N95-24123

Client/Server Data Serving for High Performance Computing

Chris Wood
Maximum Strategy Incorporated
801 Buckeye Court, Milpitas CA. 95035
chrisw@maxstrat.com
408-383-1600
408-383-1616 (fax)

Abstract

This paper will attempt to examine the industry requirements for shared network data storage and sustained high speed (10's to 100's to thousands of megabytes per second) network data serving via the NFS and FTP protocol suite. It will discuss the current structural and architectural impediments to achieving these sorts of data rates cost effectively today on many general purpose servers and will describe an architecture and resulting product family that addresses these problems.

The sustained performance levels that were achieved in the lab will be shown as well as a discussion of early customer experiences utilizing both the HIPPI-IP and ATM OC3-IP network interfaces.

Introduction:

Back in the dark ages, about the time that touch-tone telephones were coming into vogue, computers were simple things that read in some data and a program, processed the data and spit out the answer. Data storage typically consisted of small amounts of core memory, card reader/punch machines and maybe a tape drive. Disk storage added the dimension of random access to data, but was typically directly attached to the central processor by any number of proprietary I/O schemes[1].

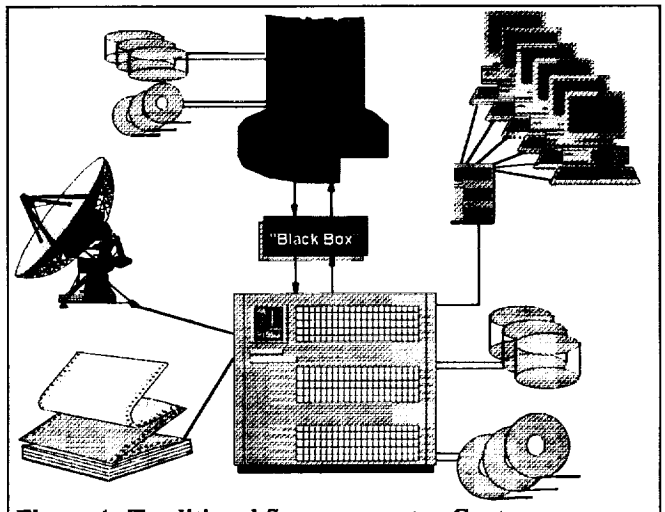

Figure 1: Traditional Supercomputer Center

Data sharing

Early customers in the high performance arena often owned several processors: one or two very large number-crunches and several smaller machines used to prepare the data for the large machine and/or print the output stream generated by the number crunchers[2]. A typical installation might have looked like that shown in Figure 1.

Sharing of data across computing platforms was typically done by copying data located on one processor to tape and reading it on another. Just attaching disk storage to multiple

processors did not address the problem since most processors utilized different physical and logical attachments. Even if two machines, by chance, could physically share a disk storage device, different machines wrote and read data in different ways and could not access each others data.

Some early solutions involved the use of black boxes that attempted to mate different interfaces and address data format incompatibilities. The large number of interfaces and file structures in use today (and growing!) tended to work against this type of an interconnect solution[3]. In an early, and very innovative, attempt to address the problem of incompatible file systems, the Los Alamos Lab's created the Common File System (CFS)[4] on an IBM mainframe base. Architecturally, CFS could be considered to be the first implementation of networked data serving. It addressed the issues of data sharing amongst heterogeneous hardware platforms, incompatible file systems (e.g. its name...Common File System) and the problem of incompatible physical I/O attachment schemes - although this was often accomplished via "black box" I/O mating hardware.

The ever growing complexity of the "black box" solution for heterogeneous platform interconnect ruled this method out as a long term answer to seamless data sharing. Alternatively, all vendors could adopt a common file system and I/O structure. This was thought unlikely.

Client/Server to the rescue?

A little over ten years ago, a group of UNIX architects at Sun Microsystems realized that the only way to address the data sharing problem (as well as data currency, consistency and access) was to remove the "ownership" of the data from the compute processor (the entity who processes the data) in a manner similar to that utilized by CFS and store it on an independent "server" who's only job is to store and retrieve that data when requested to do so by the compute processor (e.g. the "client"). Most importantly, they also defined a standard way to access the data that would be independent of any particular physical file system and physical interconnect. Thus was born the Network File System (NFS); the original foundation of client/server computing.

Speed limit: 5 MPH.

DEC talking to IBM, SGI communing with HP, The USSR making peace with the USA! All these things became true; unfortunately the Cold War lasted 50 years and that seems to be how long (in a relative sense) any self respecting high performance computer seems to have to wait for data from its "server" today. The convince of sharing data across many platforms comes at a price - speed. NFS (and essentially any exportable file system available today) is primarily hamstrung by three major bottlenecks:

1. The speed that the server's disk subsystem can deliver data to the NFS server, and
2. The speed that the NFS server can process this data through its own file system and encapsulate (packetize) this data with the UDP and IP network protocols and deliver it to the network fabric, and
3. The finite usable speed of the fabric[5].

