The Myth of Transfer Rate

How and Why SCSI Is Better than IPI for NFS

Bruce Nelson
Yu-Ping Cheng
Second Edition
July 1992

Abstract

Disk drives are often dismissed as mundane devices, but they are actually interesting, complicated, and misunderstood. As used in traditional Unix compute servers or general-purpose servers, disk storage subsystems have usually been optimized for excellent sequential-transfer performance. Perhaps counter-intuitively, however, NFS file servers exhibit marked *random-access* disk traffic. This report explores this apparent contradiction. It investigates the anatomy of a client-to-server NFS I/O operation in complete, microsecond-level detail. It debunks the myth of disk transfer rate, and shows instead that disk-drive concurrency is the most important factor in disk storage performance for most NFS network file servers.

The paper presents a careful analysis of both SCSI and IPI disk drives. It begins with a concrete and detailed comparison of both performance-oriented and non-performance-oriented technical specifications of SCSI and IPI drive and interface types. It offers a thorough empirical evaluation of SCSI disk drive performance, varying parameters such as synchronous or asynchronous bus transfers, random and sequential access patterns, and multiplicity of drives per SCSI channel. It discusses (nonempirically) similar characteristics for IPI-2 drives. Measured disk traffic patterns are presented for several >5-GB NFS servers using SCSI disk arrays. As can be expected in a 40-200 user file server, request rates are random and independent--not sequential as often benchmarked in general-purpose servers.

The report concludes with carefully benchmarked comparisons of file servers using SCSI-based disk arrays (like the Auspex NetServer family) and IPI-2 subsystems (like the Solbourne 5E/900 and Sun SPARCserver 600MP). The results show that NFS heavy-load throughput using SCSI disk arrays scales linearly with extra drives, whereas IPI-2 throughput scales less than proportionally with extra drives. This SCSI performance advantage, combined with its outstanding storage density, price-performance, and price-capacity, make SCSI disks a superior choice for NFS servers. IPI-2 drives, with their very high transfer rates, remain an excellent choice for compute and simulation servers executing large-file applications where sequential throughput is essential.

Document 300-TC015, V2.2.2, 920814.

Auspex Systems Inc.
2952 Bunker Hill Lane
Santa Clara, California 95054 USA
Phone: 408/492-0900 . Fax: 492-0909
EMail: Info@Auspex.com.

A shorter, edited version of this report appears in the *Proceedings of the USENIX Winter 1992 Technical Conference*, San Francisco, California, 22-24 January 1992, pp. 253-270.
Presentations based on preliminary versions of this report appear in the *Proceedings of the Sun User Group Conference*, San Jose, California, December 1990 and 1991.

Preface to the Second Edition

This second edition of *SCSI and IPI* reflects the annual advance of disk technology in the eighteen months since the first edition of January 1991:

8-inch disk technology--either SMD or IPI--is effectively dead in the Unix market. Thus references to 8-inch disks are mostly eliminated in this edition.

3.5-inch disks are now available with both high performance and high capacity (>=1GB). Thus 3.5-inch disks are introduced and compared in this edition.

SCSI's superiority is especially clear in the 3.5-inch form factor, with outstanding end-user price-capacities of ~$3.75/MB, dropping rapidly.

SCSI-2 disks are now shipping. Initially, most SCSI-2 drives simply offer SCSI-2's extended software command set, possibly with the "fast" 10-MB/s 8-bit-wide transfer mode. But "wide" 16-bit SCSI is advancing quickly, with host adapter VLSI chips now available to OEMs, and drive manufacturers supporting 16-bit cabling.

More disk manufacturers and computer vendors support SCSI, driven directly by SCSI's price-performance and price-capacity victories. Some disk manufacturers are abandoning support for IPI; others skipped IPI altogether (e.g., Hewlett-Packard). Some Unix vendors that previously supported SCSI primarily for their workstations now have medium-capacity SCSI-disk server subsystems as well (e.g., SGI and Sun). Finally, StorageTek, an IBM-plug-compatible mainframe "DASD" supplier, formally announced its Iceberg 5.25-inch SCSI RAID-5 disk array product.

In January 1992, a shortened version of this paper was presented and published at the USENIX Winter technical conference in San Francisco. Since that time, the following changes have been incorporated in this full-length edition:

Tables are updated with the latest, production-level specifications for the new drives.

A new chart, 2b, is included that shows SCSI channel performance with up to *six* drives per channel in both random- and sequential-transfer tests.

Section 4.1--The Anatomy of an NFS Operation--is revised to show the new, faster *system-level* timings that result from using the latest SCSI drives in a shipping NFS server.

To conclude this preface, then, SCSI storage now sweeps across the full range (and scale) of contemporary computer systems.

-- Nelson & Cheng, July 1992

1 Introduction and Terminology

1.1 Disks and NFS

Disk drives are complex storage devices. The typical speeds of different disk functions span five decimal orders of magnitude--almost a million to one--from transfer rates of about 25 Mb/s to access times of about 10 ms. Individual disk-drive complexity is compounded when disks are used in computer systems, where *overall* storage performance is governed by hardware-software interactions. Consequently, common disk performance metrics such as transfer rate and access time can be misleading indicators of

total *system* performance. This report examines disk subsystem design in the context of a specific, system-level storage application: the NFS Network File System [Sandberg85].

NFS file server traffic has an often-surprising disk access pattern: *random* I/O using small (8-KB) blocks. This is in complete contrast to the disk access patterns of compute servers or stand-alone workstations, which are usually sequential. Recognizing this random-access NFS traffic pattern--and then designing for it--is the crucial conceptual element of high-performance NFS storage design.

As mainframe database vendors have long known, random-access disk storage performance is directly related to the number of disk actuators and storage channels.

An *actuator* is an independently operated disk arm-and-head assembly that can move to a specified disk address and transfer data there. Most disk drives have only one actuator (there are exceptions in mainframe and supercomputer products).

A *storage channel* or *channel* is a hardware datapath that can funnel data from an actuator's heads to and from main memory. Conventional Unix-system disk controllers usually have only one channel per controller, although two is not uncommon.

Excellent random-access disk subsystem performance requires a fine balance between many actuators, multiple channels, and peak system I/O demands.

The objective of this report is to show that SCSI disk technology easily outperforms IPI-2 and SMD technology in NFS server disk subsystems. Recent commercial developments seem to confirm the price-performance advantage of this conclusion: Both Silicon Graphics and Sun Microsystems--once SMD and IPI stalwarts--are chasing Auspex with initial, medium-capacity SCSI subsystems for their server products.

While IPI and SMD are outstanding technologies for sequential access, they are not cost effective for high-performance random-access configurations. SCSI disk technology, in contrast, offers a built-in balance of actuators and channels that intrinsically optimizes random access at low cost. Because our system-level goals include low price and large capacity as well as high performance, this report explores these important issues in conjunction with performance.

1.2 Performance Measured as Throughput and Access Time

In this report, throughput is the primary metric of disk subsystem performance. Disk subsystem *throughput* is the capability of the entire subsystem to perform work.

