Network-Attached Storage Systems

Randy H. Katz

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

Abstract

In the traditional mainframe-centered view, storage devices are coupled to the system through complex I/O channels. With the dramatic shift towards distributed 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 describe a research prototype, developed at Berkeley, that takes a new approach to high performance computing based on network-attached storage.

1. Introduction

The traditional mainframe-centered model of computing can be characterized by small numbers of large-scale mainframes, 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, computation servers, and storage servers connected via high performance networks [Verity 90].

What makes this new paradigm possible is the rapid development and acceptance of the *client-server model* of computation. 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 back-ups. The resulting organization combines distributed processing power with a centralized view of storage.

Technology developments in processors, networks, and storage systems are affecting the relationship between clients and servers. It is well known that processor performance is increasing at an astonishing rate. 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 (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 [Anon 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. Unfortunately, the random I/O rate is improving only very slowly, due to

0-8186-2775-1/92 $3.00 © 1992 IEEE

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 [Katz 89]. 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. The rest of this paper is organized as follows. In the next section, we review network, channel, and backplane trends. Section 3 reviews storage controller and file server architectures. We discuss storage trends in Section 4. Section 5 describes a prototype high performance network-attached I/O controller being developed at Berkeley. Our summary, conclusions, and suggestions for further research are found in Section 6.

2. Interconnect

2.1. Networks, Channels, 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.

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.

2.2. Communications Networks and Controllers

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. Performance measurements of network transmission services all point out that the significant overhead is not protocol interpretation. 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].

The network controller is the collection of hardware and firmware that implements the interface between the network and the host processor. It 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 the physical and link level protocols of the network.

The interaction between the network controller and the host's memory is depicted in Figure 1. Lists of blocks containing packets to be sent and packets that have been received are maintained in the host processor's memory.

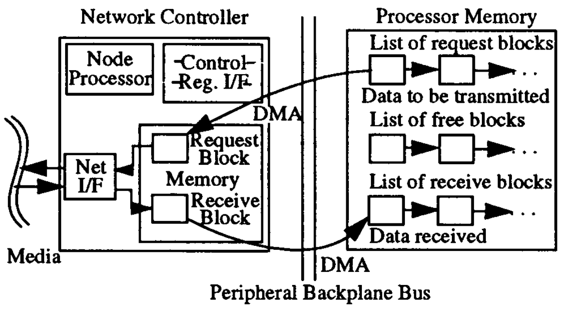

Figure 1: *Network Controller/Processor Memory Interaction*

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.

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.

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. SCSI treats peripheral devices in a largely device-independent fashion. For example, a disk drive is viewed as a linear byte stream.

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.

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

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 Controllers and File Servers

3.1. I/O Data Flow

Figure 2 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 (a *host bus adapter*, or HBA). 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

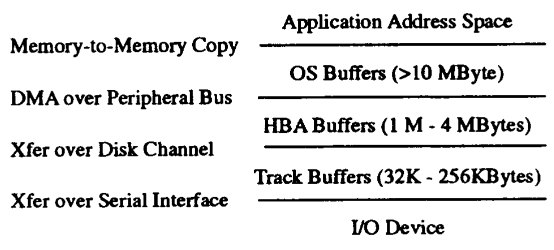
Application Address Space
Memory-to-Memory Copy
OS Buffers (>10 MByte)
DMA over Peripheral Bus
HBA Buffers (1 M - 4 MBytes)
Xfer over Disk Channel
Track Buffers (32K - 256KBytes)
Xfer over Serial Interface
I/O Device

Figure 2: *I/O Data Flow*

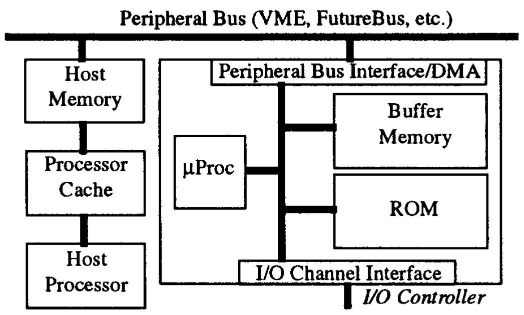

Figure 3: *Internal Organization of an I/O Controller*

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.

3.2. Internal Organization of I/O Controller

Figure 3 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 1. 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 a network controller. 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 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.

3.3. 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.

Figure 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.

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

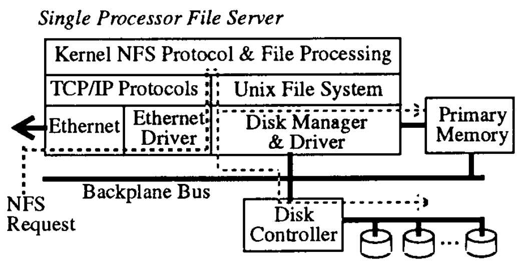

Figure 4: *Conventional File Server Architecture*

Explore Litigation Insights

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

Real-Time Litigation Alerts

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

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

Advanced Docket Research

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

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

Analytics At Your Fingertips

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

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

API

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

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

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

FINANCIAL INSTITUTIONS
Litigation and bankruptcy checks for companies and debtors.

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

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

fastcase® Smarter legal research.