data to be accessed is spread across many disk actuators. Disk arrays provide a method of organizing many disk drives to appear logically as a very reliable single drive of high capacity and high performance [Katz 89].

Disk array organizations are organized into a multilevel taxonomy. Here, we concentrate on the two most prevalent RAID organizations: RAID Level 3 and RAID Level 5. Each of these spreads data across N data disks and an N+1st redundancy disk. The group of N+1 disks is called a *stripe set*. In a RAID Level 3 organization, data is interleaved in large blocks (for example, a track or cylinder) across all of the disks within a stripe set. The redundancy disk contains a parity bit computed bit-wise across the rows of bits on the associated data disks. If a disk should fail, its contents can be reconstructed simply by examining the surviving N disks and restoring the sense of the parity computed across the bit rows. Suppose that the redundancy disk contained odd parity before the failure. If, after a disk failure, the examination of a bit row yields even parity, then the failed disk must have had a 1 in that bit row. Similarly, if the row has odd parity, then the missing bit must have been a 0. RAID Level 3 organizations are read and written in full stripe units, simultaneously accessing all disks in the stripe. The organization is most suitable for high bandwidth applications such as image processing and scientific computing.

If RAID Level 3 is organized for high bandwidth, then RAID Level 5 is organized for high I/O rate. The basic organization is the same: a stripe set of N data disks and one redundancy disk. However, data is accessed in smaller units, thus making it possible to support multiple simultaneous accesses. Consider a data write operation to a single disk sector. This requires the parity redundancy to be updated as well. We accomplish this by first determining the bit changes to the data sector and then invert exactly these bits in the associated parity sector. Thus a logical single sector write may involve four physical disk accesses: read old data, read old parity, write new data, and write new parity. Since placing all parity sectors on a single disk drive would limit the array to a single write operation at a time, the parity sectors are actually interleaved across all disks of the stripe. A RAID Level 5 can perform N+1 simultaneous reads and $\lfloor (N+1)/2 \rfloor$ simultaneous writes (in the best case).

Near-Line Storage

A comparable revolution has taken place in tertiary storage: the arrival of near-line storage systems. These provide relatively rapid access to enormous amounts of data, frequently stored on removable, easy to handle optical disk or magnetic tape media. This is accomplished by storing the high capacity media on shelves that can be accessed by robotic media "pickers." When a file needs to be accessed, special file management software identifies where it can be found within the tape or optical disk library. The picker exchanges the currently loaded media with the one containing the file to be accessed. This is accomplished within a small number of seconds, without any intervention by human operators. By carefully exploiting caching techniques, in particular, using the secondary storage devices as a cache for the near-line store, the very large storage capacity of a tertiary storage system can appear to have access times comparable to magnetic disks at a fraction of the cost. We describe the underlying storage technologies next.

Optical Disk Technology for Near-line Storage

Optically recorded disks have long been thought to be ideal for filling the near line level of the storage hierarchy [Ranade 90]. They combine improved storage capacity (2 GBytes per platter surface originally to over 6 GBytes per side today) with access times that are approximately a factor of ten slower than conventional magnetic disks (several hundred milliseconds). The first generation of optical disks were written once, but could be read many times, leading to the term "WORM" to describe the technology. The disk is written by a laser beam. When it is turned on, it records data in the form of pits or bubbles in a writing layer within the disk. The data is read back by detecting the variations of reflectivity of the disk surface.

The write once nature of optical storage actually makes it better suited for an archival medium than near-line storage, since it is impossible to accidently overwrite data once it has been written. A problem has been its relatively slow transfer rate, 100K – 200K bytes per second. Newer generations of optical drives now exceed one megabyte per second transfers.

Magneto-optical technologies, based on a combination of optical and magnetic recording techniques, have recently led to the availability of erasable optical drives. The disk is made of a material that becomes more sensitive to magnetic fields at high temperatures. A laser beam is used to selectively heat up the disk surface, and once heated, a small magnetic field is used to record on the surface. Optical techniques are used for reading the disk, by detecting how the laser beam is deflected by different magnetizations of the disk surface. Read transfer rates are comparable to that of conventional magnetic disks. Access times are still slower than a magnetic disk due to the more massive read/write mechanism holding the laser optics, which takes longer to position than the equivalent low mass magnetic read/write head assembly. The write transfer rate is worse in optical disk systems because (1) the disk surface must first be erased before new data can be recorded, and (2) the written data must be reread to verify that it was written correctly to the disk surface. Thus, a write operation could require three disk revolutions before it completes. ([Kryder 89] details the trends and technology challenges for future optical disk technologies).

Nevertheless, as the formfactor and price of optical drives continue to decrease, optical disk libraries are becoming more pervasive. Sony's recent announcement of a consumer-oriented recordable music compact disk could lead to dramatic reductions in the cost of optical disk technology. As an example of an inexpensive optical disk system, let's examine the Hewlett-Packard Series 6300 Model 20GB/A Optical Disk Library System [Hewlett-Packard 89]. Based on 5.25" rewritable optical disk technology, the system provides two optical drives, 32 read/write optical disk cartridges (approximately 600 Mbytes per cartridge), and a robotic disk changer that can move cartridges to and from the drives, all in a desk side unit the size of a three drawer filing cabinet. The optical cartridges can be exchanged in 7 seconds, and require a 4 second load time and 2.4 second spin-up time. The unload and spin-down times are 2.8 and 0.8 seconds respectively. An average seek time requires 95 ms. The drives can sustain 680 KBytes/second transfers on reads and 340 KBytes/second transfers on writes.

The Kodak Optical Disk System 6800 Automated Disk Library is characteristic of the high end [Kodak 90]. The system can be configured with 50 to 150 optical disk platters, and 1 to 3 optical disk drives.It is capable of storing from 340 GBytes to 1020 GBytes (3.4 GBytes for each side of a 14" platter). The average disk change time is 6.5 seconds. The optical disk surface is organized into five bands of varying capacity, with a certain number of tracking windows per band. The drives can sustain 1 MByte/second transfers, with 100 ms access times for data anywhere within the current band to 700 ms for data anywhere on the surface.

Magnetic Tape Technology for Near-line Storage

The sequential nature of the access to magnetic tape has traditionally dictated that it be used as the medium for archive. However, the success of automated tape libraries from Storage Technology Corporation has demonstrated that tape can be used to implement a near-line storage system. The most pervasive magnetic tape technology available today is based on the IBM 3480 half-inch tape cartridge, storing 200 MBytes and providing transfer rates of 3 MBytes per second. A second generation technology recently introduced doubles the tape capacity and transfer rate.

However, there has been an enormous increase in tape capacity, driven primarily by *helical scan recording* methods. In a conventional tape recording system, the tape is pulled across stationary read/write recording heads. Recorded data tracks run in parallel along the length of the tape. On the other hand, helical scan methods slowly move the tape past a rapidly rotating head assembly to achieve a very high tape to head speed. The tape is wrapped at an angle around a rotor assembly, yielding densely packed recording tracks running diagonally across the tape. The technology is based on the same tape transport mechanisms developed for video cassette recorders in the VHS and 8mm tape formats and the newer digital audio tape (DAT) systems.

Each of these systems provide a very high storage capacity in a small easy-to-handle cartridge. The small formfactor make these tapes particularly attractive as the basis for automated data libraries. Tape systems from Exabyte, based on the 8mm video tape format, can store 2.3 GBytes and transfer at approximately 250 KBytes per second. A second generation system now available doubles both the capacity and the transfer rate. A tape library system based on a 19" rack can hold up to four tape readers and over one hundred 8 mm cartridges, thus providing a storage capacity of 250 – 500 GBs [Exabyte 90].

DAT tape provides smaller capacity and bandwidth than 8mm, but enjoys certain other advantages [Tan 89]. Low cost tape readers in the 3.5" formfactor, the size of a personal computer floppy disk drive, are readily available. This makes possible the construction of tape libraries with a higher ratio of tape readers to tape media, increasing the aggregate bandwidth to the near-line storage system. In addition, the DAT tape formats support subindex fields which can be searched at speed two hundred times greater than the normal read/write speed. A given file can be found on a DAT tape in an average search time of only 20 seconds, compared to over ten minutes for the 8mm format.

VHS-based tape systems can transfer up to 4 MBytes/second and can hold up to 15 GBytes per cartridge. Tape robotics in use for the broadcast industry have been adapted to provide a near-line storage function.

Helical scan techniques are not limited to consumer applications, but have also been applied for certain instrument recording applications, such as satellite telemetry, which require high capacity and high bandwidth. These tape systems are called DD1 and DD2. A single tape cartridge can hold up to 150 GBytes, and can transfer at a rate of up to 40 MBytes/second. However, such systems are very expensive, and a good rule of thumb is that the tape recorder will cost $100,000 for each 10 MBytes/second of recording bandwidth it can support.

Optical Tape Technology for Near-Line Storage

A recording technology that appears to be very promising is *optical tape* [Feder 91]. The recording medium is called *digital paper*, a material constructed from an optically sensitive layer that has been coated onto a substrate similar to magnetic tape. The basic recording technique is similar to write once optical disk storage: a laser beam writes pits in the digital paper to indicate the presence (or absence) of a bit. Since the pits have lower reflectivity than the unwritten tape, a reflected laser beam can be used to detect their presence. One 12 inch by 2400 foot reel can hold 1 TB of data, can be read or written at the rate of 3 MBytes per second, and can be accessed in a remarkable average time of 28 seconds.

Two companies are developing tape readers for digital paper: CREO Corporation and LaserTape Corporation. CREO makes use of a 12 inch tape reels and a unique laser scanner array to read and write multiple tracks 32-bits at a time [Spencer 88]. The system is rather expensive, selling for over $200,000. LaserTape places digital paper in a conventional 3480 tape cartridge (50

Technology	Capacity (MB)	BPI	TPI	BPI*TPI (million)	Data Transfer (KBytes/s)	Access Time
Conventional Tape						
Reel-to-Reel (1/2")	140	6250	18	0.11	549	minutes
Cartridge (1/4")	150	12000	104	1.25	92	minutes
IBM 3480 (1/2")	200	22860	38.1	0.87	3000	seconds
Helical Scan Tape						
VHS (1/2")	15000	?	?	?	4000	minutes
Video (8mm)	4600	43200	1638	70.56	492	minutes
DAT (4mm)	1300	61000	1870	114.07	183	20 seconds
Optical Tape						
CREO (35mm)	1TB	9336000	24	224	3000	28 seconds
Magnetic Disk						
Seagate Elite (5.25")	1200	33528	1880	63.01	3000	18 ms
IBM 3390 (10.5")	3800	27940	2235	62.44	4250	20 ms
Floppy Disk (3.5")	2	17434	135	2.35	92	1 second
Optical Disk						
CD ROM (3.5")	540	27600	15875	438.15	183	1 second
Sony MO (5.25")	640	24130	18796	453.54	87.5	100 ms
Kodak (14")	3200	21000	14111	296.33	1000	100's ms

TABLE 3.1: *Relevant Metrics for Alternative Storage Technologies*

GBytes capacity and 3 MBytes per second transfer rate), and replaces a 3480 tape unit's magnetic read/write heads with an inertialess laserbeam scanner. The scanner operates by using a high frequency radio signal of known frequency to vibrate a crystal, which is then transferred to a laser beam to steer it to the desired read/write location. A 3480 tape reader can be "retrofitted" for approximately $20,000. Existing tape library robotics for the 3480 cartridge formfactor can be adapted to LaserTape without changes.

Summary

Table 3.1 summarizes the relevant metrics of the alternative storage technologies, with a special emphasis on helical scan tapes. The metrics displayed are the capacity, bits per inch (BPI), tracks per inch (TPI), areal density (BPI*TPI in millions of bits per square inch), data transfer rate (Kbytes per second sustained transfers), and average positioning times. The latter is especially important for evaluating near-line storage media. An access time measured in a small number of seconds begins to make tape technology attractive for near-line storage applications, since the robotic access times tend to dominate the time it takes to pick, load, and access data on near-line storage media.