Sequential transfer throughput is usually expressed as an aggregate transfer rate in MB/s. It is typically measured by summing the sequential transfer rate of each disk that can be attached to a separate, available channel. High sequential throughput is useful for the large data transfers (>>100 KB) characteristic of data-intensive technical and scientific applications executing on compute servers.

Random access throughput is expressed as an aggregate read or write I/O rate on small-block transfers in disk I/O operations per second (disk IOPS). It is usually measured by summing the I/O rates of all independent disks on each separate, available channel. In this NFS-oriented report, a disk's small-block random-access I/O rate is the disk's ability to perform 8-KB block transfers from uniformly distributed addresses across the entire disk.

Sequential throughput favors high transfer rates and is less sensitive to initial positioning (access) times. Random throughput favors fast access times and is less sensitive to transfer rate. Unless otherwise mentioned in this report, disk throughput and disk performance refer to *random access* throughput because random access throughput is the essential metric for NFS performance.

1.3 Latent SCSI Skepticism

Readers whose previous experience with SCSI disks has been limited to the Macintosh, IBM PC, or CISC-generation workstation arenas may be skeptical of "high-performance" SCSI claims. These prejudices are justified. Early SCSI drive implementations, and many contemporary low-cost, low-capacity implementations, do (or did) not provide basic performance comparable to 8-inch SMD or IPI drive technology. However, current-generation, high-capacity (>1 GB), 5.25-inch SCSI *and* IPI disks do provide performance comparable to or better than their 8-inch cousins.

Fortunately, the SCSI *specification*--as opposed to early SCSI realizations--permits a wide spectrum of performance implementations. An examination of actual high-performance SCSI drive specifications in section 3 will dispel latent SCSI prejudice.

1.4 The SCSI-1 and SCSI-2 Specifications

SCSI is an acronym for *small computer system interface*. The original SCSI-1 specification [SCSI1] defines an 8-bit-wide bus that operates at 1-2 MB/s asynchronously and up to 5 MB/s synchronously. The SCSI-1 *common command set* (CCS) comes in two parts: the mandatory commands like *inquiry*, *read,* and *write* and optional commands like *readbuffer, writebuffer*, and *logselect*. Different vendors often implement non-common--or semantically incompatible common--subsets of SCSI-1 commands. Consequently, many SCSI-1 drives are software plug-incompatible, especially the high-performance drives with more available commands.

The new SCSI-2 specification [SCSI2] defines 8-, 16-, and 32-bit busses, operating at maximum synchronous speeds of 10-, 20-, and 40-MB/s. Eight-bit SCSI-2 is defined to be hardware backward compatible with SCSI-1. The SCSI-2 command set is more precisely defined and functionally richer than SCSI-1. It is also software backward compatible with SCSI-1. This has three benefits:

SCSI-2 drives can initially be used with SCSI-1 hosts and software drivers.

Different vendors' SCSI-2 drives will be more freely interchangeable because of accurate SCSI-2 common command set implementations.

SCSI-2 drives, because of more built-in intelligence such as command reordering and multisegment caching, will behave better in low-end computer systems that may perform no disk optimization in higher-level software (e.g., Macintoshes and PCs).

In this report, SCSI-1 and SCSI-2 can be considered equivalent unless otherwise specified.

1.5 IPI-2 and SMD Functional Equivalence

IPI and SMD are acronyms for *intelligent peripheral interface* and *storage module device*. Because IPI-2 and SMD both have *device-level* disk interfaces (explained later in section 3.5), they are functionally equivalent for the purpose of this report. For simplicity, then, the remainder of the report mentions only IPI disk drives, and uses unsuffixed IPI to refer to IPI-2.

2 Why NFS Servers Experience Random Disk Traffic

2.1 A Server's View of NFS Clients

Figure 1 shows a typical large NFS network. From 40 to 200 client workstations are being served from a single NFS server. Notice that the server "sees" aggregate, mixed NFS file traffic as it arrives spontaneously and independently from all workstations on the attached networks. Now consider two facts of NFS implementation:

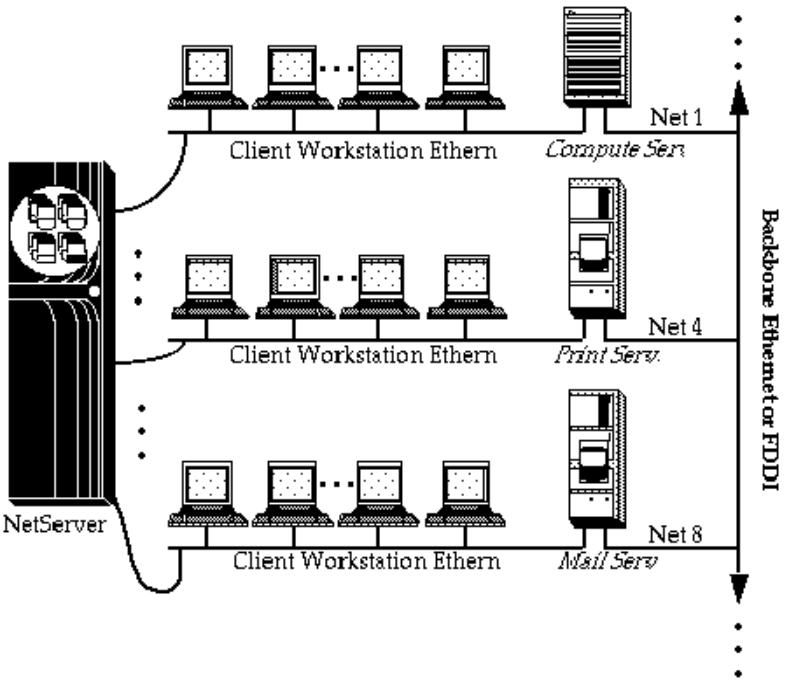

Figure 1: A large NFS network with 40-200 client workstations served by a single NFS server. The key notion is that the manifold users offer an *aggregate* random-access disk workload. This is because it's common in medium-to-large Unix networks to attach 4-8 Ethernet segments, each loaded with RISC workstations, to a consolidated data server. Auspex NetServers are one example of consolidated NFS servers. Many organizations are unifying an overgrowth of traditional, limited-performance file servers onto large, consolidated NFS servers. The reasons are many: up to one-third lower cost per seat; a single point of administration and backup; higher NFS throughput; increased disk and network expandability (scalability); better reliability by excising NFS crossmounts; eliminated free space fragmentation; reduced /usr and project library file replication; and decreased power and floor space requirements.

Small blocks are mandatory. NFS I/O requests are limited to a maximum of 8 KB per read or write operation. This 8-KB maximum is required *by the definition of NFS* [Sandberg85]--and by the operating implementations of over 300 licensed NFS vendors. This means that to transfer a 1-MB file between a client and server, the client must issue 128 *separate* read or write requests, and the server must respond to each and every one of these small 8-KB requests. This consumes substantial network capacity and causes much client and server software overhead--but the NFS protocol permits no shortcuts.