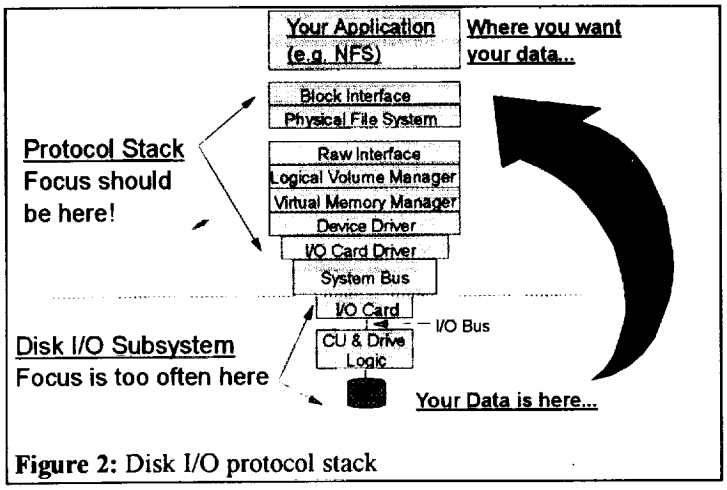

Figure 2: Disk I/O protocol stack

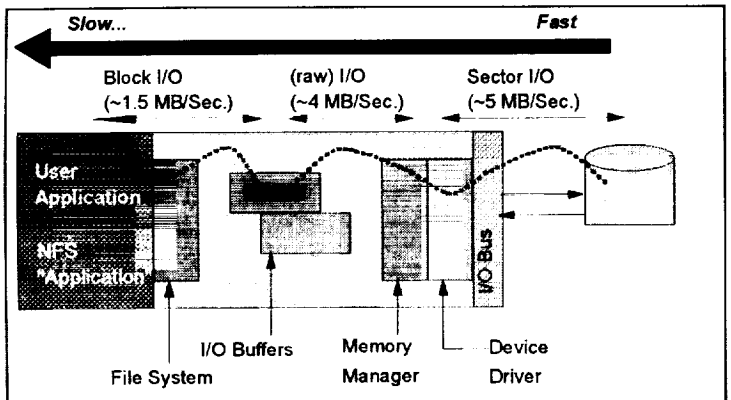

Figure 3: Effective data rate through a UNIX platform

Ethernet, at 10 Mbit/second focused everybody on item three because of its low usable bandwidth and, after some frustration, begat FDDI (and later HIPPI, Fibre Channel and ATM) which was supposed to be 10 times faster. To everybody's amazement, they did not get 10X the data rate, they got 2-3X the data on a good day and often less. Adding more FDDI rings, routers, bridges and other network paraphernalia did not seem to help. Just speeding up the fabric did not seem to be the answer. Items one and two were now the gate, but seemed to have received less attention by industry over the past several years than may have been deserved[6].

The real culprits exposed!

For an NFS server to deliver data to a client it first has to read the data off the disk subsystem in the server. In a typical UNIX environment the software protocol stack would look something like that shown in Figure two. If you measure actual disk data rates starting at the disk interface and working your way towards an application (e.g. NFS server or a user application) the effective delivered rate decreases with each step up the protocol stack. Figure three illustrates this point graphically[7]. Fast disk I/O becomes slow disk data by the time it reaches the requesting application. A typical SCSI disk of recent vintage can deliver about 5 megabytes a second of user data off the media. By the time the data has traversed the protocol stack labyrinth, the sustained delivery rate has often decreased to about 1.5 megabytes a second. Most storage subsystems[1] are not capable of delivering high sustained bandwidth to the requesting application; be it the NFS server or a users application.

Incrementalism: Disk striping, data caches and other software improvements.

Various incremental methods have been proposed and/or implemented in a limited sense to attempt to address this problem. The most common of these is disk striping wherein the disk device drivers and (usually) the layer of code that performs the function of the logical volume manager are modified to break up a users data record into smaller chunks and write (stripe) these chunks onto multiple disks simultaneously. Various RAID types may

[1] A "storage subsystem", as used here, represents the complete collection of components necessary to deliver data to the requesting application: Disk drives, I/O cards, software layers, file systems, etc.

also be imbedded in the software to increase data availability. Some high data rates have been achieved under laboratory conditions by using this method but they typically required extremely large data request sizes on the order of multiple 100's of megabytes or more that are primarily sequential single stream/single user in nature.[8] By definition, a shared network server must deliver multiple streams of data to multiple users. Each network request, when using the NFS protocol, is limited to 8 kilobyte datagrams under NFS-2 and 60 kilobyte datagrams under NFS-3. Software disk striping, at least as currently implemented, does not seem to be the answer.

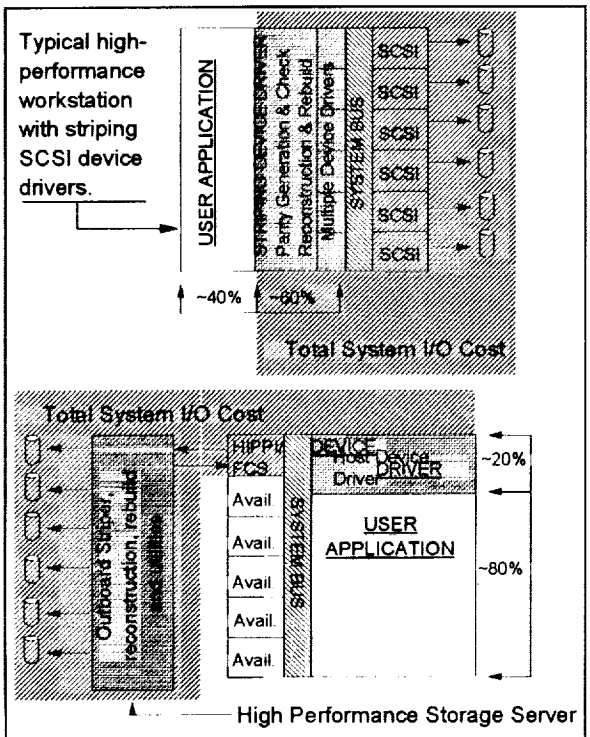

Figure 4: Software vs. Hardware striping

Massive data caching can address some of these concerns by preemptively reading ahead (i.e. turning small user requests into large I/O requests) multiple megabytes of data in order to achieve high bandwidth from the disk subsystem. Depending on the locality of reference, sequential (or non sequential) nature of the clients data access patterns, caching may or may not help. In all cases, allocating large amounts of main memory for preemptive disk caching is not cheap nor always possible without additional modification to most file systems and virtual memory managers.

Striping, and optionally software RAID artifacts, tend to add significant overhead to basic I/O operations. A 4+P RAID 5 stripe implemented on a set of generic SCSI drives and adapters requires 5 invocations of the basic disk I/O protocol stack (SCSI disk driver, card driver and unique device head codes) all contending for system I/O bus bandwidth and main memory access. Data transfer is not really parallel due to the Von Nuemann nature of most machines; rather it is (hopefully) rapidly interleaved in such a way as to appear parallel in nature to the requester. Sitting on top of these multiple device drivers would be a "collection manager" who's role in life is to re-assemble the users data records out of the disk chunks read by the device drivers (or on a write operation perform the "chunking" of the data records), verify correct parity and pass the re-assembled records to the user and/or to the file system buffer space.

We have noticed that it required approximately 5% of a large UNIX servers compute cycles to sustain a 4-5 megabyte per second data stream at the raw interface[9]. With the addition of "collection manager" overhead and optionally a software RAID function, the I/O overhead actually experienced by the user would be higher. To hypothetically sustain a 50 megabyte per second striped SCSI stream at the raw interface may consume up to 50-60% of a typical servers available cycles; thus not leaving much for any useful work such

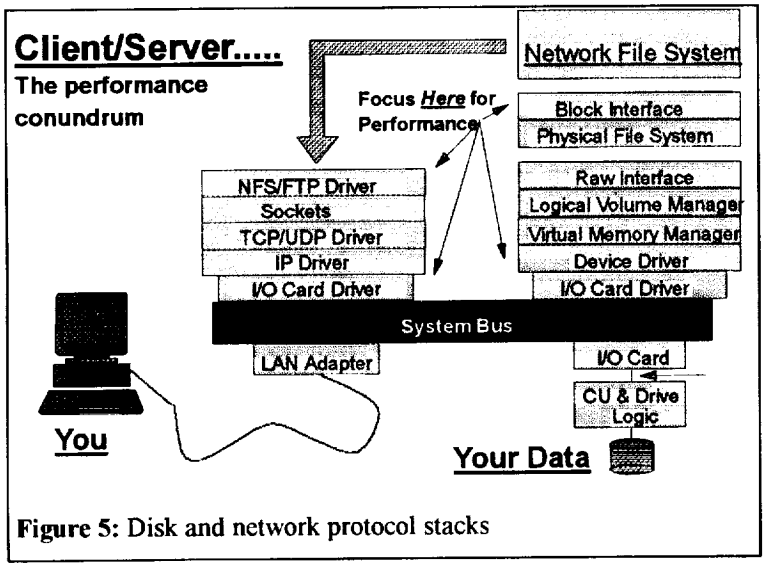

Figure 5: Disk and network protocol stacks

as actually delivering data to a client application. A better way to deliver data to the NFS server needs to be found.