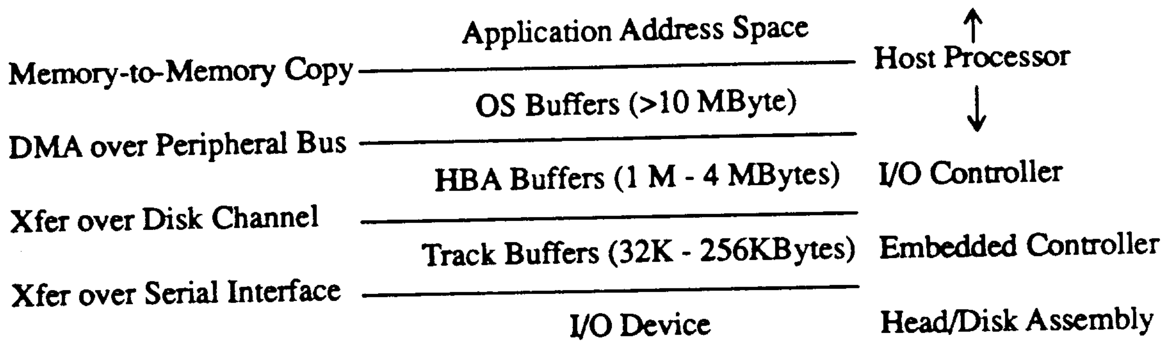


Figure 3.4: *I/O Data Flow*
In response to a read operation, data moves from the device, to the embedded controller, to the I/O controller, to operating system buffers, and finally to the application across a variety of different interfaces.

3.2. Storage Controller Architecture

3.2.1. I/O Data Flow

Figure 3.4 shows the various interfaces across which a typical I/O request must flow. The actual flow of data starts at the I/O device. In the following discussion, we will assume that the device is an intelligent magnetic disk for something like a SCSI interface and that we are considering a read operation. The mechanical portion of the disk drive is called the *head/disk assembly*, or HDA. The control and interface to the outside world is provided by an *embedded controller*.

Data moves across a bit-serial interface from the disk signal processing electronics to track buffers associated with the embedded controller. The amount of memory associated with the track buffers varies from 32 KBytes to 256 KBytes. Since the typical track on today's small formfactor disks is in the range of 32 K – 64 KBytes, a typical embedded controller can buffer more than one track.

The interface between the embedded controller and the host is provided by an I/O controller. We called such a controller a *host bus adapter*, or HBA, in Section 2.3.1. It couples the host peripheral bus to the disk channel interface. Data is staged into buffers within the HBA, from which they are copied out via direct memory access techniques to the host's memory. The typical size of I/O controller buffers is in the range of 1 to 4 MBytes.

The host's memory is coupled to the processor via a high speed cache memory. The connection to the I/O controllers is through a slower speed peripheral bus. Direct memory access operations copy data from the controller's buffers to operating system buffers in main memory. Before the data can be used by the application, it may need to be copied once again, to stage it into a portion of the memory address space that is accessible to the application. Note that the same memory and operating system overheads that limit network performance also affect I/O performance. This is critically important in file and storage servers, where both the I/O and network traffic must be routed through the memory system bottleneck.

If the host is actually a file server, the size of the operating system's buffers may be quite large, perhaps as large as 128 MBytes. In addition, the data flow must be extended to include transfers across the network interconnect into the application's address space on the client. A detailed examination of the operating system management of the I/O path will be left until Section 4.

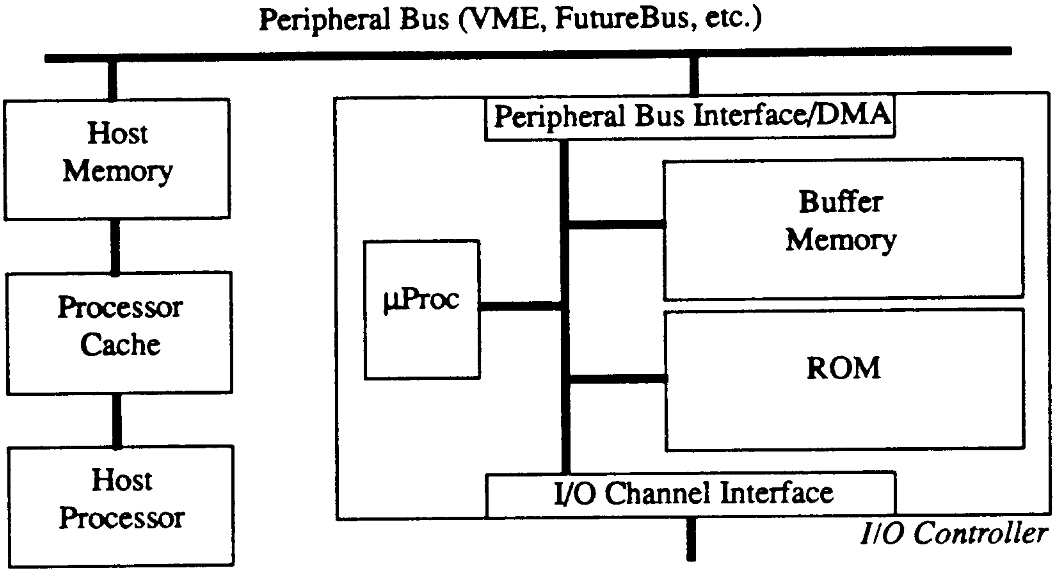


Figure 3.5: *Internal Organization of an I/O Controller*
The I/O controller couples a peripheral bus to the I/O channel via a buffer memory. Hardware in the peripheral interface implements direct memory access between this buffer memory and the host's memory. The I/O channel interface implements the handshaking protocols with the I/O devices. The microprocessor is the "traffic cop," coordinating the actions of the two interfaces.

3.2.2. Internal Organization of I/O Controller

Figure 3.5 shows the internal organization of a typical high performance host bus adapter I/O controller. Interestingly enough, it is not very different in its internal architecture from the network controller of Figure 2.2. Usually implemented on a single printed circuit board, the controller contains a microprocessor, a modest amount of memory dedicated to buffers and run-time data structures, a ROM to hold the controller firmware, a DMA/peripheral bus interface, and an I/O channel interface.

The system interface is also similar to the network controller described previously. Request blocks containing I/O commands and data are organized into as a linked list in the host memory. The host writes to a memory-mapped command register within the I/O controller to initiate an operation. Using DMA techniques, the controller fetches the request blocks into its own memory. The on-board microprocessor unpackages the I/O commands and write data, and sends these over the I/O channel interface. Status and read data are repackaged into response blocks that are copied back to reserved buffers in the host memory. The host can choose whether the I/O controller will interrupt the host whenever an operation has been completed.

The controller of Figure 3.5 is notable because of its support for direct memory access. Some lower performance controllers require that commands and data be written a word (or half word) at a time to memory-mapped controller registers over the peripheral bus. Since a typical command block can be 16 to 32 bytes in length, simply downloading a command may take tens of microseconds, requiring a good deal of host processor intervention.

In implementing a high performance file service on a network, a critical relationship exists between the network and I/O controller architectures. The network interface and the I/O controller must be coupled by a high performance interconnect and memory system. This key observation provides the motivation for several of the systems reviewed in Section 5, especially the prototype being developed at U. C. Berkeley described in Section 5.6.

4. Software Trends

4.1. Network File Systems

One of the most important software developments over the past decade has been the rapid development of the concept of remote file services. In a location transparent manner, these systems provide a client with the ability to access remote files without the need to resort to special naming conventions or special methods for access.

It is important to distinguish between the related concepts of *block server* and *file server*. A block server (sometimes called a "network disk") provides the client with a physical device interface over a network. The block server supports read and write requests to disk blocks, albeit to a disk attached to a remote machine. A file server supports a higher level interface, providing the complete file abstraction to the client. The interface supports file creation, logical reads and writes, file deletion, etc. In a file server, file system related functions are centralized and performed by the server. In a block server, these functions must be handled by the clients, and if the disks are to be shared across machines, this requires distributed coordination among them.

The most ubiquitous file system model is based on that of UNIX, and so we begin our discussion with its structure. A *file* is uniquely named within an hierarchical name space based on directories. As far as the user is concerned, a file is nothing more than an uninterpreted stream of bytes. The file system provides operations for positioning within the file for the purposes of reading and writing bytes. Internally, the file system keeps track of the mapping between the file's logical byte stream and their physical placement within disk blocks through a data structure called an *inode*. The inode is "metadata," that is, data about data, and contains information such as the device containing the file, a list of the physical disk blocks containing the file's data, and pointers to additional disk blocks (called *indirect blocks*) should the file be large enough to exceed the mapping space of a single inode.

From the operating system perspective, tracing an I/O request from the application to disk proceeds as follows. The application program must make a system call, such as *read* or *write*, to request service from the operating system. This is handled by the UNIX System Call Layer, which in turn calls the file system to handle the request in detail. Within the file system are block I/O routines which handle read or write requests. These call a particular disk driver to schedule the actual disk transfers. The software layers are shown in Figure 4.1.

The figure shows the software architecture for a file system on the same machine as the client application. The major innovation of SUN's *Network File System*, or NFS, is its ability to map remote file systems into the directory structure of the client's machine. That is, it is transparent to the user whether the referenced file is available locally or is being accessed over the network. This is accomplished through the new abstraction of a *virtual file system*, or VFS [Sandberg 85]. The VFS interface allows file system requests to be dispatched to the local file system, or sent to a remote server across the network. The generic software layers are shown in Figure 4.2. and the path through the software taken between the client and the server is shown in Figure 4.3.

The access to the remote machine is implemented via a synchronous *remote procedure call* (RPC) mechanism. This is a communications abstraction that behaves much like a conventional procedure call, except that the procedure being invoked may be on a remote "server" machine. Since the RPC is synchronous, the client must wait or *block* until the server has completed the call and returned the requested data or status.

The NFS protocol is a collection of procedure calls and parameters built on top of such a RPC mechanism. One of the key design decisions of NFS is to make this protocol *stateless*. This

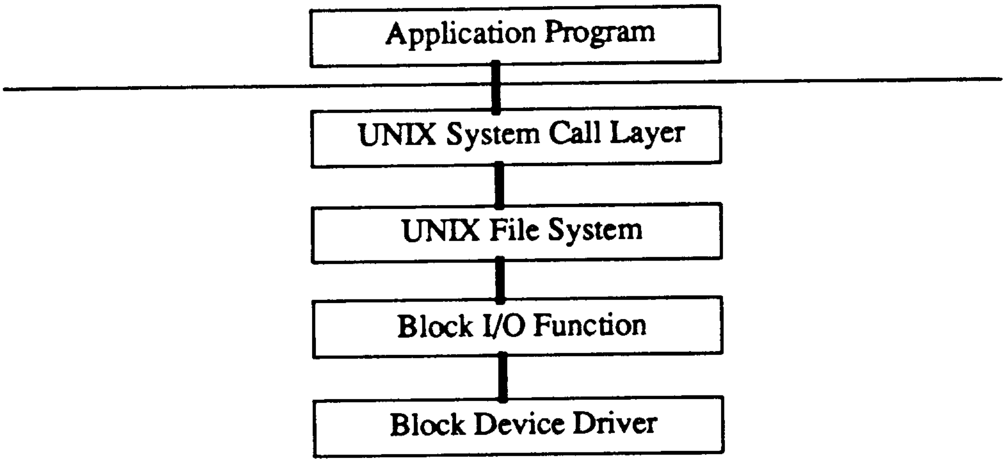


Figure 4.1: *Software Layers in the UNIX File System.*
The UNIX System Call Layer dispatches a read or write request to the File System, which in turn calls a Block I/O routine. This calls a specific device driver to handle the scheduling the I/O request.

means that each procedure call is completely self-describing; the server keeps track of no past requests. This choice was made to drastically reduce the complexity of recovery. In the event of a server crash, the client simply retries its request until it is successfully serviced. As far as the client is concerned, there is no difference between a crashed server and one that is merely slow. The server need not perform any recovery processing. Contrast this with a "stateful" protocol in which both servers and clients must be able to detect and recover from crashes.

However, the stateless protocol has significant implications for file system performance. In order to be stateless, the server must commit any modified user data and file system metadata to stable storage before returning results. This implies that file writes cause the affected data blocks, inodes, and indirect blocks to be written from in-memory caches to disk. In addition, housekeeping operations such as file creation, file removal, and modifications to file attributes must all be performed synchronously to disk.