Randomized sequentiality. Even in a best-case scenario where each workstation is *itself* sequentially reading or writing a large file, the server still receives an aggregate request stream that is unordered and random (assuming that most files are not shared simultaneously, almost universally true in department- and campus-sized technical environments). A single user's sequential intentions can be lost in

the randomness of the whole.

In total, an NFS server sees autonomous client workstations that collectively and rapidly issue random I/O requests for small data blocks.

2.2 The Nature and Limits of NFS Optimizations

Mild NFS performance optimizations are possible. Client workstations can help themselves somewhat by performing read-ahead or write-behind buffering within their own file systems (typically using block I/O demons (BIODs)). Servers could perform similar optimizations, but are usually constrained by the limited read-ahead policy of the Unix File System [McKusick84, Kleiman86], the simple-and-stateless objective of most NFS servers [Sandberg85], or the synchronous write policy of NFS itself. These complex issues of buffer cache management, per-client (rather than per-server) file contexts, and fast stable storage (to mask synchrony) are beyond the scope of this paper. They have been extensively addressed in non-NFS distributed file systems such as Sprite and Andrew [Nelson88, Rosenblum90, Howard88] that have traded some reliability for much greater performance. Srinivasan and Mogul [Srinivasan89] have applied some Sprite-inspired optimizations in a "Spritely NFS" prototype with moderate results. Lyon and Sandberg [Lyon89] have shown that fast stable storage for disk-level NFS write buffering can boost performance on write-intensive workloads (e.g., writes >=15%).

In practice, the most beneficial file-server performance optimizations have been applied to non-NFS file systems. This demonstrates the limited protocol evolution of NFS and hence the fundamental importance of a strong, underlying random-access storage subsystem for NFS itself. Keeping NFS servers simple (stateless) has explicitly forsaken some classes of software optimization for improved reliability. This is a worthwhile tradeoff, but it places strong demands on an NFS server's disk subsystem, which remains in the uncomplicated-but-unoptimized critical path of NFS server performance.

2.3 Storage Scalability and Disk Subsystem Design Goals for NFS

Disk storage *scalability* is a special concern in NFS servers. For example, growing an NFS server from 40 to 100 to 200 workstation users nominally requires not only double and then quadruple the storage capacity, but also *two and then four times the random-access disk I/O rate*. It is this random-access throughput increase, more than any other factor, that limits the NFS abilities of IPI disk technology.

This section has outlined why high-performance NFS file servers using standard extent-based file systems need large-capacity, scalable-throughput, random-access-optimized disk storage subsystems. (A traditional extent-based file system attempts to allocate files contiguously--and usually with less success as free space decreases. In contrast, a log-structured file system like LFS [Rosenblum90] batches temporally-local writes into physically-contiguous groups, but without regard to *individual*-file contiguity). This leads to some natural goals for NFS storage subsystem design:

Select individual disks with high random throughput (fast access times).

Design the storage controller to initiate fully concurrent seeks and allow parallel transfers.

Pre-sort disk seeks in "elevator" queues to minimize actuator time and motion.

Use disk drives (like SCSI) that allow autonomous seeks and buffered read-ahead transfers.

Use multiple storage channels that permit parallel, low-latency disk-to-memory transfers.

Finally, all of the above performance goals must be balanced against cost objectives. Price-performance and price-capacity tradeoffs must be made in the context of the NFS throughput and storage capacity requirements of the target file server.

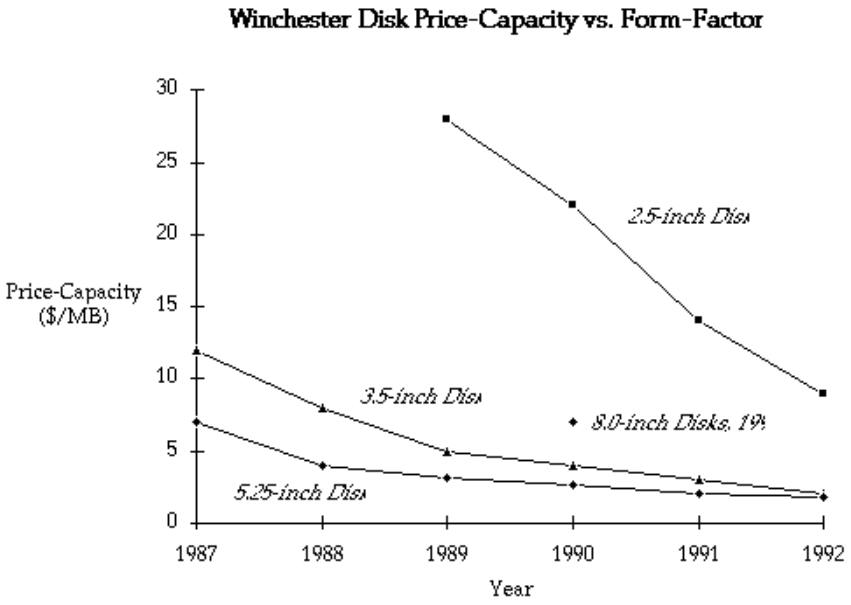

Chart 1: Disk price-capacity versus form factor. Both historical and future price trends are indicated. The 5.25-inch drive is the clear price-capacity leader, and will remain so until 3.5-inch drives are available in >= 1-GB sizes. Prices are wholesale, based on large-volume OEM purchases. The fundamental data is from Seagate and Dataquest.

3 SCSI and IPI Disk Drive Trends and Comparison

3.1 Trends in Form Factor and Price-Capacity

Chart 1 shows the annual decrease in disk price-capacity for four drive form factors--8.0-, 5.25-, 3.5-, and 2.5-inches. Price-capacity is a different metric from price-performance and measures normalized storage costs, usually in $/MB based on large-quantity OEM costs. The important trends in chart 1 are:

5.25-inch disks are the industry's current price-capacity leader.

3.5-inch disks should overtake 5.25-inch in 1992 in raw price-capacity, although equivalent overall performance in >= 1-GB sizes could lag another year.

8-inch disks were nearing the end of their lifecycle in 1990 as their price-capacity ($7.00/MB) was higher than either 5.25-inch ($2.70/MB) or 3.5-inch ($4.00/MB) disks. By the end of 1991, 8-inch IPI disks in high-performance Unix systems had been mostly replaced by their 5.25-inch successors.

2.5-inch disks, popular in laptop computers, will remain premium-priced for several years.

In summary, the highest-capacity SCSI disk drives will be implemented in 5.25-inch form factors for several more years. The highest-capacity IPI disks will be 8-inch, but will be used only in high-end supercomputer-like applications; 5.25-inch IPI will be used in most Unix compute servers.