The search for a better way

Based on what has been discussed previously, we recognized that using a general purpose UNIX (or other OS) compute platform as a "data bus" is inefficient, costly and may never deliver the performance required to satisfy the data absorption rate demanded by large HPC clients no matter what modifications we made to the software. Through-memory data transfer, system bus contention and general purpose I/O drivers were not designed to efficiently deliver high, sustained data rates and, when used in that capacity, deliver sub-optimal bit rates to your network clients.

Unfortunately, the worst is still to come. Assuming that the NFS server code finally gets the data it needs from the physical file system, the disk data blocks have to be sized to the users actual NFS request, packetized into datagrams and shipped over some fabric via IP protocol. Please see Figure 5 for a diagram of this protocol stack. As should be no surprise to the reader, another complete software protocol stack comes into play here further impacting the ability of a server to deliver meaningful data rates. Imagine all those little IP packets interrupting the I/O bus all the time, the OS frantically moving bits of data here and there through memory and the IP, UDP and LAN drivers all contending for precious CPU cycles. Performance problems are inevitable.

To eliminate these data bottlenecks you have to re-architect and completely re-define what a "server" is from the ground up. From our prior discussion, I think we can safely agree that it is not a workstation with lots of disk and some LAN cards. What it must be is a machine designed to manage and move large amounts of data efficiently and rapidly from disk storage to a fabric. Essentially, it should connect the disk subsystem directly to the network. It should scale (i.e. grow in usable bandwidth) as the clients and, as a second order, the request rate grows with no loss in performance. Since we are suggesting that many users entrust their crown jewels (i.e. data) to this machine, it should offer complete redundancy, virtually 7X24 access to the "jewels" and a bullet proof file system and backup scheme. A failure affects 10's to 100's of users, not just one or two. Since we are addressing high speed transfer of very large files on the order of multiple megabytes to gigabytes, file system corruption and/or physical disk storage failure could be catastrophic.

Client/Server network performance requirements definition

In order to solve the performance conundrum described above and design a file server capable of truly utilizing high bandwidth fabrics (e.g. ATM, FCS) you have to start with a

set of design points far above what has, to date, been deemed as acceptable. Some of these points that we picked for our initial design were as follows:

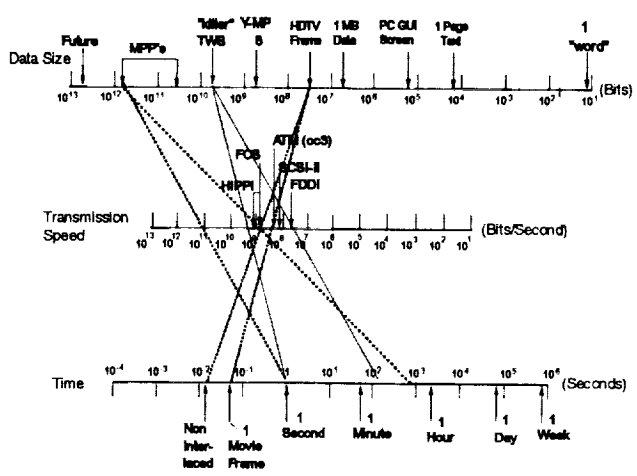

Figure 6: Data Demand by HPC Platforms and Applications

1. Sustained data delivery rates in excess of 50 Megabytes a second in order to utilize the bandwidth offered by FCS and/or multiple OC3 ATM links. As faster fabrics become available the server must be designed to accommodate them without extensive redesign of the architecture.
2. A scaleable design where the servers network bandwidth grows with the addition of more network ports vs. most current architectures where additional network ports deliver additional connectivity and fault tolerance but not necessarily more bandwidth[10].
3. Sufficient storage capacity to address the large data objects that graphical and "grand challenge" type applications tend to generate coupled to enough internal bandwidth to allow the server to access its storage subsystem fast enough to serve, for instance, multiple 155 megabit ATM OC3 links at rated speed.
4. 100% fault tolerance and 7x24 data availability for the reasons previously described.

Does anybody really need this?

Firstly, as sort of a reality check on the above specifications, we decided to more accurately understand if there is really strong market demand for extremely fast NFS/FTP servers. Earlier in this article, we discussed what was plaguing current server designs, but we did not discuss whether the user demand was 2X, 5X, 10X or whatever. Mark Seagert and Dale Nielsen at the Lawrence Livermore Computing Lab developed a model to illustrate the "data demand" of various compute platforms and/or applications. A version of that model, expressed as a Nomograph is shown in Figure 6[11].

The upper line, representing some common high performance computers aggregate data demand is compared to the lower line representing time (e.g., how long will a user wait for the data). Transmission speed (the middle line) can then be extrapolated from the size of the data object and the users "patience". We quickly realized that today's HPC demand is in the 100's of megabytes a second and growing fast.

We also interviewed most of our major customers and were able to identify four basic client/server application sets that had broad applicability in both the HPC community and the general commercial marketplace and required high sustained data delivery rates to perform well.

1. Animation: The studio standard for uncompressed, high resolution video is the D1 bit stream at 270 Megabits a second. Failure to deliver and sustain this isochronous bit stream will not allow for full motion playback of the digitized clip. All major studios, post-production and special effect houses are investing heavily in animation studios.
2. Simulation codes ability to accurately predict behavior improve as the number of points measured and the depth and width of the data stack associated with each point increases. From data preparation on workstations through large scale computing and eventual output display on frame buffers, massive amounts of data must flow through the network quickly and efficiently.
3. Data Mining: The "killer app" of the Ninety's says it all: Sifting through vast quantities of data to extract information useful to the client. Speed of data access (e.g. time to market) is everything.
4. Non-coded data storage and delivery; the collection of applications that concern themselves with the processing of non-coded (e.g. Video on demand, multi media, raw seismic data, high speed telemetry, image and pattern recognition etc.) data either deal with extremely large objects or deal with demanding isochronous data flow or, in many cases, both.