Some controversy surrounds the real source of bottlenecks in NFS performance. Network protocol overheads and server processing are possible culprits. However, it has now become clear

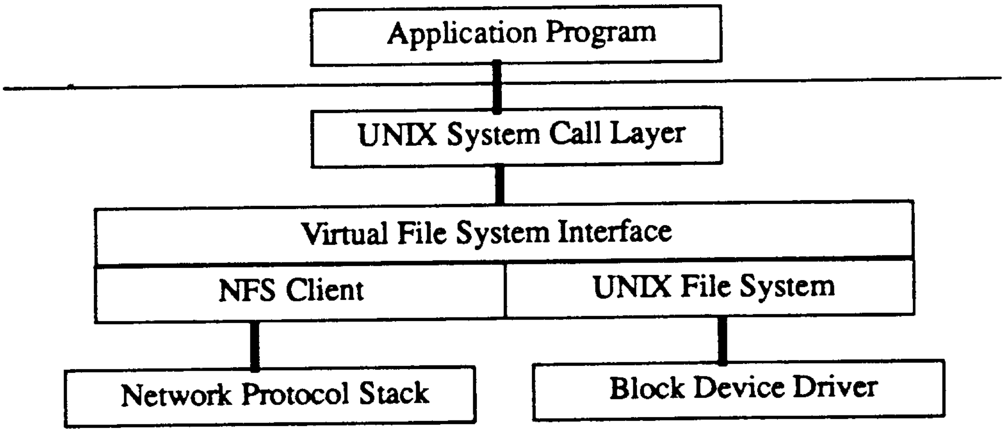


Figure 4.2: *Software Layers in the UNIX File System Extended for NFS*
The VFS interface allows requests to be mapped transparently among local file systems and remote file systems.

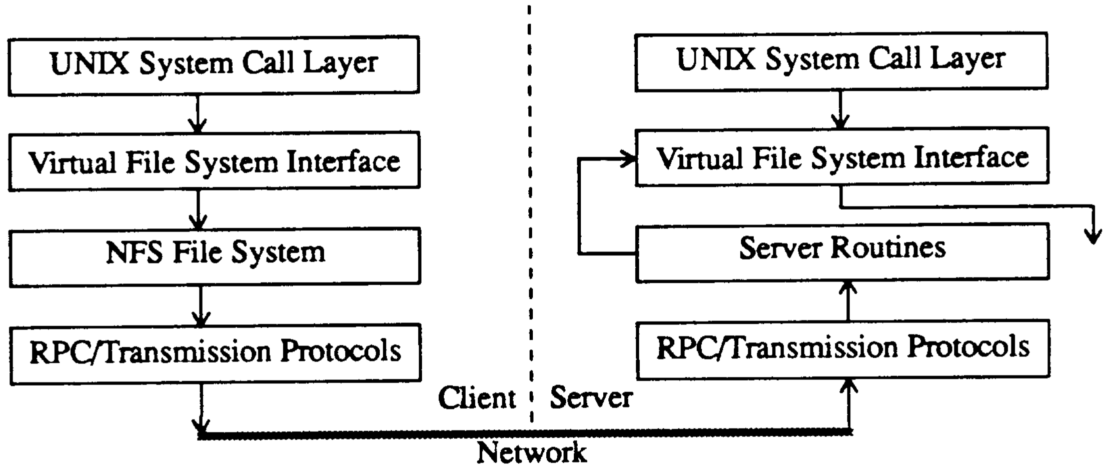

UNIX System Call Layer
Virtual File System Interface
NFS File System
RPC/Transmission Protocols
UNIX System Call Layer
Virtual File System Interface
Server Routines
RPC/Transmission Protocols
Client
Server
Network

Figure 4.3: *Path of an NFS Client to Server Request*
A request to access a remote file is handled by the client's VFS, which maps the request through the NFS layer into RPC calls to the server over the network. At the server end, the requests are presented to the VFS, this time to be mapped into calls on the server's local UNIX file system.

that the real problem is the stateless nature of the NFS protocol, and its associated forced disk writes. By making file system operations synchronous with disk, the performance of the file system is (overly) coupled to the performance of the disk system [Rosenblum 91].

4.2. File Server Architecture

In this subsection, we examine the flow of a network-based I/O request as it arrives at the network interface, through the file server's hardware and software, to the storage devices and back again to the network. Our goal is to bring together the discussions of network interface, I/O controller, and network file system processing of an I/O request, initiated by a client on the network.

Figure 4.4 shows the hardware/software architecture of a conventional workstation-based file server. A data read request arrives at the Ethernet controller. The network messages are copied from the network controller to the server's primary memory. Control passes through the software levels of the *network driver* and *protocol interpretation* to process the request. At the file system level, to avoid unnecessary disk accesses, the server's primary memory is interrogated to determine if the requested data has already been cached from disk.

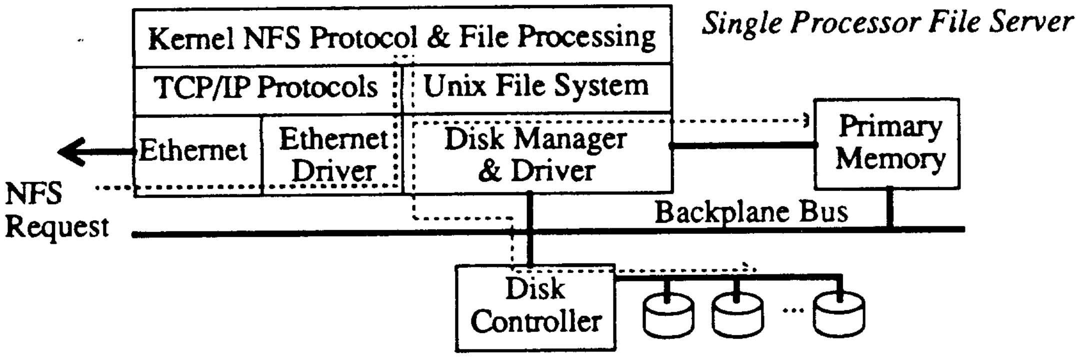


Figure 4.4: *Conventional File Server Architecture*
An NSF I/O request arrives at the Ethernet interface of the server. The request is passed through to the network driver, the protocol processing software, and the file system. The request may be satisfied by data cached in the primary memory; if not, the data must be accessed from disk. At this point, the process is reversed to send the requested data back over the network. This figure is adapted from [Nelson 90].

If the request cannot be satisfied from the file cache, the file system will issue a request to the disk controller. The retrieved data is then staged by the disk controller from the I/O device to the primary memory along the backplane bus. Usually it must be copied (at least) one more time, into templates for the response network messages. The software path returns through the file system, protocol processing, and network drivers. The network response messages are transmitted from the memory out through the network interface.

There are two key problems with this architecture. First, there is the long instruction path associated with processing a network-based I/O request. Second, as we have already seen, the memory system and the backplane bus form a serious performance bottleneck. Data must flow from disk to memory to network, passing through the memory and along the backplane several times. In general, the architecture has not been specialized for fast processing between the network and disk interfaces. We will examine some approaches that address this limitation in Section 5.

4.3. Mass Storage System Reference Model

Supercomputer users have long had to deal with the problem that high performance machines do not come with scalable I/O systems. As a result, each of the major supercomputer centers has been forced to develop its own mass storage system, a network-based storage organization in which files are staged from the back-end storage server, usually from a near-line subsystem, to the front-end supercomputer.

The Mass Storage System (MSS) Reference Model was developed by the managers of these supercomputer centers, to promote more interoperability among mass storage systems and influence vendors to build such systems to a "standard" [Miller 88]. The purpose of the reference model is to provide a framework within which standard interfaces can be defined. They begin with the underlying premise that the storage system will be distributed over heterogeneous machines potentially running different operating systems. The model firmly endorses the client/server model of computation.

The MSS Reference Model defines six elements of the mass storage system: *Name Server*, *Bitfile Client*, *Bitfile Server*, *Storage Server*, *Physical Volume Repository*, and *Bitfile Mover* (see Figure 4.5). Bit files are the model's terminology for uninterpreted bit data streams. There are different ways to assign these elements to underlying hardware. For example, the Name Server and Bitfile Server may run a single Mass Storage control processor, or they may run on independent communicating machines.

An application's request for I/O service begins with a conversation with the Name Server. The name service maps a user readable file name into an internally recognized and unique bitfile ID. The client's requests for data are now sent to the Bitfile Server, identifying the desired files through their IDs. The Bitfile Server maps these into requests to the Storage Server, handling the logical aspects of file storage and retrieval, such as directories and descriptor tables. The Storage Server handles the physical aspects of file storage, and manages the physical data volumes. It may request the Physical Data Repository to mount volumes if they are currently off-line. Storage servers may be specialized for the kinds of volumes they need to manage. For example, one storage server may be specialized for tape handling while another manages disk. The Bitfile Mover is responsible for moving data between the Storage Server and the client, usually over a network. It provides the components and protocols for high-speed data transfer.

The MSS Reference Model has been incorporated into at least one commercial product: the Unitree File Management System sold by General Atomics, Inc. This is a UNIX-based hierarchi-

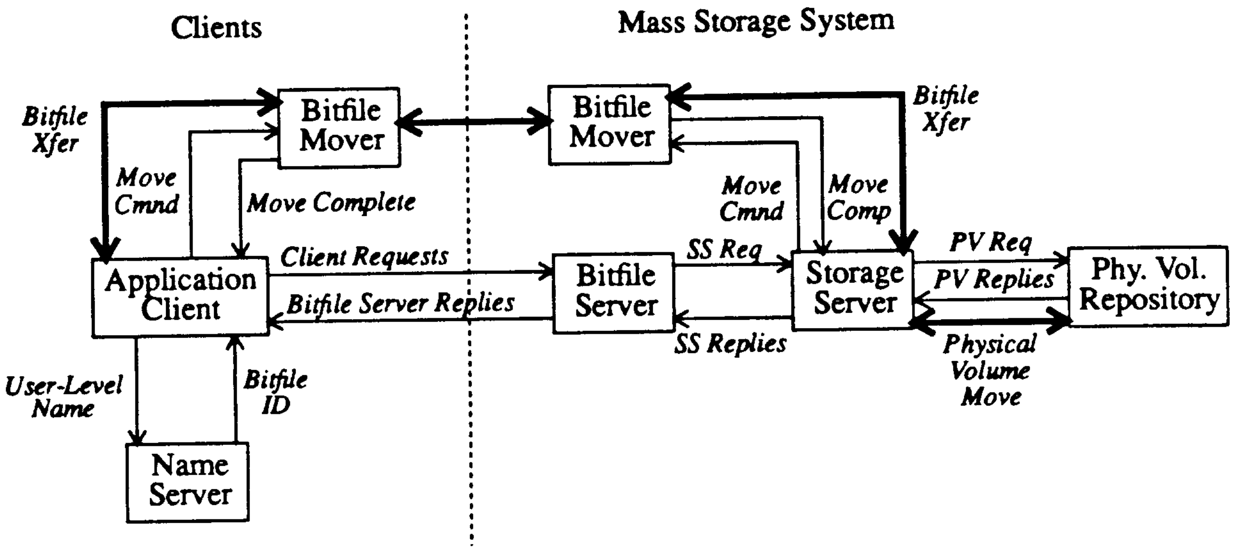


Figure 4.5: *Elements of the Mass Storage System Reference Model*
The figure shows the interactions among the elements of the MSS Reference Model. Command flows are shown in light lines while data flows are in heavy lines. The reference model clearly distinguishes among the software functions of name service, mapping of logical files onto physical devices, management of the physical media, and the transfer of files between the storage system and clients. The figure has been adapted from [Miller 88].

cal storage management system, based on software originally developed at the Lawrence Livermore National Laboratory.

5. Case Studies

In this section, we look at a variety of commercial architectures and research prototypes for high performance networks, file servers, and storage servers. Within these systems, we will see a common concern for providing high bandwidth between network interfaces and I/O device controllers.

5.1. Ultranet

5.1.1. General Organization

The UltraNetwork is a hub-based multihop network capable of achieving up to 1 Gbit/second transmission rates. Its most frequent application is as a local area network for interconnecting workstations, storage servers, and supercomputers.

Figure 5.1 depicts a typical Ultranet configuration. The hubs provide the basis of the high speed interconnect, by providing special hardware and software for routing incoming network packets to output connections. Hubs are physically connected by *serial links*, which consist of two unidirectional connections, one for each direction. If optical fiber is chosen for the links, data can be transmitted at rates of up to 250 Mbit/second and distances to 4 Km. The Gbit transmission rate is achieve by interleaving transmissions across four serial links. The point-to-point links are terminated by *link adapters* within the hubs, special hardware that routes the transmissions among input and output serial links. These are described in more detail below.