3.2 Head-Disk Assemblies for SCSI and IPI

A *head-disk assembly*, or HDA, is the electromechanical portion of a disk drive. It consists of platters, spindle, motor, seek actuator, arms, read-write heads, etc. Interestingly, the HDAs of a single vendor's SCSI *and* IPI high-performance disks will usually be identical in the future. The SCSI interface will appear first on smaller form-factor HDAs, driven by personal-computer needs. But only the interface electronics will distinguish SCSI and IPI in the same form factor. For instance, 5.25-inch SCSI has been common for several years, with 5.25-inch IPI appearing in late 1991 (e.g., Seagate's high-end 5.25-inch SCSI and IPI disks use the same HDA). High-performance 3.5-inch SCSI is now available (e.g., from IBM); 3.5-inch IPI will lag and probably appear in 1994 (if ever).

This implies that the differences in SCSI and IPI manufacturing cost will be based solely on the drive interface electronics. These *costs* are usually (but not always, depending on volume) more expensive for SCSI because of read-ahead track buffers and more powerful embedded microprocessors. Yet SCSI drive *prices* are expected to remain substantially *lower* than IPI because of intense vendor competition and volume SCSI shipments. The massive scale of SCSI production actually reduces the cost of SCSI manufacturing. These competitive and volume factors will make SCSI the absolute leader in price-capacity, and among the leaders for price-performance.

3.3 SCSI and IPI General HDA Comparison

Table 1 is a comparison of three manufacturers' HDAs: 5.25-inch "Elite-2" IPI from Seagate [Seagate91], 5.25-inch "Coyote-4" SCSI from Hewlett-Packard [HP92], and 3.5-inch "Corsair" SCSI from IBM [IBM91]. The IPI drive is representative of those used by Silicon Graphics, Solbourne, and Sun in their general-purpose servers. The 5.25-inch SCSI drive is representative of those used by Auspex and others in NFS file servers, and the 3.5-inch SCSI drives of those used in workstations and personal computers. Reviewing table 1, some results are:

SCSI's higher recording density. The 1-GB IBM SCSI drive has 20-40% higher recording density than either of the 5.25-inch HDAs. Recording density and miniaturization are the primary technology thrusts of drive manufacturers, so SCSI's advantage here represents a certain amount of vendor leap-frog activity. The important point is that competitive workstation factors will push 3.5-inch SCSI technology the farthest in the next few years, and that IPI will move from its late-1991 introduction in 5.25-inch form more slowly to the 3.5-inch HDA platform.

SCSI's volumetric efficiency. SCSI has over twice the storage capacity of IPI in similar cabinets, since SCSI--in 3.5-inch--leads IPI--in 5.25-inch--in smaller form factors by several years. This is a result of both smaller-form-factor drives and lower-power-dissipation drives, which can be packed tightly in rectangular arrays without cooling problems. This volumetric efficiency can result in substantial floor-space (footprint) savings.

Non-Performance Metrics	ST41800K ipi-2	HP C3010 scsi-2	IBM 0663 scsi-2
Customer Shipment Date	March 1992	May 1992	November 1991
Size (form factor)	5.25 inch	5.25 inch	3.5 inch
Formatted Capacity (gb)	1.43 est. (1.986 unf'mt) gb	2.003 gb	1.004 gb
Track buffer size (kb)	0 kb	256 kb	256 kb
Spindle-synchronization Option	Yes	Yes	No
Dual-ported Controller Option	Yes	Yes	No
Typical End-user List Price	$6,400 est.	$7,500 est.	$3,800 est.
Price-capacity ($/mb)	$4.48/mb	$3.75/mb	$3.78/mb
# of Drives in 19x72-inch rack	50	50	>= 150
Volumetric Efficiency (gb per full-size cabinet)	72 gb	100 gb	>= 150 gb
mtbf (rated hours)	250,000 hr	300,000 hr	400,000 hr
Power (watts during seek)	44 W	38 W	16 W
Surfaces (data + servo)	18 + 0	19 + 1	15 + 1
Tracks/Surface (data + spare)	2,627 + ?	2,255 + 70	2,051 + 0
Sectors/Track (data + spare)	68 + ?	76-96 + 0	66 + 30-40
Constant-density Recording	No	Yes (for data)	Yes (for spares)
Tracks/inch (tpi)	2,250 tpi	2,000 tpi	2,238 tpi
Linear Bit Density (bits/in)	42,000 bpi	47,751 bpi	58,874 bpi

Table 1: A SCSI and IPI HDA comparison on non-performance metrics. Some comments on the metrics: *Formatted capacity* is measured after hardware disk formatting of 512-byte sectors, but before Unix *newfs* formatting. Formatted IPI capacities are controller dependent, usually 80-85% of unformatted. Parallel-head drives such as the dual-head ST41800K IPI drive usually lose about 10% of their capacity in multihead operation. This is because more embedded control information is consumed at hardware formatting time (e.g., the union of bad spots, a doubling of spare sectors and tracks, etc.). The advantages of SCSI's *track buffers*--not now available in IPI--are discussed in section 3.5.1. *Spindle synchronization* is useful for ganged high-bandwidth disk-array configurations, especially RAID-3 [Patterson88]. *Spare tracks and sectors* are assigned during hardware formatting to allow the disk controller to remap bad spots with minimal performance impact. For IPI, sparing is controller dependent, so spare tracks and sectors are indicated with ? in the table. *Constant-density recording* (sometimes called *zone-bit recording*) adds more sectors per track as track radius increases, usually in a few fixed zones of constant sectors/track. This boosts disk capacity at the expense of extra controller firmware complexity.

3.4 SCSI and IPI HDA Performance Comparison

Table 2 is a performance comparison of the three HDAs reviewed in the previous section: 5.25-inch dual-head IPI from Seagate, 5.25-inch SCSI from Hewlett-Packard, and 3.5-inch SCSI from IBM.

A review of table 2 discloses generally minor drive differences: The IBM HDA has a slightly longer rotational latency because it rotates 1,100 RPM slower than the other drives. The IBM HDA has the best seek times because of its new-technology "nanoslider"

actuator. Command overhead times (which reflect controller software and hardware overhead) have similar figures, with HP leading. Transfer rates differ: the dual-head Seagate drive is obviously fastest; the HP drive is next because of its high recording density and high rotational speed; the IBM HDA's exceptional bit density is counterbalanced by its smaller form factor, which reduces track linear velocity and thus transfer rate. This leads to two final conclusions:

Similar small-block I/O rate (random-access throughput). For all drive types, the sum of average seek, rotational latency, and 8-KB transfer times is close to 20ms. This yields random-access throughput of about 50 IOPS on a single-drive basis. This means that factors other than raw performance--for instance, cost, interface, and controller issues--should be key determinants in disk-drive choices for random-access applications like NFS.

Superior IPI transfer rate (sequential-transfer throughput). IPI drives are readily available with parallel-head transfer capability. By multiplexing the individual data streams of two or three read/write heads, aggregate transfer rates of 6 or 9 MB/s (or more) are possible. (Parallel-head SCSI drives could be easily built, but are not available today.) This high IPI sequential throughput can be important for the data-intensive, large-file applications typically run on supercomputers and minisupercomputers. Such throughput is largely wasted using NFS, as is quantitatively discussed in section 4.1.