Based on the above and other market research we conclude that a large (and growing) segment of the client/server market could use a very high performance data server.

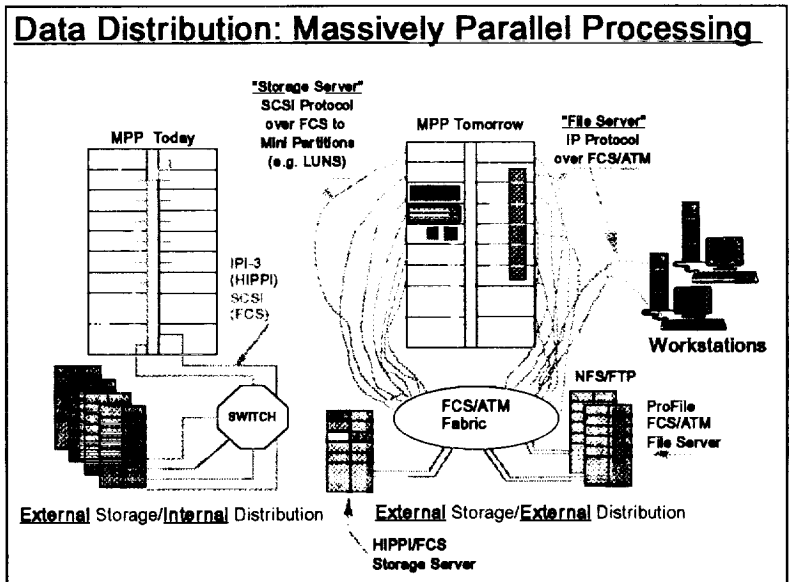

Figure 7: Data serving in a MPP environment

The MPP issue

The movement of massively parallel MIMD machines into the commercial sector coupled with the expected growth of full motion video data representations virtually guarantees that today's generation of servers will not be able to satisfy the data demand that these new applications and parallel processors will require to operate efficiently. Commercial applications tend to exacerbate the classic problems of delivering a large MIMD machine enough data to the correct node on a timely basis so as to actually utilize the massive compute power that it can bring to bear. Figure seven illustrates a conceptual idea of how very high bandwidth data serving might address this well known problem.

We are currently working with certain MPP vendors to further refine and validate the concept of massively parallel external data distribution.

Rising to the "grand" challenge

We realized that several challenges would have to be overcome to realize true high performance NFS and FTP bandwidth. Starting at the network access side, we recognized that we would have to provide multiple independent network ports which could be mixed or matched in any combination due to the heterogeneous nature of most users hardware install base. (Please see Figure 8.) HIPPI-IP, ATM-IP and Fibre Channel-IP were chosen to be the first three network interfaces supported for the following reasons:

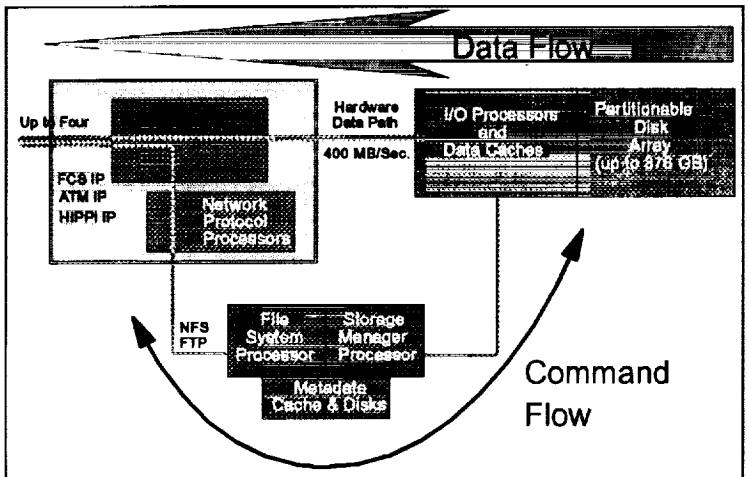

Figure 8: High Speed Server data flow ...

1. HIPPI, while somewhat costly and not as flexible as ATM and Fibre Channel, is here today, supports 800 megabit transfer speeds and is supported on most all high performance workstations, MPP's and traditional super computers.
2. ATM is rapidly developing into the high speed LAN/WAN of choice and, again enjoys near universal acceptance. While OC3 speeds of 155 megabits/second are lower that the full rated speed of our design, most clients cannot currently absorb IP data rates even that high. We recognized that as OC12 capable clients emerge, we would be well positioned to support that data rate.
3. The Fibre Channel Standard (FCS) supports gigabit transfer rates, is mature in its specifications and has been adopted publicly by IBM, HP and SUN. Other workstation vendors have told us that they plan to support this standard, at both quarter and full speed implementations, during 1995.

We considered FDDI and 10 megabit Ethernet but decided not to directly support these interfaces primarily because they lacked the bandwidth to support the marketplace we were interested in addressing and interconnection to these legacy networks could be handled by numerous vendors of routers and switches[12].

Outboard protocol processing

In order to achieve the scaleability criterion described earlier, we equipped each network port with its own integrated protocol engine to handle the IP, TCP or UDP protocol stacks completely within the attachment port. In addition, and modeled after some of the seminal work performed by the National Storage Lab (NSL) and others[13], we recognized the need to separate the command and control paths from the actual user data transfer paths so as to maximize the speed and efficiency of our internal data bus while providing for completely asynchronous and concurrent command flows.

Specifically, in the design we implemented, the outboard protocol stack engines strip the NFS and/or FTP payloads (RPCs) out of the network IP packets and route them off for processing by an independent dedicated filesystem processor. It is at least metaphorically