Computer are connected to the network in two different ways: through *host adapters* and *hub-resident adapters*. A host-based adapter is similar to the network controller described in Fig-

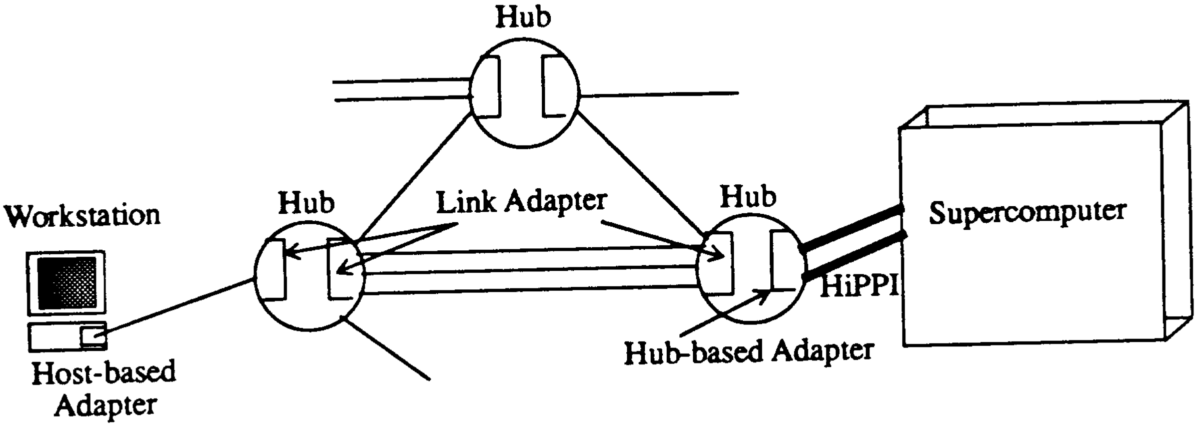


Figure 5.1: *UltraNet Configuration*
The network interconnection topology is formed by hubs connected by optical serial links. The maximum link speed is 250 Mb/s; higher transmission bandwidth is obtained by interleaving across multiple links. Host-based adapters plug into computer backplanes while adapters for channels such as HiPPI reside within the hub.

ure 2.2, and resides within the host computer's backplane. This kind of interface is appropriate for machines with industry standard backplanes, such as workstations and mini-supercomputers. In these kinds of clients, processors and I/O controllers, including the network interface, are treated as equals with respect to memory access. The adapter contains an on-board microprocessor and can perform its own direct memory accesses, just like any other peripheral controller.

A different approach is needed for mainframe and supercomputers, since these classes of machines connect to peripherals through special channel interfaces rather than standard backplanes. I/O devices are not peers, but are treated as slaves by the processor. The hub-resident adapters place the network interface to the Ultranet within the hub itself. These provide a standard channel interface to the computer, such as HiPPI or the IBM Block Multiplexer interface.

5.1.2. UltraNet Hub Organization

The heart of the Ultranet hub is a 64-bit wide (plus 8 parity bits), high bandwidth backplane called the *UltraBus*. Its maximum bandwidth is 125 MBytes/second. The serial links from other hubs and host-based adapters are interfaced to the UltraBus through *link multiplexers*, which in turn, are controlled by the link adapters. The link adapters route the serial data to the parallel interface of the UltraBus. Physically, it is a bus, but logically, the interconnect is treated more like a local area network. Packets are written to the bus by the source link adapter and are intercepted by the destination link adapter. If the output link is controlled by the same link adapter as the input link, the transfer can be accomplished without access to the UltraBus. Figure 5.2 illustrates the internal organization of the hub.

The link adapter contains a *protocol processor* and two modules that interface to the link multiplexers on the one hand and the UltraBus on the other. The protocol processor is responsible for handling the network traffic. The datapath that couples the personality modules on either side of the protocol processor consists of two unidirectional 64-bit wide busses with speed matching FIFOs at the interface boundaries. The busses operate independently and achieve peak transfers of 100 MByte/second.

The protocol processor consists of three components: the *Data Acknowledgment and Command Block Processor* (DACP), the *Control Processor* (CP), and the *Transfer Engine* (TE). The DACP performs fast processing of protocol headers and request blocks. The CP is responsible for managing the network, such as setting up and deleting connections between network nodes. The

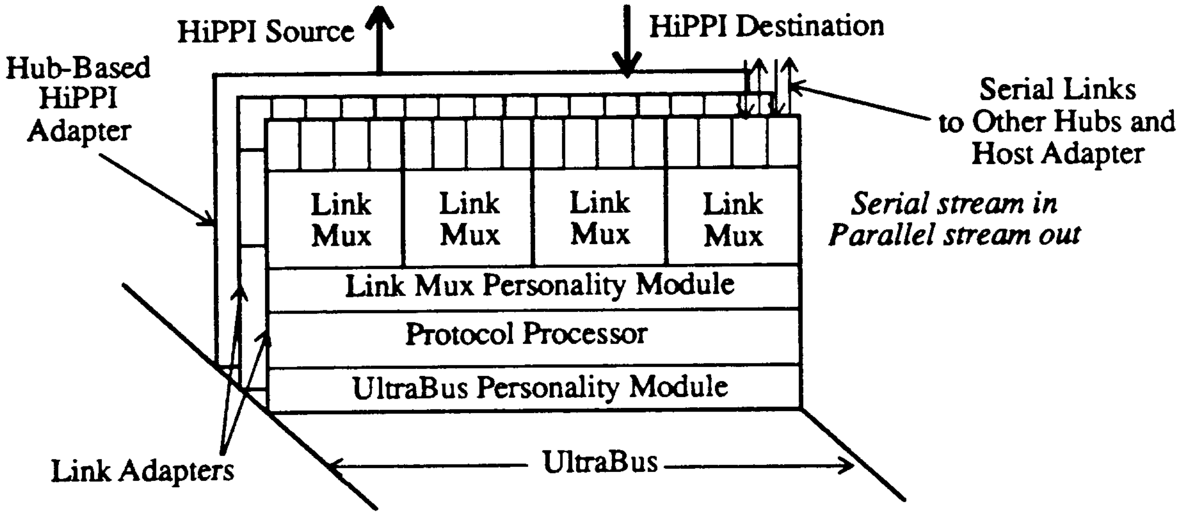

HiPPI Source
HiPPI Destination
Hub-Based HiPPI Adapter
Serial Links to Other Hubs and Host Adapter
Link Mux
Link Mux
Link Mux
Link Mux
Serial stream in Parallel stream out
Link Mux Personality Module
Protocol Processor
UltraBus Personality Module
Link Adapters
UltraBus

Figure 5.2: *Internal Organization of the UltraNet Hub*
The serial links, one for each direction, connect to the link multiplexers. Each link mux can handle up to four serial link pairs. The link adapter interfaces the link multiplexers to the wide, fast UltraNet bus. The Link adapter has enough intelligence to do its own routing of network traffic among the link multiplexers it manages. With a well configured hub, little traffic should need access to the UltraBus.

TE rapidly moves data through the protocol processor.

Figure 5.3 depicts the protocol architecture supported by the UltraNet. The combination of the UltraNet firmware and software implements the industry standard TCP/IP protocols on top of the UltraNet, as well as UltraNet specific protocols. The lower levels of the network protocol, namely the transport, network, data link, and physical link, are implemented with the assistance of the UltraNet protocol processor and host or hub-resident adapter hardware.

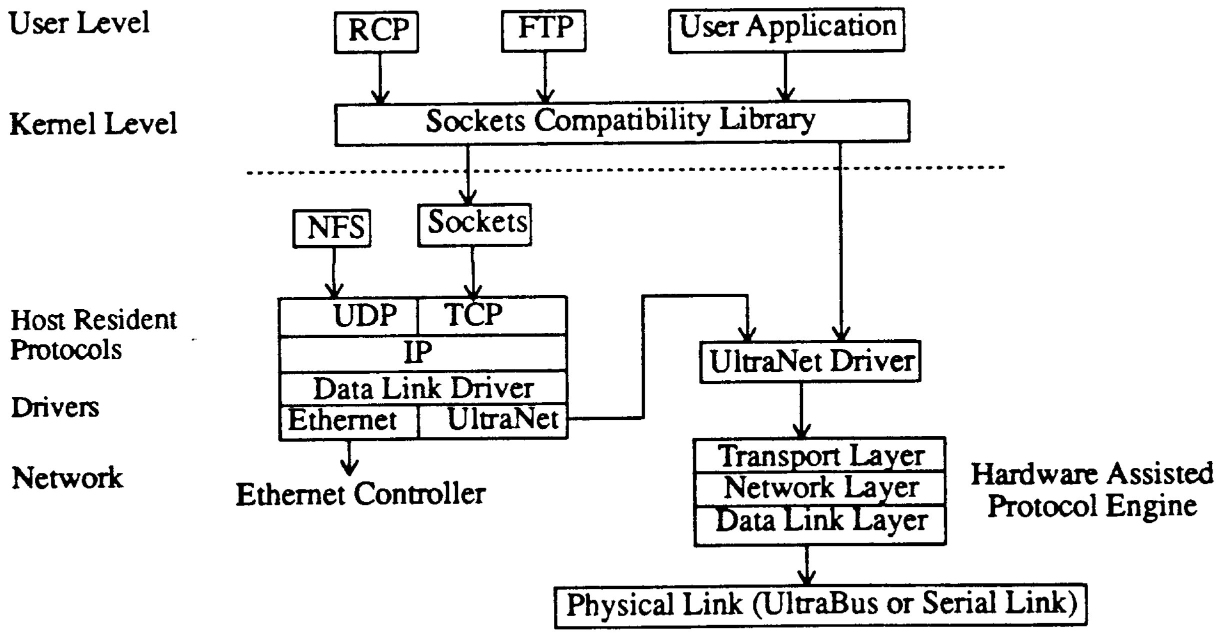


Figure 5.3: *UltraNet Protocol Architecture*
Access to the UltraNet is through the standard UNIX Socket interface. It is possible to use standard TCP/IP protocols on top of the UltraNet or UltraNet-specific protocols. The lower levels of the network are implemented with the assistance of the protocol engines and adapters throughout the UltraNet system.

5.2. Digital Equipment Corporation's VAXCluster and HSC-70

5.2.1. VaxCluster Concept

Digital Equipment Corporation's VAXCluster concept represents one approach for providing networked storage service to client computers {Kronenberg 86, Kronenberg 87]. The VAXCluster is a collection of hardware and software services that closely couple together VAX computers and Hierarchical Storage Controllers (HSCs). A VAXCluster lies somewhere between a "long distance" peripheral bus and a communications network: a high speed physical link couples together the processors, but message-oriented protocols are used to request and receive services. The VAXCluster concept is characterized by (1) a complete communications architecture, (2) a message-oriented computer interconnect, (3) hardware support for the connection to the interconnect, and (4) message-oriented storage controllers.

The hardware organization of a VAXCluster is shown in Figure 5.4. Its elements include VAX processors, HSC storage controllers, and the Computer Interconnect (CI). The latter is a high speed interconnect (dual path connections, 70 Mbits/second each), similar in operation to an Ethernet, although the detailed methods for media access are somewhat different. Physically, the CI is organized as a star network, but appears to processors as though it were a simple broadcast bus like the Ethernet. Up to sixteen nodes can be interconnected by a single star coupler, with each link being no more than 45 meters in length. A processor is connected to the CI via a CI port, a collection of hardware and software that provides the physical connection to the CI on one side and a high level queue-based interface to client software on the other side.