Performance Metrics	ST41800K ipi-2	HP C3010 scsi-2	IBM 0663 scsi-2
Command Overhead	1 ms est.	500 us	950 us
Seek (random average, read & write)	11.0 ms	11.5 ms	10.4 ms
Seek (short, track-to-track)	1.7 ms	2.0 ms	0.6 R & 3.0W ms
Rotational Speed (rpm)	5,400 rpm	5,400 rpm	4,316 rpm
Rotational Latency (1/2 revolution)	5.56 ms	5.56 ms	6.95 ms
Seek+Rotate Time (average)	16.5 ms	17.0 ms	17.4 ms
Transfer Rate from Buffer (mb/s)	7.5 mb/s unbuffered	10 mb/s @ 8b	5 mb/s @ 20 mb/s @ 16b
Transfer Rate (raw media mb/s)	7.5 mb/s (dual-head)	4.2-5.3 mb/s	3.0 mb/s
Transfer Rate (sector-to-sector data mb/s)	6.0 mb/s	4.1 mb/s (mean)	2.4 mb/s
Data Transfer Time for 8-kb (ms)	1.4 ms	2.0 ms	3.4 ms
Fraction of Full Rotation for 8-kb Block	12 %	18 %	24 %
Random i/o Rate (8-kb i/os/second)	53 iops	51 iops	46 iops

Table 2: A SCSI-2 and IPI-2 HDA performance comparison. Some comments on the metrics: *Command overhead* for IPI drives is dependent on the controller, and is thus estimated here. The *average seek time* is computed over all possible seek distances, a uniform distribution. *Transfer rate* is a complex number, because the media transfer rates quoted by vendors usually exceed the sector-to-sector data rate by 15-20% and the sustained cylinder-to-cylinder data rate by up to 35%. These rate reductions are caused by format information that must be read, but that is not data; the need to switch between tracks and cylinders, which can take time but transfers no data; and occasional bad spots, requiring time to process remappings. The *sector-to-sector transfer rate* is the real rate at which *data* is transferred to or from a track on the disk. It is computed by dividing the total data bytes in a track by the time of a disk revolution. For the constant-density-recorded HP drive, the middle of the three zones is used for the mean. The 8-KB *data transfer time* uses the realistic sector-to-sector transfer rate and assumes that all 8KB is on the same disk track. The *random I/O rates* indicated are computed by taking the inverse of the (command overhead+ average seek time+ average rotational latency+ data transfer time for 8KB).

3.5 SCSI and IPI Interface Definitions and Operation

Comparing the SCSI and IPI interfaces is a bit like comparing bread and flour. The inquisitive reader has undoubtedly been waiting for *something* to be different, for previous sections have concluded that typical SCSI and IPI HDA performance is essentially identical (unless two IPI heads are multiplexed to double transfer rate). The wait is over: it is the SCSI and IPI interfaces that duel as competitors, not the HDAs. This section examines how the two interfaces work. Section 3.6 will compare their specifications and limits.

The fundamental difference between SCSI and IPI-2 is that SCSI is a *bus* interface, and IPI-2 is a *device* interface. Refer to figure 2.

3.5.1 The SCSI Bus Interface

SCSI, as a bus interface, defines a shared bus that peer devices such as disks and tapes use to conduct and communicate their business using packets. The SCSI-1 bus is 8-bits wide and transfers data at <=5 MB/s synchronously, and 1-2 MB/s asynchronously. There can be 8 devices per SCSI-1 bus and 16 per SCSI-2--addressed 0-7 or 0-15 (only two disks are shown in figure 2). The implementation of the SCSI interface in each device is called the *SCSI controller*, or just the controller. Device 7 (the highest priority)

is usually reserved for host system access, and this host interface is conventionally called the *SCSI host adapter*, not the "host SCSI controller." In figure 2, the Storage Processor has 10 host adapters on a single board. Since the SCSI bus fills the traditional I/O channel role in SCSI storage subsystems, this report often uses the less-precise term "SCSI channel" to refer to the SCSI bus.

The SCSI controller built into SCSI disk drives is frequently called "intelligent." This is because the controller responds to very high-level, almost un-disk-like commands. For example, *read (logical block number L, number of sectors S)* is a SCSI command to read *S* 512B sectors of data from the disk starting at 512B-block *L*. Note that no disk-style cylinder, track, or sector addresses are given--just a linear block number. Also, multiple blocks (sectors) can be conveniently read--even if they cross track or cylinder boundaries, or include remapped bad spots. In a SCSI disk drive, the controller imposes a simple linear addressing scheme on top of complex physically-addressed media. The controller also transparently deals with issues of error detection and correction, bad-sector remapping, error logging, and data buffering (see below). This intelligence is integrated *directly* into the drive by the disk vendor, who presumably best knows how to deal with these issues--in the lowest-overhead manner--for his own HDA.

SCSI is not a fixed master-slave bus. Each device can assume control as the bus master and transact business in a request-reply fashion with any other device on the bus--although usually to the host, through the host adapter. Consequently, SCSI device communication is interrupt driven, not polled, which reduces bus contention. Because the single SCSI bus is used to pass both control and data information, the SCSI bus is the storage channel (defined in section 1.1) for all attached devices.

The SCSI bus permits substantial device concurrency, since, for instance:

The host can connect to a tape drive and give it a command to read data. The tape then disconnects from the bus--freeing the host--and continues executing the read command and filling its internal buffers.

While the tape is at work (and off the bus), the host can *immediately* connect to another device--say, a disk drive--and give it a command to write data (including sending the data with the command). The host can now disconnect from the disk.

While both the tape and disk are at work (and still off the bus), the host can request another disk to read data without waiting for the previous disk or tape. Again, once the disk has received the read command, it can disconnect from the host and actually perform the read into internal buffers.

Now, when the tape has actually read its data, it becomes bus master, reconnects to the host adapter, and transmits the data requested by the host over the SCSI channel.

While the tape is sending its data, assume that the disk finishes its read operation. The disk tries to connect to the bus, but finds it busy with tape traffic. The disk non-intrusively monitors the bus until the tape finishes, then takes its turn as bus master and sends its data to the host adapter.

As the disk transfers the requested data to the host, many SCSI controllers will automatically read the remainder of the current disk track into their internal buffers. This is called *read ahead*. If the host should soon ask for more data on the same track (e.g., sequentially read the next few sectors), read ahead permits the drive to quickly respond from its buffers rather than from the (slower) disk itself. If the host asks for other data (not read ahead), read-ahead operations can be instantly aborted and the requested data sought without delay. Some SCSI disks have *multisegment caching* that stores noncontiguous tracks or sectors, corresponding to different disk-traffic localities (i.e., user working sets).