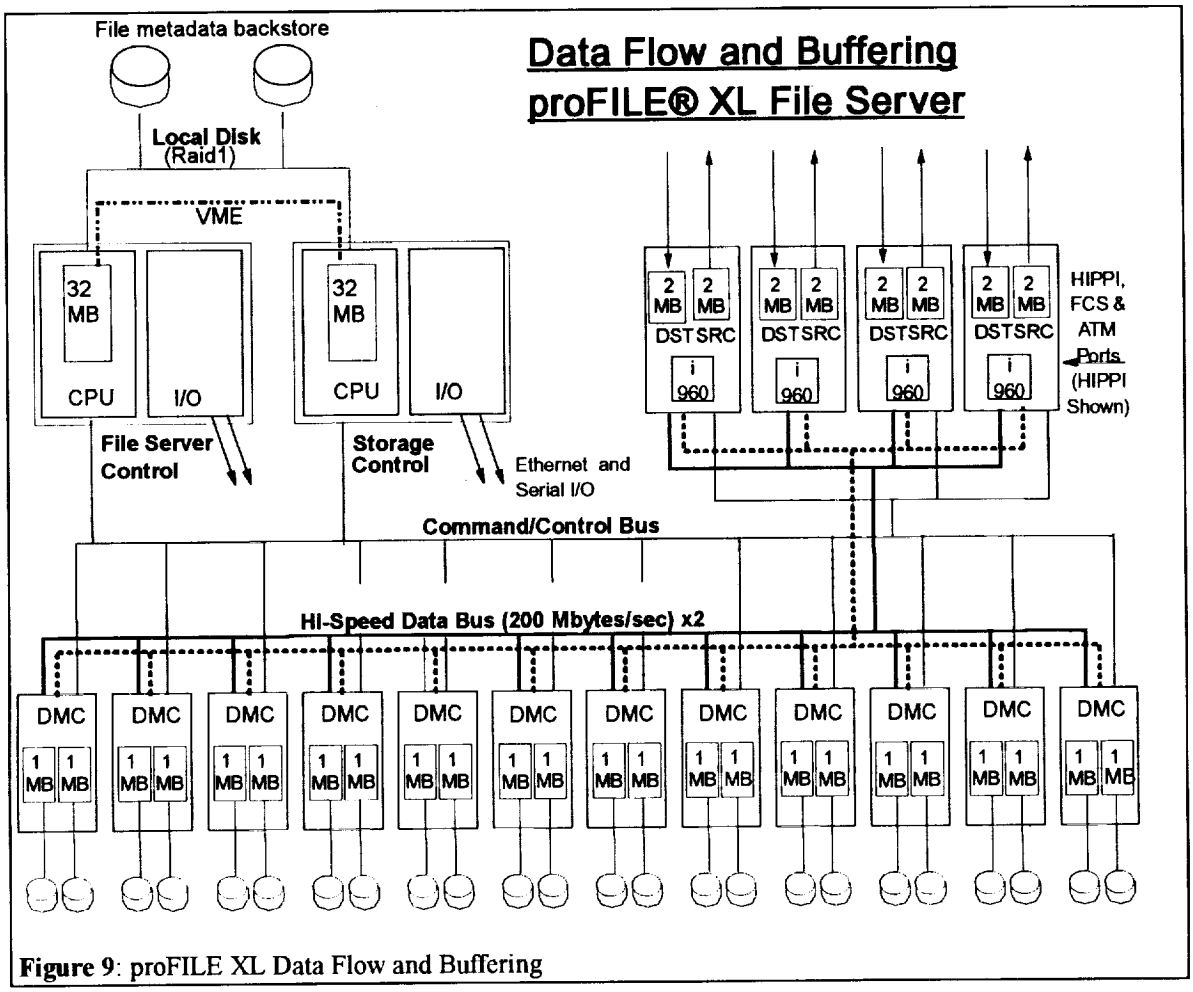

Figure 9: proFILE XL Data Flow and Buffering

correct to view the server's filesystem as a hardware file system rather than a shared software construct. A design of this nature allows the protocol engines to scale up as network connections are added to keep protocol handling from bogging down the server as users add connectivity and/or clients. This implementation addresses one of problems with today's fully software based servers; they do not scale well with connectivity and/or client load.

Filesystem processing and storage management

The file system processor, currently a Motorola 68060, accesses metadata (file identifier and i-node data) from a internal 32 MB local cache backed up by dedicated mirrored (RAID 1) metadata disks attached to both the file system processor and the storage manager processor on a local SCSI bus that is independent of the SCSI busses used to transfer user data. The local metadata cache is large enough to hold the metadata required to open and access approximately 50,000 user files under normal circumstances. Internal caching of filesystem metadata drastically reduces the time required to locate and access the appropriate user data disk in order to fulfill a NFS transfer request.

Connected to the File System Processor over a short inter-processor VME bus is a second, identical processor, the Storage Manager Processor, dedicated to managing the

physical organization of the user data and in setting up the transfers of the requested user data from the network ports either to disk or to the small write behind (fast write) caches located in the Device Module Controllers (DMC's). The DMC's are responsible for the attachment of the physical disk subsystem and can be considered to be "hardware" device drivers - please see Figure 9 This second processor, operating concurrently with the File System Processor manages the internal RAID 5 organization of the disk backstore, is responsible for management of the DMC write behind caches and controls any required recovery/rebuild processes should there be a failure of one of the DMC's and/or it's attached disks[14]. The Storage Manager Processor sets up, but does not manage, all data transfers from the DMC caches or disks to the appropriate network ports and vice versa. Over the command bus, the Storage Manager Processor instructs the DMC(s) to read or write the required number of blocks of data on/off each DMC's directly attached disk drives and transfer those disk blocks directly up to the network ports over the servers internal high speed data busses.

This function segregation between the File System Processor, The Storage Manager Processor and the multiple Protocol Processors has allowed us to not only scale the server as client connectivity and data demand grows but to tune each hardware process to efficiently implement just the functions that it was designed for and no others[15]. We refer to this design methodology as "MacroRISC(tm)" design: "Only those functions most needed shall be implemented on a processor and that processor shall be a RISC processor that efficiently implements those functions."

Internal data transfer bandwidth

The current design implements two (2) 200 megabyte/second redundant data transfer busses attached to the network ports and the DMC's via custom designed low latency chip sets. The replacement of through memory data transfer by internal "third party" transfers over 400 megabyte/second worth of hardware bandwidth eliminates another of the performance bottlenecks experienced by software only server architectures.

Each DMC is equipped with a Motorola 68020 processor that allows it to maintain its own queue of work and operate asychronously and concurrently with all other processes within the server. Under ideal conditions up to 24 simultaneous SCSI lower interface data transfer operations can be going on delivering an aggregate internal data transfer bandwidth of over 160 megabytes a second[14]. This 24-wide striped I/O subsystem allows for 100's of gigabytes of storage to be attached efficiently (e.g. no more than four LUNS on a SCSI bus) and could allow for the attachment of integrated tape backup systems, if desired, sometime in the future. Conceptually, what this distributed design does is to connect one or more disk drives directly to the network with no intervening software protocol stacks or memory bandwidth limitations.

Data availability

The File System Processor and the Storage Manager Processor are identical in hardware design and are designed to back each other up. They constantly monitor each others health and maintain mirrored metadata caches and system status latches. Should one of the two

processors fail, the other is capable of performing both the filesystem function and the storage manager function albeit at a significantly reduced level of performance. This takeover capability leads to increased levels of data availability as seen by the using clients.