The communications protocols layered onto the CI and CI ports support three methods of transmission: *datagrams*, *messages*, and *blocks*. Datagrams are short transmissions meant to be used for status and information requests, and are not guaranteed to be delivered. Messages are similar to datagrams except that delivery is guaranteed. Read/write requests and other device control

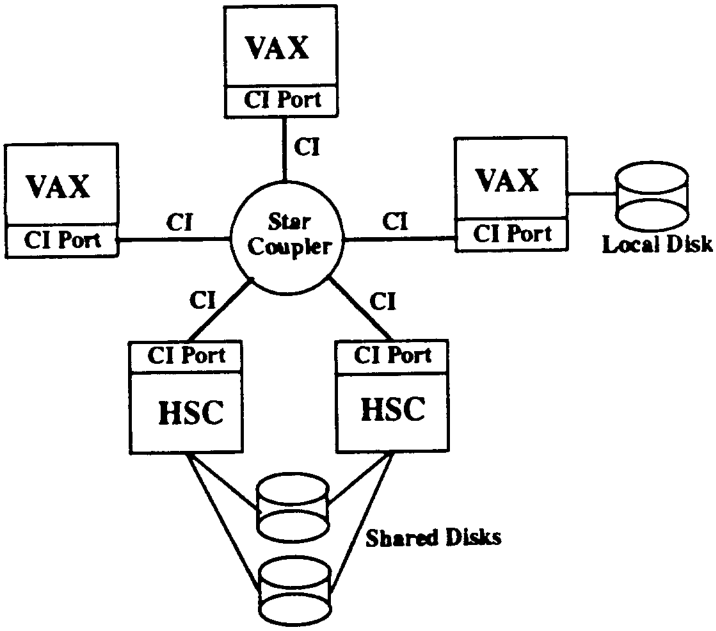


Figure 5.4: *VAXCluster Block Diagram*
A VAXCluster consists of client processors (VAX), server storage controllers (HSC), a high speed interconnect (CI), adapters (CI port), and coupling hardware (Star Coupler). A message-oriented protocol is layered onto the interconnect hardware to implement client/server access to storage services.

transmissions to storage controllers are handled via messages. The hardware in the CI ports provide special support for block transfers: an ability to copy sequential large blocks of data from the virtual address space of a process on one processor to the virtual address space of another process on another CI node. Block transfers are exploited to move data back and forth between client nodes and the storage controllers.

An interesting aspect of the VAXCluster architecture is its support for a Mass Storage Control Protocol (MSCP), through which clients request storage services from storage controllers attached to the CI. A message-based approach has several advantages in the distributed environment embodied by the cluster concept. First, data sharing is simplified, since storage controllers can extract requests from message queues and service them in any order they choose. Second, the protocols enforce a high degree of device independence, thus making it easier to incorporate new devices into the storage system without a substantial rewrite of existing software. Finally, the decoupling of a request from its servicing allows the storage controllers to apply sophisticated methods for optimizing I/O performance, including rearranging requests, breaking large requests into fragments that can be processed independently, and so on.

5.2.2. HSC-70 Internal Organization

The internal organization of an HSC is shown in Figure 5.5. The HSC was originally designed in the late 1970's, and has been in service for a decade. Its internal architecture was determined by the technology limits of that time. Nevertheless, there are a number of notable aspects about its organization. An HSC is actually a heterogeneous multiprocessor, with individual processors dedicated to specific functions. The three major subsystems are: (1) the host interface, (2) the I/O control processor, and (3) the I/O device controllers. They communicate via shared control and data memories accessed via a control and data bus respectively.

The Host Interface, called a K.CI, is responsible for managing the transfer of messages over the CI Bus. The hardware is based on an AMD bit-slice processor. The device controllers, called K.SDIs for disk interfaces and K.STIs for tape interfaces, use the same bit-slice processor. They implement device-specific read and write operations, as well as format, status, and seek operations

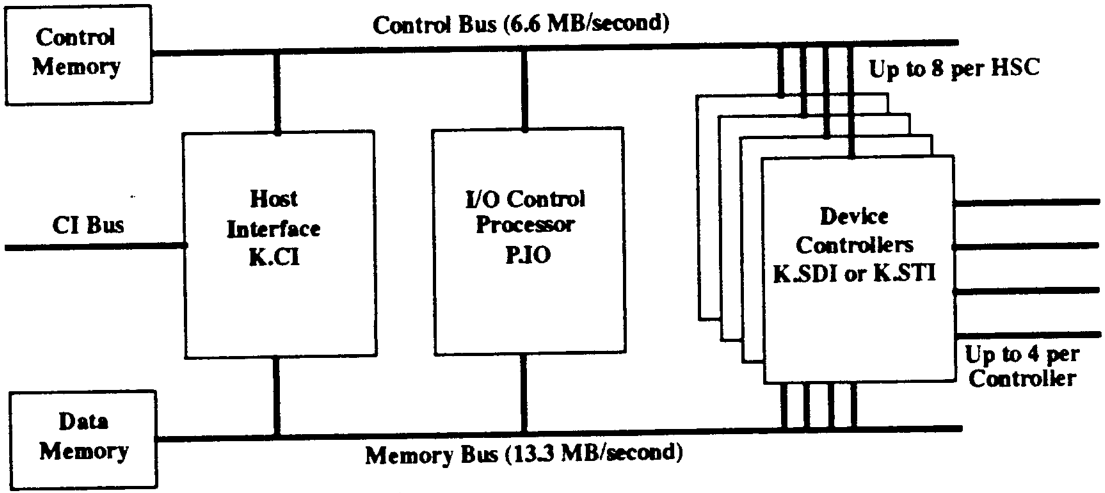

Control Memory
Control Bus (6.6 MB/second)
Up to 8 per HSC
CI Bus
Host Interface K.CI
I/O Control Processor P.IO
Device Controllers K.SDI or K.STI
Up to 4 per Controller
Data Memory
Memory Bus (13.3 MB/second)

Figure 5.5: *HSC Internal Architecture*
The Host Interface is managed by a dedicated bit slice processor called the K.CI. Devices are attached to K.SDI (disk) and K.STI (tape) device controllers. High level control is performed by the P.IO, a PDP-11 microprocessor programmed to coordinate the activities of the device controllers and the host interface.

(for disk) over Digital's proprietary device interfaces. Up to four devices can be controlled by a single K.X device controller, and up to eight K.X controllers can be attached to an HSC, for a total of twenty-four devices. All policy decisions are handled by the I/O control processor, which is based on a microprocessor implementation of the PDP-11 [Lary 89].

The shared memory subsystem of the HSC plays a critical role in its ability to sustain I/O traffic. Private memories deliver instructions and data to the various processors, keeping them off the shared memory busses. Data structures used for interprocessor communications are located in the control memory. The control memory and bus support interlocked operation, making it possible to implement an atomic two-cycle read-modify-write. Data moving between I/O devices and the Computer Interconnect must be staged through the data memory. The sizes of both memories are rather modest by today's standards: 256K bytes in each.

The performance bottlenecks within the HSC come from two primary sources: *bus contention* and *processor contention* [Bates 89]. We examine bus contention first. Internal bus contention affects the maximum data rate that the controller can support. The controller's transfer bandwidth (MBytes/second) is limited by its memory architecture and the implementation of the CI interface, both on the controller and on a processor with which it communicates. Because data must traverse the memory bus twice, the effective internal bandwidth to I/O devices is limited to 6.6 MBytes per second. For example, on a device read, data must be staged from the device controller to the data memory over the memory bus, and then transferred once again over the bus to the CI interface. The HSC's software includes mechanisms for accounting for the amount of internal bandwidth that has been allocated to outstanding I/O requests. It will throttle I/O activity by delaying some requests if it detects saturation.

While this may appear to be a limitation, a more serious restriction is imposed by the HSC's CI interface itself. In general, it is designed to sustain on the order of 2 MBytes/second. For some low-end members of the VAX family, even this may exceed the bandwidth of the host's CI interface. To avoid overrunning a host, and thus limit CI bandwidth wasted on retransmissions, the CI interface will only transmit a single buffer to a given client as long as there are buffers waiting to be sent to it.

Next we turn to processor contention. This is due to some extent to the design of the SDI disk interfaces. Each disk has a dedicated control bus, but a single data bus is shared among the devices attached to a controller. Thus, high data bandwidth can be sustained by spreading disks among as many controllers as possible. For example, two disks on a single K.SDI will transfer fewer bytes per second than a configuration with one disk on each of two disk controllers.

5.2.3. Typical I/O Operation Sequencing

To understand the flow of data and processing through the HSC, we shall examine the processing steps of a typical disk read operation. The steps that we outline next are described at a high-level. Considerably more detail, including the detailed data structures used, can be found in [Lary 89].

The first step is the arrival of the MSCP command over the CI bus. The message is placed in a K.CI reception buffer, where it is checked for well-formedness and validity. If it passes these checks, it is copied to a special data structure in the control memory, and pointers to this data structure are placed on a queue of work for the I/O policy processor.

The next step involves the execution of MSCP server software on the policy processor. The software is structured as a process that wakes up whenever there are pending requests in the work queue. The software examines the queue of commands, choosing the next one to execute based on the currently executing commands. It constructs a data structure that maps the MSCP command into physical disk operations, such as the disk seek command and a sequence of sector transfer requests. The original MSCP command message is modified to become a response message. The last phase is to place a pointer to the disk request data structure on a K.SDI work queue, to be found in the control memory.

The next thing that happens is the disk portion of the transfer. The K.SDI firmware reads the request on its work queue, extracts the seek command, and issues it to the appropriate drive. When the drive is ready to transfer, it indicates its status to the K.SDI. At this point the disk controller allocates buffers in the data memory, and stages the data as it comes in from disk to these buffers. When the list of sector transfers is complete, a completion message is placed on the work queue for the K.CI.

We are now ready to transfer the data from the controller's data memory to the host. The K.CI software wakes up when new work appears in its queue. It then generates the necessary CI message packets to transfer data from data memory out over the CI to the originally requesting host processor. As a data buffer is emptied, it is returned to a list of free buffers maintained in the control memory. When the last buffer of the transmission has been sent, the K.CI now transmits the MSCP completion message that was built by the I/O policy processor from the original request message.

The steps outlined above have assumed that processing continues without error. There are a number of error recovery routines that may be invoked at various points in the process described above. For example, if the transfer request within a K.SDI fails, the software is structured to route the request to error handling software to make the decision whether to retry or abort the request.

5.3. Seagate ARRAYMASTER (9058 Controller)

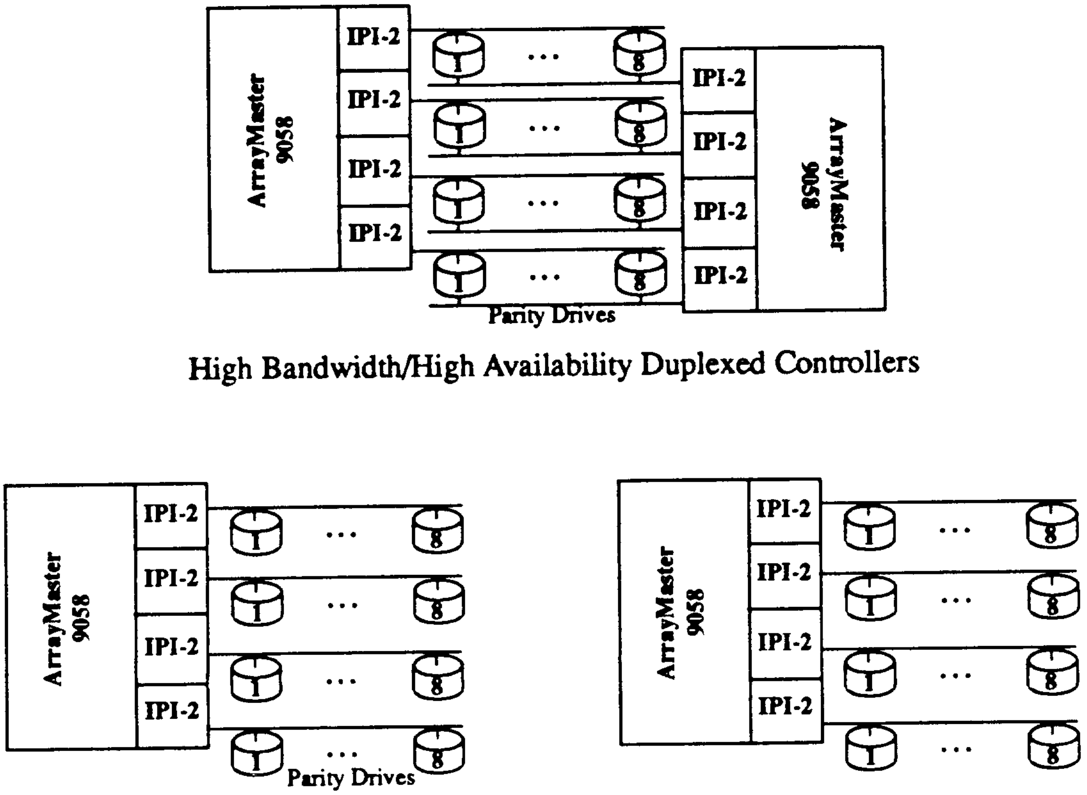
5.3.1. General Organization

The Seagate ARRAYMASTER is an example of an I/O controller design targeted for high bandwidth environments. To this end, it supports multiple (4) IPI-2 interfaces to disks, which can burst at 10 MBytes/second each, and multiple (2-4) IPI-3 interfaces to the host, each of which can burst transfer at 25 MBytes/second. (The IPI interface is similar in concept to the SCSI protocols described in Section 2.3, but provide higher performance though they are more expensive to implement.) A single controller can handle up to 32 disk drives, organized into eight stripe units (called *drive clusters* by Seagate) of four disks each.