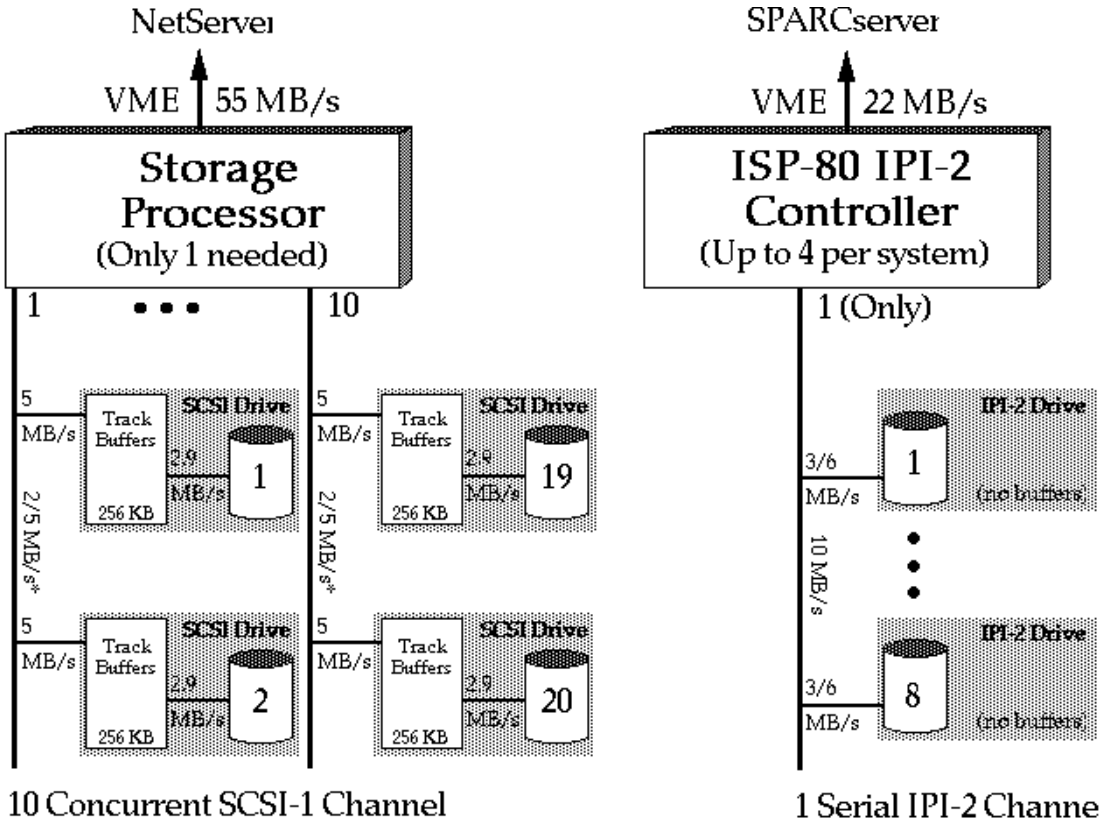

Figure 2: Typical SCSI disk array and IPI-2 disk subsystem architectures. The storage subsystems of Auspex NetServers and Sun SPARCservers are illustrated as concrete examples of each architecture. The transfer rates in the figure correspond to current-generation 5.25-inch disks, not the forthcoming drives in tables 1-2. The Storage Processor has a maximum of 20 drives--2 per SCSI channel--and each drive can read and write concurrently as described in section 3.5.1. Do not construe the term *disk array* too elaborately: it simply refers to a disk subsystem organization that has high drive packing density (volumetric efficiency) with the drives attached to multiple parallel channels (typically >=5). (For the curious, Katz and Patterson define disk arrays in detail [Patterson88].) The NetServer Storage Processor is an intelligent, 10-SCSI-channel controller that supports 10^{x} 5 MB/s= 50 MB/s aggregate channel rate. (The asterisk (*) indicates 5 MB/s synchronous or 2 MB/s asynchronous transfer rates.) SPARCserver IPI subsystems support up to 8 disks per controller, but each controller has strictly serial read and write access to each disk, as described in section 3.5.2. There is a current limit of 4 IPI controllers per SPARCserver system, with a single IPI channel per controller, although other vendors support more controllers or more channels per controller. The Sun ISP-80 controllers handle a 10 MB/s channel transfer rate and are optimized (with controller buffering) for sequential read-ahead. Despite this sequential transfer advantage, section 4.3 discusses empirical testing that confirms that the limited drive concurrency of this style of IPI subsystem greatly restricts NFS (random-access) throughput.

Notice that the intelligent nature of the SCSI interface permits and even encourages a complete decoupling of a device's operation requests, operation execution, and operation replies. Further, requests and replies to different SCSI devices can be freely intermixed in time--device access is not *serialized* (funneled down to a single active device), and device execution is completely autonomous and overlapped. For electromechanical devices like disks and tapes, buffering in the device's controller matches the millisecond speeds of the recording hardware to the microsecond speeds of the SCSI bus--analogous to a freeway on-ramp. This thousand-fold impedance matching between SCSI devices and the SCSI bus is a great advantage: it works smoothly, it's economical since it is built into the device, and it enables scalable device parallelism. As we will see in section 4.4, arrays of SCSI disks can easily deliver this random-access throughput at the system level--ideal for NFS service.

3.5.2 The IPI-2 Device Interface

IPI-2 is a 16-bit parallel synchronous data interface with five separate control signals for disk device control. Up to eight IPI devices can be attached to a single 10-MB/s IPI *channel*, addressed 0-7. IPI is a *master-slave* channel, unlike SCSI, which has a peer-to-peer protocol. As a result, an IPI channel attaches disks to a host system through a distinguished *disk controller*. This controller is the channel master and needs no device address of its own since it controls slave devices (e.g., disk drives 0-7).

Compared to SCSI, the IPI interface is an unintelligent interface. IPI drives do not offer a bad-spot-free, linear-address disk model to the IPI channel. Rather, the IPI controller assumes these tasks, and uses the many data and control signals of the IPI channel to implement a higher-level model to the host. In essence, while SCSI puts the intelligence in each drive (in the drive's SCSI controller), IPI pulls the intelligence out of the drive and puts it up on the disk channel controller, where intelligent functions are shared among the up to eight drives per channel. While this may appear to have some economic benefit, in practice the low volumes of IPI drives--and even lower volumes of frequently host-vendor-specific controllers--keep IPI subsystem prices well above similarly-configured SCSI

drive prices.

Pushing IPI intelligence back to the controller can have serious performance disadvantages. Typical IPI controllers can command disk drives to perform concurrent seeks. But the IPI channel performs only strictly serialized data transfers. Furthermore, IPI drives do not (yet) have individual read-ahead track buffers--buffering is pushed back to the shared controller. The result is that IPI drives on the same channel end up waiting for each other--each waiting to initiate raw transfers to the controller.

Consider this typical sequence of events for the IPI subsystem in figure 2:

The IPI controller commands one disk to seek to a specified track.

The controller immediately commands another disk on the same channel to a different track.