A full UNIX-like system administration shell, implemented on its own administration processor, is provided for operational consistency with existing UNIX servers. A GUI interface for this shell is planned for mid '95 availability. Hot pluggable disks, imbedded RAID 0,1,3,5 hardware and N+1 power with an integrated uninterruptible power supply (UPS) complete the data availability aspects of the package.

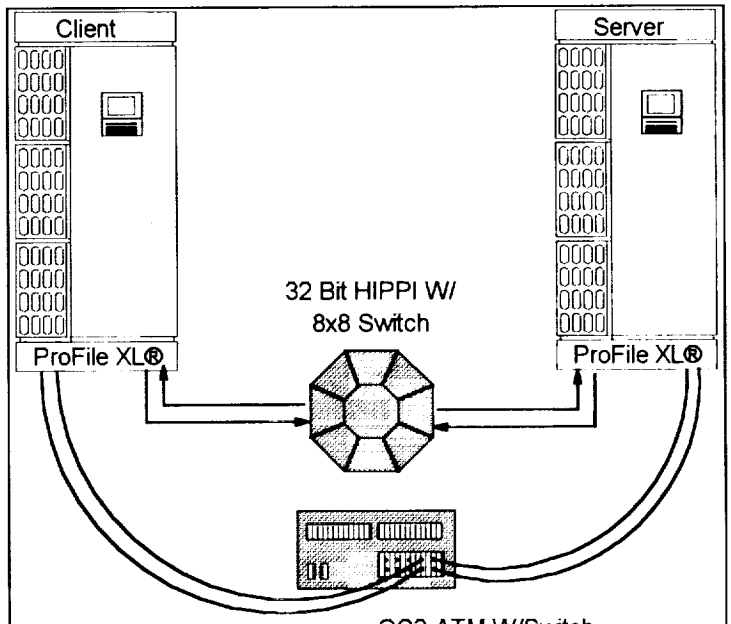

Figure 10: Early lab test bed

Measured performance and early customer experience:

When we set out to initially test the performance of beta level machines in our lab we rapidly realized that the existing "industry standard" NFS test suites based on LADDIS type workloads or the older NFS "stones" type of tests were not appropriate for this type of server. LADDIS and "stones" type tests are oriented towards measuring short, fast OLTP type workloads and not towards measuring sustained throughput of large files. Additionally, since we have optimized the server towards NFS-3 large datagram performance (although it also fully supports NFS-2 workloads) measuring short (e.g. 8K or less) requests would not allow us to test the sustained large datagram transfer rate. FTP performance was easier since measuring "throughput" is a simple matter of measuring how fast files of various sizes actually are transferred to various clients.

What to measure?

What we decided to use as a metric to represent data throughput was to measure the servers ability to deliver "N" Datagrams per Second, wherein datagrams can range in size from 8K (NFS-2 limit) to 60K (NFS-3). Sustained Throughput is the product of N and the datagram size.

During November and December of 1994 we were able to begin testing with beta level hardware and code. Recognizing that we did not have any client machines fast enough to drive the server to its limits we slightly modified one of our early servers to enable it to act as a fast client machine (See Figure 10). All initial measurements were taken utilizing 32

bit HIPPI channels. A later set was taken using OC3 ATM.[2] The graph (Figure 11) shows the relationship between throughput, datagram size and datagram delivery rate over a single HIPPI-IP port configured per the test bed shown in Figure 10 above. Several interesting items immediately come to light:

1. The server essentially has the capability to sustain a constant datagram delivery rate regardless of the size of the datagram packet. Values ranging between 800-1000 data delivery datagrams/second have been observed across all datagram sizes tested.
2. Because of observation one, delivered throughput is primarily a function of datagram size.

It should be noted that these tests were performed using a modified server as a "client" so as to remove, as much as possible, the "clients" effects on data rate. The strong relationship between datagram size and throughput may not be as linear with more traditional clients due to their potential inability to absorb high delivery rates of large NFS-3 style datagrams.

The specific initialization and opening state parameters for this test were as follows:

1. The file to be accessed had been opened and at least one request had been made so as to prime the read-ahead data caches and force the caching of necessary file and filesystem metadata.
2. Subsequent accesses were of a sequential nature.
3. The files accessed were 10's of megabytes in size or larger. Most of the small variances noticed are probably explained by file (request) size.

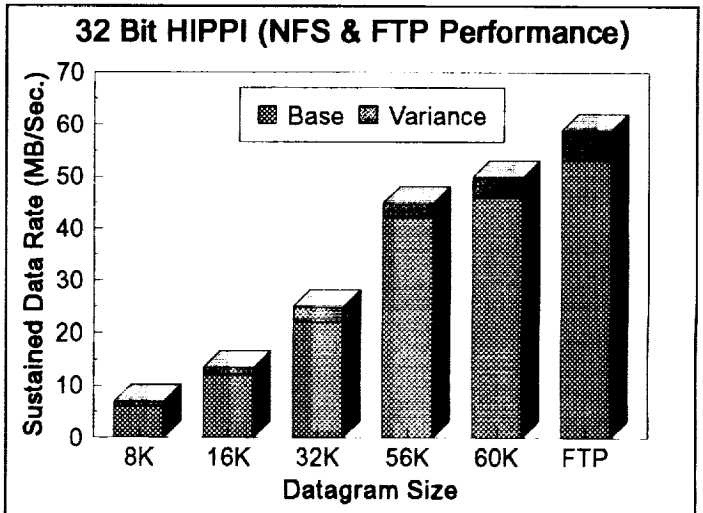

Figure 11: HIPPI-IP NFS/FTP Throughput

We felt that these initial state parameters were appropriate since the target use of this server is for applications where large files are to be accessed and the amount of data that the client requests is substantial. This test was designed to measure sustained data throughput to a client both requiring and capable of absorbing high speed IP traffic. Priming the data and metadata caches eliminates the initial latency of the "get attributes" sequences and most mechanical disk effects. Where file and request sizes are large, start-up latencies are essentially amortized over many megabytes of data transfer and become trivial. For small requests this is not the case and different start up states should be assumed for any kind of performance testing. We plan to perform more extensive performance testing over a wider range of workloads during the first half of 1995.