The controller supports three alternative disk organizations: a high transfer rate/high availability mode with duplexed controllers, a simplex version of this organization, and a high transaction rate/high capacity organization. These are summarized in Figure 5.6. The first organization, for high transfer rate and very high system availability is distinguished by duplexed controllers, dual ported and spindle synchronized disk drives, and a 3 + 1 RAID Level 3 parity scheme.

Enhanced system availability is achieved by the duplexed controllers and dual ported drives: if a single controller fails, then a path still exists from the host to a device through a functional controller. Seagate claims 99.999% availability with a Mean Time to Data Loss (MTTDL) that exceeds 1 million hours with this configuration. This is probably a conservative estimate. A lost drive can be reconstructed within four minutes, assuming 1 GByte Seagate Sabre disk drives. The organization can sustain 36 MByte/seconds, assuming a sustained transfer bandwidth of 6 MByte/second per drive.

The second organization is characterized by single ported drives, organized into the RAID Level 3 scheme, and a single controller. System availability is not as good as the previous organization: a failure within the controller renders the disk subsystem unavailable. However, media



High Bandwidth/High Availability Duplexed Controllers

High Bandwidth/High Availability
Simplex Controller

High I/O Rate/High Capacity
Simplex Controller

Figure 5.6: *Alternative Disk Organizations for the ArrayMaster 9058*
The ArrayMaster can be configured in three alternative organizations: duplexed controller parity array for extremely high system availability, simplex parity array for high media availability, and a non-redundant organization for maximum I/O rate and capacity.

availability is just as good because of the parity encoding scheme. This organization can sustain 18 MBytes/second and also claims a 1 million hour MTTDL.

The last organization represents a tradeoff between performance, availability, and capacity. It gains capacity by dispensing with the parity drives, supporting a maximum of 32 GBytes versus 24 GBytes in the other two organizations, assuming 1 GByte drives. However, there is also no protection against data loss in the case of a disk crash. Data is no longer interleaved, thus sacrificing data bandwidth for a higher I/O rate. In the previous organizations, up to 8 I/Os can be in progress at the same time, one for each drive cluster. In this organization, 32 I/Os can simultaneously be in progress. The controller supports 500 random I/Os per second, approximately 16 I/Os per second per disk drive (this represents a disk utilization of 50%).

The controller designer's have placed considerable emphasis on providing support for very high data integrity within the controller and disk system. All internal data paths are protected by parity, data is written to disk with an enhanced ECC coding scheme (a 96 bit Reed-Solomon code that can correct up to 17 bit errors and even some 32 bit errors), and a large number of retries are attempted in the event of an I/O failure (three attempts at normal offset, all with ECC; three attempts at late and early data strobes with nominal carriage offset, all with ECC; three attempts at +/- carriage offset with nominal data strobes, all with ECC).

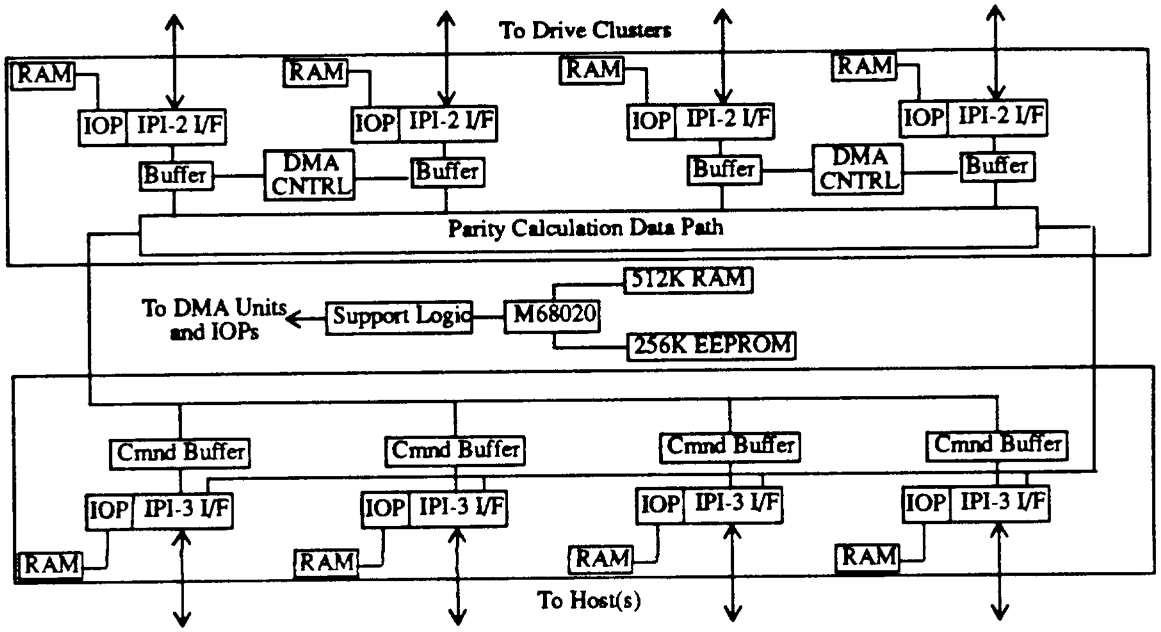


Figure 5.7: *Internal Organization of ArrayMaster 9058 Array Controller*
The ArrayMaster's internal structure consists of the disk interfaces, host interfaces, parity calculation logic, and a "traffic cop" microprocessor to determine the I/O strategy. I/O processors associated with each of the interfaces handle the low level details of the interface protocols. Data movement is controlled by direct memory access engines associated with the disk interfaces.

5.3.2. Controller Internal Organization

Figure 5.7 shows the internal organization of the ArrayMaster controller. An I/O request can be traced as follows. The host issues the appropriate command to one of the IPI-3 interfaces. This is staged to a command buffer within the controller. The central control microprocessor examines the command and determines how to implement it in detail.

Suppose that the command is a data write and that the array is organized into a RAID Level 3 scheme. The control processor maps this logical write request into a stream of physical writes to the disks within a drive cluster. As the data streams across the host interface, it passes through the parity calculation datapath, where the horizontal parity is computed. DMA controllers move data and parity from this datapath to buffers associated with individual disk interfaces. I/O processors local to the IPI-2 disk interfaces manage the details of staging data from the buffers to particular disk drives. Read operations are performed in much the same manner, but in reverse.

Note that reconstruction operations can be performed without host intervention. Assume that the failed disk has been replaced by a new one. Under the control of the central microprocessor, data is read from the surviving members of the drive cluster. The data is streamed through the parity calculation datapath, with the result being directed to the disk interface associated with the failed disk. The reconstituted data is then written to its replacement.

5.4. Maximum Strategies HiPPI-2 Array Controller

Maximum Strategies offers a family of storage products oriented towards scientific visualization and data storage applications for high performance computing environments. The products offer a tradeoff between performance and capacity, spanning from high MBytes/seconds but low MBytes (based on parallel transfer disks) to high performance/high capacity (based on arrays of disk arrays). In the following discussion, we concentrate on their HiPPI-based storage server.

Figure 5.8 shows the basic configuration of the Strategy-2 Array Controller. It supports one

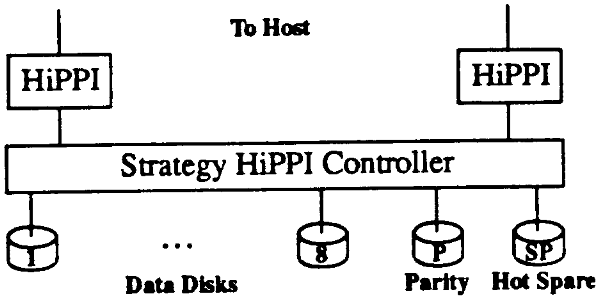


Figure 5.8: *Strategy HiPPI Controller Block Diagram*
The Strategy controller couples multiple HiPPI interfaces to an 8 + 1 + 1 RAID Level 3 disk organization.

or two 100 MByte/second host HiPPI interfaces or a single 200 MByte/second interface.The controller supports a RAID Level 3 organization calculated over eight data disks and one parity disk. Optionally, hot spares can be configured into the array. This allows reconstruction to take place immediately, without needing to wait for a replacement disk. It also helps the system achieve an even higher level of availability. Since reconstruction is fast, the system becomes unavailable only when two disks have crashed within a short period.

The controller can be configured in a number of different ways, representing alternative tradeoffs between performance and capacity. The low capacity/high performance configuration stripes its data across four parallel transfer disks. This yields 3.2 GBytes of capacity and can reach a 60 MByte/second data transfer rate. It provides no special support for high availability, such as RAID parity.

A second organization stripes across 8 + 1 parallel transfer disks implementing a RAID Level 3 organization. This organization provides 6.4 GBytes and achieves a 120 MByte/second transfer rate. Both configurations are called the Strategy HiPPI-SM Storage Server.

These organizations provide relatively little capacity for the level of performance provided. In addition, parallel transfer disks are quite expensive per MByte and have a poor reputation for reliability. An alternative configuration uses multiple ranks of 8 + 1 + 1 commodity disk drives. Maximum Strategies' HiPPI-S2 Storage Server is shown in Figure 5.9. Backend controllers (called S2s in Maximum Strategies' terminology) manage strings of eight disks each. Maximum Strategies makes use of older technology ESDI drives (5.25" formfactor, 1.2 GByte capacity each), which can share a common control path but require dedicated datapaths. A maximum configuration can support ten of these: eight data strings, one parity string, and an optional hot spare string. The backends are connected to the frontend HiPPI interfaces through a 250 MByte/second data backplane and a conventional VME backplane used for control. Note the separation of control and datapath. The high bandwidth data transfer path is over HiPPI; the control path uses a lower latency (and lower bandwidth) VME interconnect. Parity calculations are handled in the frontend. This organization can provide a 300 GByte capacity and a 144 MByte/second transfer rate.

Maximum Strategies also provides a storage server based on a VME-based host interface. The S2R Storage Server supports up to 40 x 5.25" ESDI drives, organized into an 8 + 1 + 1 RAID Level 3 scheme that is four stripe units deep. This organization yields 38.4 GBytes of capacity and an 18 MByte/second transfer rate.

The highest capacity/highest performance system combines the S2R-based arrays with the HiPPI-attached controller of Figure 5.8. The result is an "array of disk arrays." The architecture calls for replacing the S2 controllers with S2R disk controllers. Each S2R array contains 37 disks, organized into four stripe units of 8 data disks and 1 parity disk, plus one spare for the entire sub-

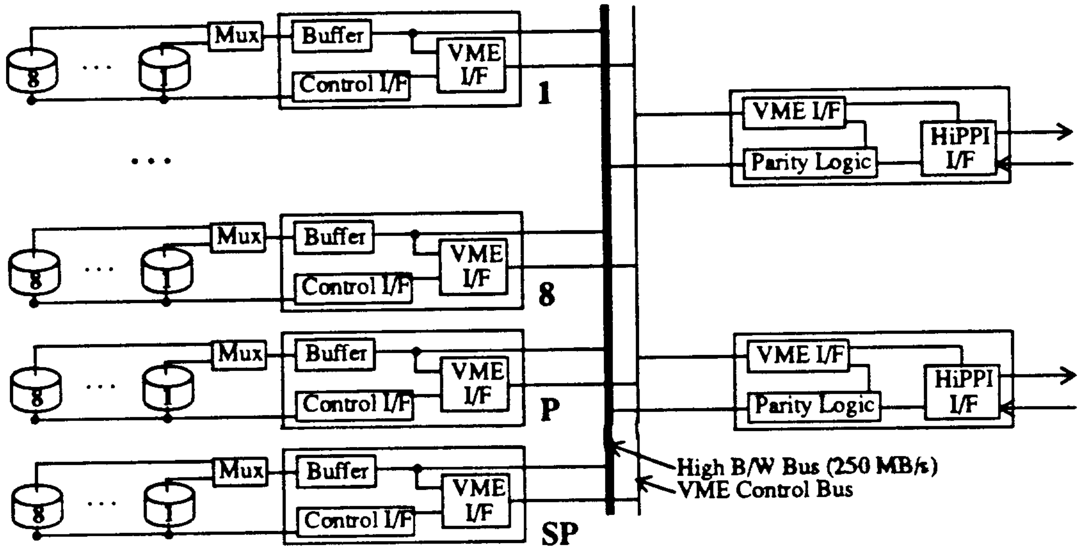


Figure 5.9: *High Capacity Strategy Array*
High capacity is achieved by using large numbers of commodity disk drives. These are coupled to the HiPPI frontends through a high bandwidth data bus and a VME-based control bus.

array. Up to 10 subarrays can be controlled by a single HiPPI controller, yielding a system configured from a total of 370 disk drives, a 345 GByte data capacity, and a 144 MByte/second transfer rate.