The controller waits for the two disks until one of them is ready to transfer data at the desired sector on the target track and interrupts the controller. (The implications of so-called *zero-latency reads* (ZLR) are not considered here. In a ZLR transfer, data movement can start from the middle sectors--not just the first--of a multisector transfer. The remaining data--i.e., the initial sectors requested--are read later when the disk head arrives there. The piecemeal transfer is spliced together by the controller before being passed, contiguous and intact, to the host. ZLR is an advantage for large, multi-track (>50 KB) transfers, but is of marginal help for small NFS-style 8-KB block transfers.)

The controller accepts raw data from the ready drive, processing the preamble and postamble, performing error detection (typically by computing a Reed-Solomon ECC code), buffering, etc. The controller is fully occupied servicing this single drive for the duration of the transfer.

During the above drive's transfer, the second drive becomes ready. But because the controller and channel are already busy with a transfer, the second drive waits--and the read-write heads spin past the desired data sectors. This is where SCSI has a key advantage: a SCSI drive that finds its channel busy never waits, because the SCSI HDA can transfer the target data into local-to-the-drive buffers. An IPI drive, on the contrary, must rotate completely around--wasting 11-16 milliseconds, depending on disk angular velocity (RPM)--before it can begin the transfer again. This is an enormous penalty. It lets us predict that *adding more drives to an IPI channel will not linearly increase random-access throughput*--a counterintuitive hypothesis that is tested in section 4.3.

After spinning around, the second drive executes its transfer, as long as the first drive is finished and no other drive has grabbed the IPI channel in the single-rotation interim. The necessarily *serialized* access to the IPI channel for data transfers severely limits IPI's random-access throughput. This is the classic and well-known *channel contention problem* in the mainframe world, where disk drives must be coupled with a balanced number of channels to maintain unhindered storage-to-host throughput.

Finally, consider the effect of IPI read-ahead. If an IPI controller has been programmed to perform track (or greater) read-ahead--typical of sequential-transfer environments--other waiting disks will wait even longer. Furthermore, if the read-ahead data is never used because of random-access, all that extra waiting time is *really* wasted. Contrast this with SCSI, where read-ahead occurs inside the drive, creating neither channel contention nor unnecessary waiting.

Several methods are available to widen IPI's serial-channel bottleneck. The easiest remedy is to use only one IPI disk per controller (which is how SCSI operates already). While computer vendors often benchmark their IPI disk subsystems using this method, it is far too expensive for most customers. A second remedy is to build multiple storage channels into one controller. Unfortunately, it takes substantial hardware real estate to cope with the low-level IPI-2 device interface, resulting in a limit on how many channels can be packed onto a printed-circuit board. Still, some clever vendors are building two-channel IPI controllers for the Unix market. But this does not match the dense channel packing possible with SCSI, where 5- and even 10-SCSI-channel host adapters exist.

Finally, remember that the serial-channel bottleneck mainly applies to random I/O patterns. One of IPI's *raisons d'être* is quick, massive, single-drive sequential transfers using parallel heads. IPI will do this well even with multiple drives per IPI channel because the transfer time--for suitably long transfers--outweighs the pretransfer positioning time.

3.6 SCSI and IPI Interface Comparison

The previous section defined the SCSI-bus and IPI-device interfaces. It continued to describe some performance-critical operational issues of disk-to-host transfers. This section returns to basics by summarizing and comparing SCSI and IPI interface specifications. See table3.

Attribute	ipi-2	scsi-1	scsi-2
Device or Bus Interface	Device	Bus	Bus
Arbitration Scheme	Master-slave	Peer-Peer	Peer-Peer
Track Buffering	No	Yes	Yes
Autonomous Device Operation	No	Yes	Yes
Multiple Transfers per Bus/Channel	No	Yes	Yes
Multiple Command Queueing	Yes*	No	Yes
Read-ahead Possible	Yes*	Yes	Yes
Zero-latency Reads (zlr) Likely	Yes*	No	Yes
Bus/Channel Width (bits)	16b	8b	8b, 16b, 32b
Max Bus/Channel Bandwidth (mb/s)	10 mb/s	5 mb/s	10, 20, 40 mb/s

Device to Channel to Controller Parity?	Yes	Yes	Yes
Device to Channel to Controller ecc?	Yes	No	No

Table 3: A SCSI-1, SCSI-2, and IPI-2 interface comparison. Some comments on the attributes: Section 3.5 defines *device and bus interfaces* and discusses SCSI's and IPI's *arbitration schemes*. *Zero-latency reads* are defined in section 3.5.2. All IPI attributes marked with an asterisk (*) are optionally implemented in the IPI controller but are not available in the drives themselves.

Reviewing table 3, there are six key attributes that distinguish SCSI from IPI:

Multiple transfers per bus/channel. Because SCSI devices are buffered and intelligent, physical device data transfers can proceed in parallel, decoupled from the channel. IPI transfers, as we have seen, are direct-coupled, serial, and will often miss their optimal starting points when there is more than one drive per (busy) channel.

Multiple command queueing. With both IPI and SCSI-1, command queueing (and sorting) is left as a controller function, where it is usually performed in high-performance systems. With SCSI-2, queueing can also occur within the SCSI-2 device's own controller. As well as aiding device optimization, this queueing has the added benefit of reducing bus contention by eliminating unnecessary bus handshaking when multiple commands are sent together using *command linking*.

Read-ahead possible. With its internal buffering, read-ahead is natural and straightforward in SCSI devices. For disks, read-ahead is usually done on a track basis--that is, as a disk head sweeps an entire track, all the data passing under it is read into the local buffer. Lacking device-level buffering, IPI drives do not perform read-ahead themselves, although a high-performance controller will often do so. Unconstrained controller-based IPI read-ahead can have serious negative performance implications in random-access environments like NFS, as discussed in section 3.5.2.

SCSI read-ahead better. With its per-drive internal buffering, SCSI read-ahead can be more effective than IPI's controller-level buffering. This is because IPI-controller-driven read-ahead easily results in unnecessary disk delays and channel contention.

Zero-latency reads likely. SCSI-1 does not usually perform ZLR. Many SCSI-2 drives do. As we have seen, IPI disks do not perform either reads or zero-latency reads themselves. In an IPI disk subsystem, then, ZLR is usually implemented in the controller.

IPI has better end-to-end data integrity. Since IPI controllers perform ECC computations on raw sectors, full ECC protection is provided from the media to the host controller. SCSI typically uses only ninth-bit byte-parity protection from the drive's buffer to the host. This gives IPI a distinct data-integrity edge in data-sensitive or long-cabled environments.

3.7 A SCSI-1 Drive Laboratory Performance Analysis

Examining disk specifications is no substitute for an empirical performance analysis. Charts 2a and 2b present the results of extensive SCSI testing that measures the numbers discussed below [Cheng90, Starr90]. Hewlett-Packard 97548 663-MB SCSI-1 drives [HP89] were used. These drives--lower in both performance and capacity than the HP C3010 drives in tables 1 and 2 [HP92]--execute 2-MB/s asynchronous SCSI bus transfers and either 4- or 5-MB/s synchronous transfers. The host adapter was a 10- channel Auspex Storage Processor attached to special instrumentation.