[2] **All results shown here should be considered preliminary due to the early levels of hardware and code used to perform these tests. Final data will be formally published in an update to this paper in the second quarter of 1995.**

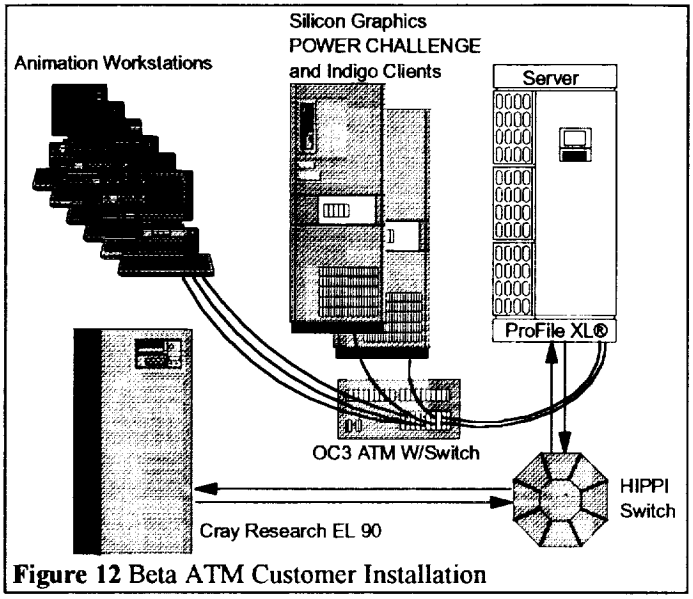

Figure 12 Beta ATM Customer Installation

ATM

ATM performance data is in the process of being developed more fully. Early lab results, obtained during December of 1994, have demonstrated the ability to sustain approximately a 12.5 megabyte/second (~100 Mbits) data stream over an OC3 (155 Mbits) ATM channel coupled to a FORE Systems *FORE-Runner*™ ATM switch. (See Figure 10) All switching and virtual circuit initializations were controlled by FORE Systems SPANS[16] interface code which we have implemented in the ATM versions of the proFILE server. The datagram size used to obtain these results was 56K and the initial state parameters were similar to HIPPI.

As of this date, December 1994, the maximum number of ATM channels that we have run simultaneously at this rate is two. No measurable degradation in performance was noticed. Both ATM channels sustained about 100 megabits/second of user data transfer. We expect that when all ATM performance tuning is completed sometime in the second quarter of 1995 that we will be able to saturate four OC3 ATM channels with large NFS-3 datagrams.

Early ATM Customer result:

In December of 1994 two proFILE HIPPI and ATM files servers were installed at a customer who's major application is various sorts of studio animation and video post processing[3]. The goals of this early beta site were fourfold:

1. Verify ATM-IP NFS operations when interconnected with FORE Systems products and Silicon Graphics POWER CLALLENGE XL(tm) and Indigo(tm) clients via the SPANS interface.
2. Verify HIPPI-IP (and HIPPI IPI-3) interconnect with Cray Research's EL and J-90 series of processors.
3. Measure what sustained performance could be achieved at various datagram sizes and application request patterns.
4. Insure proper operation to all clients in an NFS-2 and NFS-3 environment.

ATM & SPANS: Installing and setting up the ATM network went very smoothly. The ATM part of the network, with the exception of the ATM boards in the proFILE server which were designed by ourselves, was all supplied by FORE systems and operated well. A simple point to point star configuration was used for simplicity and guaranteed

[3] The results presented here are very preliminary and do not represent a production level environment; rather they represent interim results of an ongoing experiment.

bandwidth to each client. We and the customer specifically avoided complex mixed vendor fabrics due to the immature state of many ATM products and, more importantly, industry accepted specifications.

NFS-3: NFS-3, as implemented on pre-release versions of Silicon Graphics IRIX operating system Releases 5.3x and 6.1x[4] had some early-on stability problems and command/response state errors. This was not particularly surprising given that we were all working with non-released code and a completely new version of NFS. Fortunately, many of the NFS-3 problems were uncovered in our labs prior to install which made the installation far less painful that it might have been. Silicon Graphics was very helpful and responsive working with us to address any NFS-3 glitches in IRIX and our server code. Based on our progress to date, we expect that by the second quarter of 1995, NFS-3 will be ready for general availability and production use.

HIPPI-IP and Cray "big block" NFS: Cray Research, recognizing the performance limitations of NFS-2 years ago, implemented a proprietary Cray-to-Cray extension to NFS-2 that allowed the use of large datagrams up to 60K. This has proved to be very effective in speeding up interprocessor IP communication between Cray platforms. Peter Haas, at the University of Stuttgart, has measured sustained Cray NFS traffic up to 7.5 MB/second between a Cray Y-MP/2E server and a Cray C-94 client[17].

Datagram Size	Observed Data rate
8K	~1.8 MB/sec.
16K	~3.2 MB/sec.
32K	~5.4 MB/sec.
56K	~5.4 MB/sec.
60K	~5.4 MB/sec.

Table 1: HIPPI NFS Performance

When we initially installed the proFILE server on the Cray EL, it was configured as a storage server utilizing the IPI-3 protocol over HIPPI. Everything worked well, with data rates observed in excess of 50-60 MB/second. After determining that IPI-3 disk protocol operated correctly with the EL, we upgraded the proFILE to full file server mode and ran some initial NFS test runs at various datagram sizes.

The following table (Table 1) shows achieved data rates as a function of datagram size. We discovered that UNICOS 8.0 (Cray's operating system) would not generate a packet larger than 32K even though it was configured to do so. This problem was confined to the EL series and has been identified and corrected by Cray. Because of this, there was no throughput improvement above 32K datagram sizes. We plan to publish updated performance numbers when we retest with updated proFILE and UNICOS server code in early '95. We expect to see substantial improvements at that time.

ATM NFS Performance: As previously mentioned we achieved effective ATM saturation rates of over 100 megabits/second of user data when running two proFILE platforms as client and server respectively. (See Figure 10) When configured as per Figure 12 (proFILE to SGI Indigo) we achieved 15-18 Mbits/second sustained NFS transfer rates

[4] IRIX releases coded as "5.2x" support 32 bit hardware platforms. Versions coded 6.0x support 64 bit platforms. Release 5.2 and 6.0 are at the same basic function level. The "x" represents a non released version of the OS.

per physical ATM channel using NFS-3 8K packets. IRIX 5.3x's support of NFS-3 does not yet support any datagram sizes larger than 8K nor the ability to configure significant additional quantities of UDP datagram buffers. We expect this situation to be corrected in 1Q95.

Given that the proFILE server can hold a constant datagram delivery rate regardless of datagram size and that the client seemed to be primarily gated by the virtual memory manager, IP protocol stack and the NFS RPC interrupt handler components, we can extrapolate performance in the range of 50-80 Mbits/sec. when IRIX fully supports large (56K) datagram sizes and sufficient UDP buffering is available. (e.g. The OS client components most involved in limiting datagram absorption rate will be executed far less frequently.) Updated information will be provided in a revision to this paper later in 1995.

While both we and the customer are pleased with these early results we feel that they are not indicative of the throughput that we can achieve with production level operating system code, some application tuning and, most importantly, experience.