5.5. AUSPEX NS5000 File Server

5.5.1. General Overview

AUSPEX has developed a special hardware and software architecture specifically for providing very high performance NFS file service. The system provides a file system function integrated with an ability to bridge multiple local area networks. They claim to have achieved a performance level of 1000 NFS 8 KByte read I/O operations per second, compared with approximately 100 – 400 I/O operations per second for more conventional server architectures [Nelson 90].

They call their approach *functional multiprocessing*. Rather than building a server around a single processor that must simultaneously run the UNIX operating system and manage the network and disk interfaces, their architecture incorporates dedicated processors to separately manage these functions. By running specialized software within the network, file, and storage processors, much of the normal overhead associated with the operating system can be eliminated.

A functional block diagram of the NS5000 appears in Figure 5.10. The system backbone is an enhanced VME bus that has been tweaked to achieve a high aggregate bandwidth (55 MBytes/second). A conventional UNIX host processor (a SUN-3 or SUN Sparcstation board), the various special purpose processors, and up to 96 MBytes of semiconductor memory (the primary memory) can be installed into the backplane. We examine each of the special processors in the next subsection.

5.5.2. Dedicated Processors

A dedicated *network processor* board contains the hardware and software needed to manage two independent Ethernet interfaces. Up to four of these can be incorporated into the server to inte-

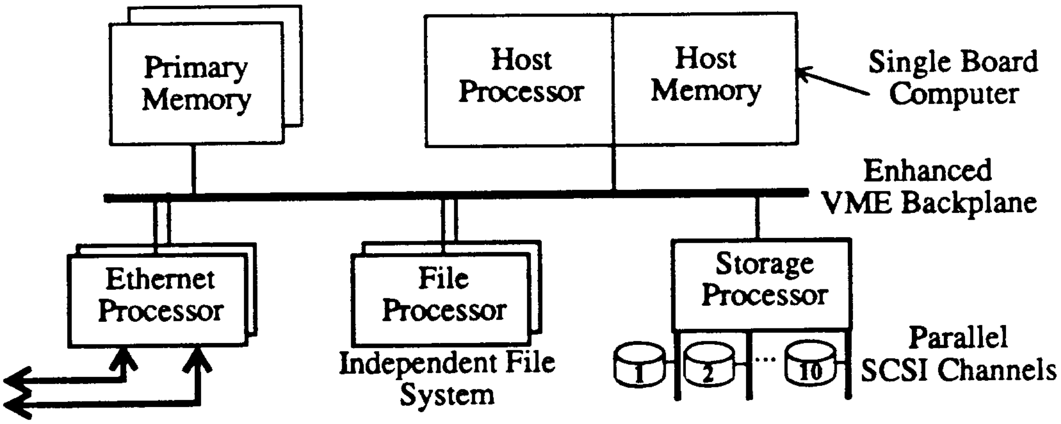


Figure 5.10: *NS5000 Block Diagram*
The server incorporates four different kinds of processors, dedicated to network, file, storage, and general purpose processing. The server can integrate up to eight independent Ethernets through the incorporation of multiple network processors. The storage processor supports ten SCSI channels, making it possible to attach up to twenty disks to the server.

grate a reasonably large number of independent networks. The board executes all of the necessary protocol processing to implement the NFS standard. Because the network boards implement their own packet routing functions, it is possible to pass packets from one network to another without intervention by the host. Some cached network packet headers are buffered in the primary memory.

The *file processor* board runs dedicated file system software factored out of the standard UNIX operating system. The board incorporates a large cache memory, partitioned between user data and file system meta-data, such as directories and inodes. This makes it possible for the file system code to access critical file system information without going to disks.

The *storage processor* manages ten SCSI channels. Disks are organized into four racks of five 5.25" disks each (20 disks per server). It is also possible to organize these into a RAID-style disk array, although the currently released software does not support the RAID organization at this time. Most of the primary memory is used as a very large disk cache. Because of the way the system is organized, most of the memory system and backplane bandwidth is dedicated to supporting data transfers between the network and disk interfaces.

The *host processor* is either a standard SUN-3 68020-based processor board or a Sparcstation host processor board. These run the standard Sun Microsystems' UNIX, as well as the utilities and diagnostics associated with the rest of the system.

5.5.3. Software Organization

A significant portion of Auspex's improved performance comes from the way in which the network and file processing software are layered onto the multiprocessor organization described above. The basic software architecture, its mapping onto the processors, and their interactions are shown in Figure 5.11.

Consider an NFS read operation. Initially, it arrives at an Ethernet processors, where the network details are handled. The actual data read request is forwarded to a file processor, where it is transformed into physical read requests, assuming that the request cannot be satisfied by cached data. The read request is passed to the storage processor, which turns it into the detailed operations to be executed by the disk drives. Retrieved data is transferred from the storage processor to primary memory, from which the Ethernet processor can construct data packets to be sent to the

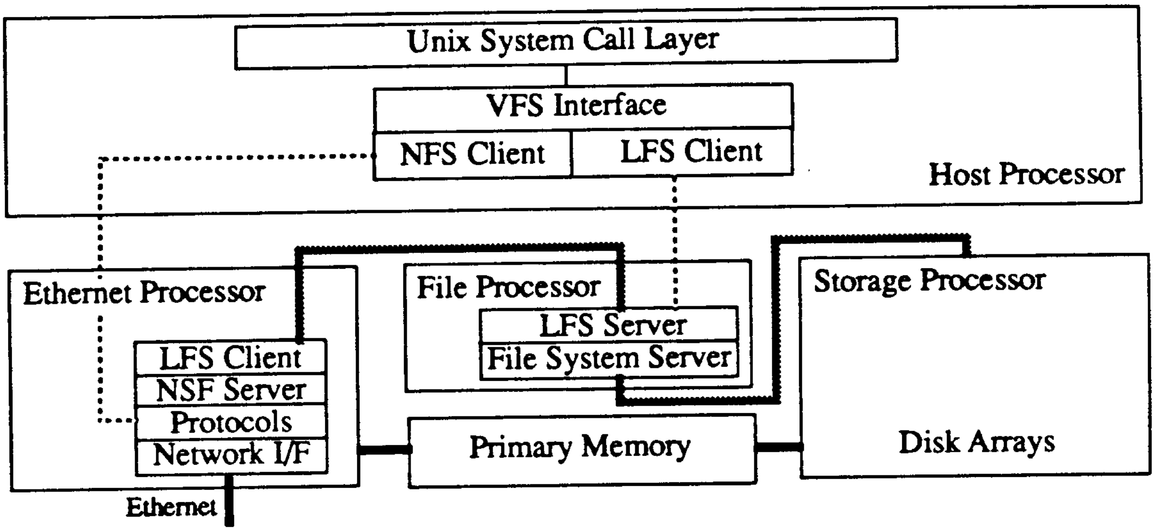


Figure 5.11: *Auspex NS5000 Software Architecture*
The main data flow is represented by the heavy black line, with data being transmitted from the disks to the primary memory to the network interface. The primary control flow is shown by a heavy gray line. File system requests are passed between LFS (Local File System) client software on the Ethernet processor to server software on the File Processor. These are mapped onto detailed requests to the Storage Processor by the File System Server. Limited control interactions involve the Virtual File System interfaces on the Host, and are denoted by dashed lines.

client. Note the minimal intervention from the host processor and software.

5.6. Berkeley RAID-II Disk Array File Server

Our research group at the University of California, Berkeley is implementing a high performance I/O controller architecture that connects a disk array to an UltraNet network via a HiPPI channel. We call it RAID-II to distinguish it from our first prototype, RAID-I, which was constructed from off-the-shelf controllers [Chervenak 90]. Given the observations about the critical performance bottlenecks in file server architectures throughout this paper, our controller has been specifically designed to provide considerable bandwidth between the network, disk, and memory interfaces.

A block diagram for the controller is shown in Figure 5.12. The controller makes use of a two board set from Thinking Machines Corporation (TMC) to provide the HiPPI channel interface to the UltraNet interfaces. The disk interfaces are provided by a VME-based multiple SCSI string board from Array Technologies Corp. (ATC).The major new element of the controller, designed by our group, is the *X-Bus board*, a crossbar that connects the HiPPI boards, multiple VME busses, and an interleaved, multiported semiconductor memory. The X-bus board provides the high bandwidth datapath between the network and the disks. The datapath is controlled by an external file server through a memory-mapped control register interface.

The X-bus board is organized as follows. The board implements an 8 by 8 32-bit wide crossbar bus. All crossbar transfers involve the on-board memory as either the source of the destination of the transfer. The ports are designed to burst transfer at 50 MByte/second, and sustain transfers of 40 MByte/second. The crossbar is designed to provide an aggregate bandwidth of 320 MByte/second.

The controller memory is allocated eight of the crossbar ports. Data is interleaved across the eight banks in 32 word interleave units. Although the crossbar is designed to move large blocks from memory to or from the network and disk interfaces, it is still possible to access a single word when necessary. For example, the external file server can access the on-board memory through

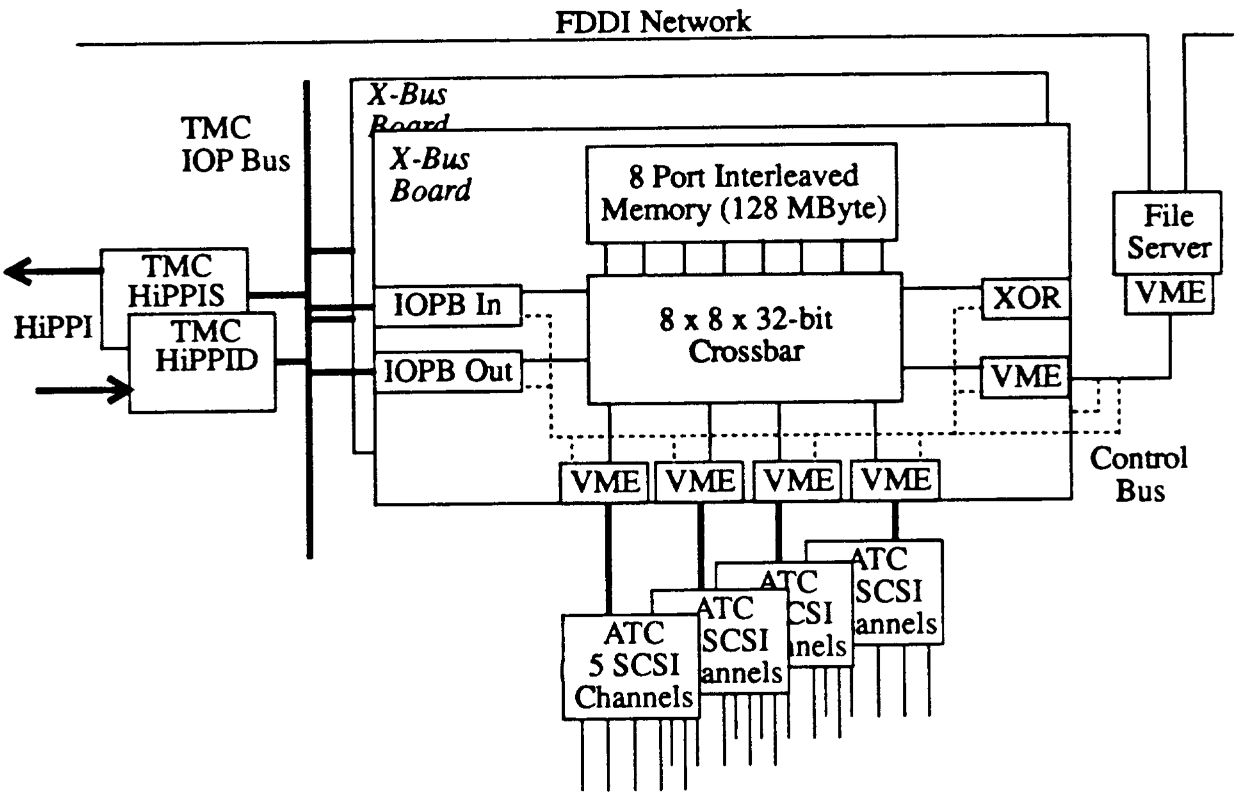


Figure 5.12: *RAID-II Organization*
A high bandwidth crossbar interconnection ties the network interface (HiPPI) to the disk controllers (Array Tech) via a multiported memory system. Hardware to perform the parity calculation is associated with the memory system.

the X-bus board's VME control interface.

Two of the remaining eight ports are dedicated as interfaces to the Thinking Machine I/O processor bus. The TMC HiPPI board set also interfaces to this bus. Since these X-bus ports are dedicated by their direction, the controller is limited to a sustained transfer rate to the network of 40 MByte/second.

Four more ports are used to couple to single board multi-string disk controllers via the industry standard VME bus, one disk controller per VME bus. Because of the physical packaging of the array, 15 disks can be attached to each of these, in three stripe units of five disks each. Thus, 60 disk drives can be connected to each X-bus board, and a two X-Bus board configuration consists of 120 disk drives.

Of the remaining two ports, one is dedicated for special hardware to compute the horizontal parity for the disk array. The last port links the X-Bus board 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 off of the controller, to access network headers and file meta-data in the controller cache.

It may seem strange that there is no processor within the X-Bus board. Actually, the configuration of Figure 5.12 contains no less than seven microprocessors: one in each of the HiPPI interface boards, one in each of the ATC boards, and one in the file server (we are also investigating multiprocessor file server organizations). The processors within the HiPPI boards are being used to handle some of the network processing normally performed within the server. The processors within the disk interfaces handle the low level details of managing the SCSI interfaces. The file server CPU must do most of the conventional file system processing. Since it is executing file server code, the file server needs access only to the file system meta-data, not user data. This makes its possible to locate the file server cache within the X-Bus board, close to the network and disk interfaces.

Since a single X-bus board is limited to 40 MByte/second, we are examining system organizations that interleave data transfers across multiple X-bus boards (as well as multiple file servers, each with its own HiPPI interface). Multiple X-bus boards can share a common HiPPI interface through the IOP Bus. Two X-bus boards should be able to sustain 80 MByte/second, more fully utilizing the available bandwidth of the HiPPI interface.

The controller architecture described in this subsection should perform well for large data transfers that require high bandwidth. But it will not do so well for small transfers where latency dominates performance more than transfer bandwidth.Thus we are investigating organizations in which the file server remains attached to a more conventional network, such as FDDI. Requests for small files will be serviced over the lowest latency network available to the server. Only very large files will be transferred through the X-Bus board and the UltraNet.

6. Summary and Research Directions

In this paper, we have made the case for generalizing the workstation-server storage architecture to the mainframe and high performance computing environment. The concept of network-based storage is very compelling. It has been said that the difference between a workstation and a mainframe is the I/O system. The distinction will become blurred in the new system architectures made possible by high bandwidth, low latency networks coupled to the correct use of caching and buffering throughout the path from service requestor to service provider.

Nevertheless, many research challenges remain before this vision of ubiquitous network-based storage can be achieved. First, new methods are needed to effectively manage the complete and complex storage hierarchy as described in this paper. How should data be staged from tertiary to secondary storage? What are the effective prefetching strategies? How is data to be extracted from such large storage systems?

Second, it is time to apply a system-level perspective to storage system design. Throughout the I/O path, from host to embedded disk controller, we find buffer memories and processing capabilities. The current partitioning of functions may not be correct for future high performance systems. For example, some searching and filtering capabilities could be migrated from applications into the devices. The memory in the I/O path could be better organized as caches rather than speed matching buffers, given enough local intelligence about I/O patterns. A better approach for error handling is also possible given a system perspective. For example, in response to a device read error, a disk array controller could choose between retrying the read or exploiting horizontal parity techniques to reconstitute the data on the fly.

Third, new architectures are needed to break the bottlenecks, both hardware and software, between the network, memory, and I/O interfaces. The RAID-II controller tackles this at the hardware level, by providing a high bandwidth interconnection among these components. At the software level, new methods need to be developed to reduce the amount of copying and memory remapping currently required for controlling these interfaces.

Fourth, today's high bandwidth networks, such as FDDI and UltraNet, exhibit latencies that are somewhat worse than conventional Ethernets. Unfortunately, latency becomes a dominating factor as the overheads of data transfer scales down in higher bandwidth networks. New methods need to be developed to reduce this latency. One strategy is to increase the packet sizes, to better amortize the start-up latencies. A second strategy, demonstrated by the Autonet project at Digital Equipment Corporation's System Research Laboratory, is to construct a high bandwidth network using point-to-point connections and an active switching network [Schroeder 90].

Finally, the whole issue of distributed and multiprocessor file/storage servers and their role in high performance storage systems must be addressed. The technical issues include the methods for how to partition the file server software functions among the processors of a multiprocessor or a distributed collection of processors. The AUSPEX controller architecture is one approach to the former. The IEEE Mass Storage System Reference offers one model for the latter.

Acknowledgments

We appreciate the careful reading of this manuscript and the detailed comments by Peter Chen, Ann Chervenak, Ed Lee, Ethan Miller, Srini Seshen, and Steve Strange. The research that led to this paper was supported by the Defense Advanced Research Projects Agency and the National Aeronautics and Space Administration under contract NAG2-591, "Diskless Supercomputers: High Performance I/O for the TeraOp Technology Base." Additional support was provided by the State of California MICRO Program in conjunction with industrial matching support provided by DEC, Emulex, Exabyte, IBM, NCR, and Storage Technology corporations.

7. References

Anon, *Network Operations Manual*, UltraNetwork Technologies, Part Number 06-0001-001, Revision A, (1990). Chapter 2: UltraNet Architecture; Chapter 3: Ultranet Hardware.

Anon, *Strategy HPPI Disk Array Subsystem Operation Manual*, Maximum Strategies, Part Number #HPP100, (1990).

Bates, K. H., "Performance Aspects of the HSC Controller," *Digital Technical Journal*, V. 8, (February 1989)), pp. 25 - 37.

Borrill, P., J. Theus, "An Advanced Communications Protocol for the Proposed IEEE 896 FutureBus," *IEEE Micro*, (August 1984), pp. 42 - 56.

Cerf, V., "Networks," *Scientific American*, V 265, N 3, (September 1991), pp. 72 - 85.

Chervenak, A., "Performance Measurements of the First RAID Prototype," U. C. Berkeley Computer Science Division Report No. UCB/CSD 90/574, (January 1990).

Chervenak, A., R. H. Katz, "Performance Measurements of a Disk Array Prototype," ACM SIGMETRICS Conference, San Diego, CA, (May 1990).

Clark, D., V. Jacobson, J. Romkey, H. Salwen, "An Analysis of TCP Processing Overhead," *IEEE Communications Magazine*, (June 1989), pp. 23 - 29.

Exabyte Corporation, "EXB-120 Cartridge Handling Subsystem Product Specification," Part No. 510300-002, 1990.

Feder, B. F., "The Best of Tapes and Disks," *N. Y. Times*, Sunday Business Section, (September 1, 1991), p. 9.

Heatly, S., D. Stokesberry, "Analysis of Transport Measurements Over a Local Area Network," *IEEE Communications Magazine*, (June 1989), pp. 16 - 22.

Hennessy, J., D. Patterson, *Computer Architecture: A Quantitative Approach*, Morgan Kaufmann, San Mateo, CA, 1990.

Hewlett-Packard Corporation, "HP Series 6300 Model 20GB/A Rewritable Optical Disk Library System Product Brief," 1989.

Joshi, S. P., "High Performance Networks: Focus on the Fiber Distributed Data Interface Standard," *IEEE Micro*, (June 1986), pp. 8 - 14.

Kanakia, H., D. Cheriton, "The VMP Network Adaptor Board (NAB): High-Performance Network Communication for Multiprocessors," Proc. ACM SigComm '88 Symposium, (August 1988), pp. 175 - 187.

Kanakia, H., "High Performance Host Interfacing for Packet-Switched Networks," Ph.D. Dissertation, Department of E.E.C.S., Stanford University, 1990.

Katz, R., G. Gibson, D. Patterson, "Disk System Architectures for High Performance Computing," *Proceedings of the IEEE*, Special Issue on Supercomputing, (December 1989).

Kodak Corporation, "Optical Disk System 6800 Product Description," 1990.

Kronenberg, N. P., H. Levy, W. D. Strecker, "VAXClusters: A Closely Coupled Distributed System," *ACM Transactions on Computer Systems*, V. 4, N. 2, (May 1986), pp. 130 - 146.

Kronenberg, N. P., H. M. Levy, W. D. Strecker, R. J. Merewood, "The VAXCluster Concept: An Overview of a Distributed System," *Digital Technical Journal*, V. 5, (September 1987), pp. 7 - 21.

Kryder, M. H., "Data Storage in 2000 — Trends in Data Storage Technologies," *IEEE Trans. on Magnetics*, V. 25, N. 6, (November 1989), pp. 4358 - 4363.

Lary, R. L., R. G. Bean, "The Hierarchical Storage Controller: A Tightly Coupled Multiprocessor as Storage Server," *Digital Technical Journal*, V. 8, (February 1989), pp. 8 - 24.

Maximum Strategies, "Strategy HPPI: Disk Array Subsystem," Report No. HPP100, (1990).

Massiglia, P., *Digital Large System Mass Storage Handbook*, Digital Equipment Corporation, 1986.

Miller, S. W., "A Reference Model for Mass Storage Systems," *Advances in Computers*, V 27, 1988, pp. 157 – 210.

Moran, J. R. Sandberg, D. Coleman, J. Kepecs, B. Lyon, "Breaking Thru the NFS Performance Bottleneck," EUUG Sorino 90, Munich, (April 1990).

Nelson, B., "An Overview of Functional Multiprocessing for NFS Network Servers," AUSPEX Technical Report 1, (July 1990).

Nelson, M., J. K. Ousterhout, B. Welch, "Caching in the Sprite Network File System,"' *A.C.M. Transactions on Computer Systems*, V. 6, N. 1, (February 1988), pp. 134-154.

Negraponte, N., "Products and Services for Computer Networks," *Scientific American*, V 265, N 3, (September 1991), pp. 106 - 115.

Pollard, A., "New Storage Function for Digital Audio Tape," New York Times, Wednesday, (May 25, 1988), p. C10.

Ranade, S., J. Ng, *Systems Integration for Write-Once Optical Storage*, Meckler, Westport, CT, 1990.

Rosenblum, M., J. Ousterhout, "The Design and Implementation of a Log-Structured File System," *ACM Transactions on Computer Systems*, (February 1992), to appear.

Orenstein, E., "HPPI-based Storage System," *Computer Technology Review*, (April 1990).

Ousterhout, J. K., "Why Aren't Operating Systems Getting Faster as Fast as Hardware?," Proc. USENIX Summer Conference, Anaheim, CA, (June 1990), pp. 247 - 256.

Sandberg, R., D. Goldberg, S. Keiman, D. Walsh, B. Lyon, "Design and Implementation of the SUN Network Filesystem," Proc. USENIX Summer Conference, (June 1985), pp. 119 - 130.

Schroeder, M., A. D. Birrell, M. Burrows, H. Murray, R. M. Needham, T. L. Rodeheffer, E. H. Satterthwaite, C. P. Thacker, "Autonet: A High-Speed Self-configuring Local Area Network Using Point-to-Point Links," DEC SRC Tech. Rep. #59, (April 1990).

Spencer, K., "The 60-Second Terabyte," *Canadian Research Magazine*, (June 1988).

Tan, E., B. Vermeulen, "Digital Audio Tape for Data Storage," *IEEE Spectrum*, V. 26, N. 10, (October 1989), pp. 34 - 38.

Tesla, L., "Networked Computing in the 1990s," *Scientific American*, V 265, N 3, (September 1991), pp. 86 - 93.

Verity, J. W., "Rethinking the Computer," *Business Week*, (November 26, 1990), pp. 116 - 124.

Watson, R., S. Mamrak, "Gaining Efficiency in Transport Services by Appropriate Design and Implementation Choices," *ACM Transactions on Computer Systems*, V. 5, N. 2, (May 1987), pp. 97 - 120.

Wood, R., "Magnetic Megabytes," *IEEE Spectrum*, V. 27, N. 5, (May 1990), pp. 32 - 38.