Summary

The distributed, parallel server design implemented in the proFILE family of network data servers has promise to revolutionize and make practical the concept of file serving to truly high performance client machines. By eliminating software protocol stacks, system and I/O busses, memory accesses and operating system overheads inherent in most "servers" today, performance levels that used to be available only on a local file system can now be delivered (and potentially bettered) on a remote, shared file system. On the client and network side, the full implementation of the NFS-3 protocol suite and the availability of fast fabrics completes the picture.

For the first time, continual data starvation will become a thing of the past and the promise of high performance Client/Server computing will become a reality.

Endnotes and References:

[1] An informal count of physical attachments in use a few years ago was on the order of 35 or more. Some examples were: IBM BMX & ESCON, DEC Q-BUS, CI & BI BUS, Univac word channel, Boroughs A-Series disk interface, NCR direct connect, SCSI: (1&2, Fast, Fast/Wide, differential and open ended), IPI: (2 & 3, Voltage or Current mode and IBM's DFCI used on the AS/400), SMD, and numerous others.

[2] Typical installations of this sort may have utilized an IBM 7094 for arithmetic operations and one or more IBM 1401 processors as I/O support machines. Early CDC 6x00 installations were similar. Cray Research users often employed large IBM and Sperry mainframes dedicated to data preparation and input support and, more importantly, as permanent data repositories. The Los Alamos Common File System (a.k.a. "Datatree" - in it's commercial incarnation) was a well known example of this scenario.

[3] Examples of some common I/O channel to I/O channel "black boxes" were the Network Systems Corporation (NSC) "DX" (Data eXchange) family of interconnect products and, at a higher function level, the Ultra-1000 network hub offered by Ultra Network Technologies.

[4] CFS was brought up in 1979 on an IBM 370/148 processor running the MVS (Multiple Virtual Storages) operating system. Datatree, released by General Atomics is functionally equivalent to CFS release 56.

[5] Most peer to peer LANS and WANS today employ one of two generic schemes to allocate bandwidth to multiple users at the physical level: 1) CSMA/CD (Carrier Sense Multiple Access/Collision Detect) which is employed by Ethernet type LANS wherein the client "listens" to the network and, if it seems to be free, transmits its data. Obviously, collisions (and retries) are common during heavily loaded periods, or 2) token controlled access, (Token Ring, FDDI, etc.) wherein a user must have access to a "token" to transmit data. Controlled access fabrics rarely have collision problems, but suffer from the higher overhead required to manage and share the token. See ANSI standard document 802.xx for further reading.

[6] There was one major exception to this statement. Under the direction of Dr. Richard Watson, the National Storage Lab (NSL) located at the Lawrence Livermore National Laboratory (LLNL) directed it's focus towards serving HPC platforms at very high speeds primarily via utilizing a construct called Third Party Transfer over HIPPI networks. (See reference 13) All data transfer was under control of a modified version of Unitree (NSL-Unitree) and is commercially available from IBM's Federal Sector Division.

[7] SCSI data rates delivered to various points in the protocol stack (driver level, raw, and block interfaces) were obtained via the use of an IBM tool "perfmon" running on an RS/6000 98B with AIX 3.2.5. There is no guarantee that any user can or will obtain these results. They are presented for illustrative purposes only. Please see IBM publications GA23-2704-00 and GA23-2708-00 for similar information concerning achieved data rates over HIPPI channels. Interestingly, while the numbers are different, the ratios hold.

[8] Ruwart, T. M. and O'Keefe, M.T. 1993. "*Performance of a 100 megabyte/second disk array*" (Preprint 93-123), University of Minnesota, Minneapolis M.N.

[9] This approximation was developed by measuring the cycle consumption required for an IBM RS/6000 980 server to sustain a 50 MB/Second data rate over a HIPPI channel utilizing the IPI-3 protocol. It required approximately half of the available processor cycles to achieve this rate thus allowing us to extrapolate that every 5 MB/sec of data rate required 5% of the processor. Striped SCSI, due to the larger number of small I/O chunk requests and the need to re-assemble such chunks would require more. For furthur reading please see:

- Arneson, D., Beth, S., Ruwart, T. and Tavakley. 1993 "*A testbed for a high performance file server*" Procedings of the 12th IEEE Symposium on Mass Storage Systems, April 26-29, Monterey C.A.
- Chen, P.M. and Paterson, D.A. 1990. "Maximizing performance in a striped disk array" Proceddings of the 1990 International Symposium on Computer Architecture, pp. 322-331.

[10] In certain extreme cases, the addition of additional I/O ports on UNIX workstations configured as servers may actually have the effect of reducing the overall throughput of the server. This counter-intuitive phenomena results from the higher multi-programming level necessary to manage the increased number of I/O ports and data movement operations that result from such additions. The increased interrupt rate across the system bus and within the OS can lead to diminished overall throughput. Data supporting this

observation, developed on an IBM RS/6000 980 server driving multiple HIPPI channels is available from the author on request.

[11] The original Nomograph upon which the representation shown in this paper is based was developed by Dr. Mark Seagert's and Dr. Dale Nielsen, both at the Lawrence Livermore Computing Lab, as a method of estimating the data demand and transfer speeds required to feed future generations of processors envisioned at LLNL. Additional data points relating to ATM and video frame transmission were added by the author.

[12] Vendors that we are aware of today who have either announced products or announced their intentions of developing products to interface HIPPI, FC and/or ATM to existing Ethernet and FDDI networks consist of Netstar Inc., Essential Communications, Bay Networks and FORE Systems. Additional vendors have announced intentions to enter this market in some form or another.

[13] Hyer, R., Ruth, R. and Watson, R. 1993. *"High performance direct data transfer at the National Storage Lab"* Procedings of the Twelth IEEE Symposium on Mass Storage Systems, Monterey, C.A. April 26-29, 1993.

[14] Wood, L. C. 1994. *Gen 5 Storage Server - General Information.* Maximum Strategy Inc., Milpitas CA.

[15] The author would like to recognize the invaluable contribution of John Lekashman, Bruce Blaylock, Bob Ciotti, and many others at NASA-Ames for assistance in the design and validation of the specific function splits described in this paper. Without their help in measuring and understanding the choke points in NFS data flow we would not have been able to accomplish this project in the time frame required.

[16] SPANS (Simple Protocol for ATM Network Signalling) is a proprietary API developed by FORE Systems as a method to set up and control FORE's family of ATM switches. The current lack of a complete standard for ATM has led to the development of several competing proprietary access and control schemes; most of them not compatible with other vendors switch and interface hardware.

[17] Haas, P. 1994. "Optimal UDP buffering for UNICOS 8.0 NFS" University of Stuttgart, Stuttgart, Germany. (an unpublished work)