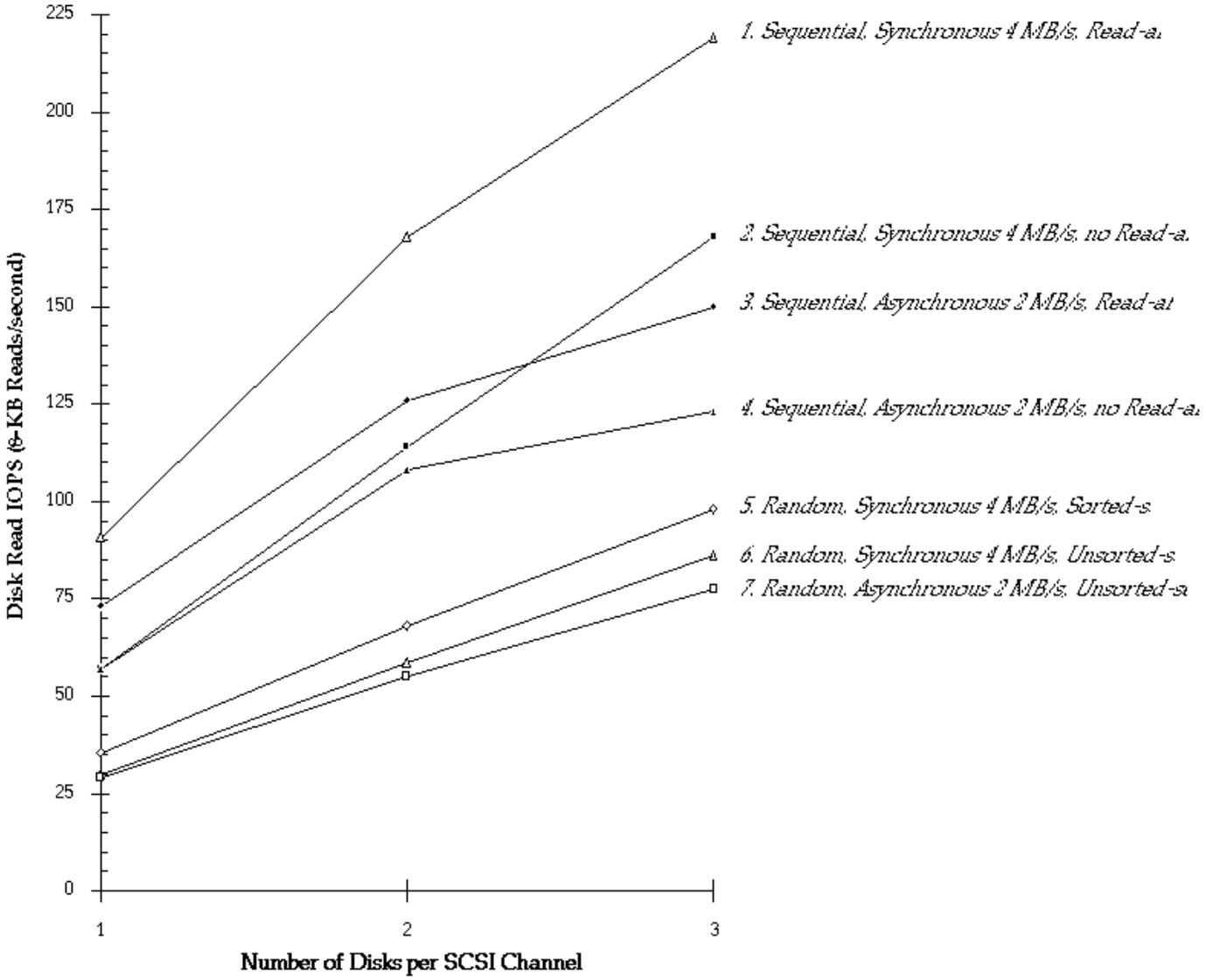

Chart 2a: A test-bench evaluation of SCSI-1 disk drive read performance on 1-3 drives. In this graphical analysis of the HP97548 SCSI drive, curves 1-7 investigate the influence of five parameters on read performance: (1)synchronous and asynchronous bus transfers, (2)random and sequential access patterns, (3)sorted and unsorted seek queues, (4)read-ahead enabled and disabled, and (5)multiplicity of drives per SCSI channel. In particular, observe that SCSI random-access throughput scales almost linearly as more drives are added to the same SCSI channel (curves 5-7). As we will discover in section 4.3, normal IPI subsystems like the one in figure 2 exhibit much worse behavior. More detailed conclusions are discussed in the text. Note that the service time for a disk operation in milliseconds is simply the inverse of its operation rate in IOPS (which is plotted on the Y-axis, above).

Here are some important conclusions from the SCSI-1 evaluation in chart 2a:

Multiple drives on one SCSI bus perform well. For random access there is only a 5% fall-off in throughput linearity with three synchronous drives--i.e., three drives on one SCSI channel perform *almost* three times as many random I/Os (curves 5&6). For sequential access, where SCSI bus data transfer time is larger, three-drive sequential-synchronous throughput linearity decreases 20% with read-ahead (curve 1) but only 1% with no read-ahead (curve2).

The cost of random access (over sequential access) is 16 ms, or exactly the average seek time of the HP 97548 disk drive--the expected result (curves 2&6, 4&7).

Sorting seeks into elevator queues works well, reducing random-access service time from 33 ms to 28 ms, or 15%, for a single synchronous drive (curves 5&6). Conversely, sorting seeks increases random-access throughput from 30 read IOPS to 36 read IOPS on one drive, and from 86 to 98 read IOPS on three drives.

Read-ahead is crucial for excellent sequential throughput. Using only one drive per channel, sequential synchronous throughput increased from 467 KB/s without read-ahead to 745 KB/s (60% more) with read-ahead of sequential 8-KB blocks (curves 1&2). On a per-operation basis, read-ahead drops a read's response time by about 7 ms, which is nearly the average rotational delay. So *not* reading ahead requires rotating around again, as expected.

For random disk transfers, asynchronous SCSI transfers are only slightly slower than synchronous ones. On a single disk, asynchronous random-access reads take an extra 0.9 ms (3%) of transfer time per 16-sector 8-KB block (curves 6&7). On three disks, the added time rises to 3.8 ms (11%) per drive.

For large sequential disk transfers, asynchronous SCSI transfers are slower than synchronous ones. On a single disk, asynchronous sequential transfers take an extra 3.1 ms (29%) of transfer time per 16-sector 8-KB block (curves 1&3). This corresponds to a drop in sequential throughput from 745 to 598 KB/s. On three disks, the asynchronous slowdown is pronounced, taking 6.3 ms (46%) longer per drive.

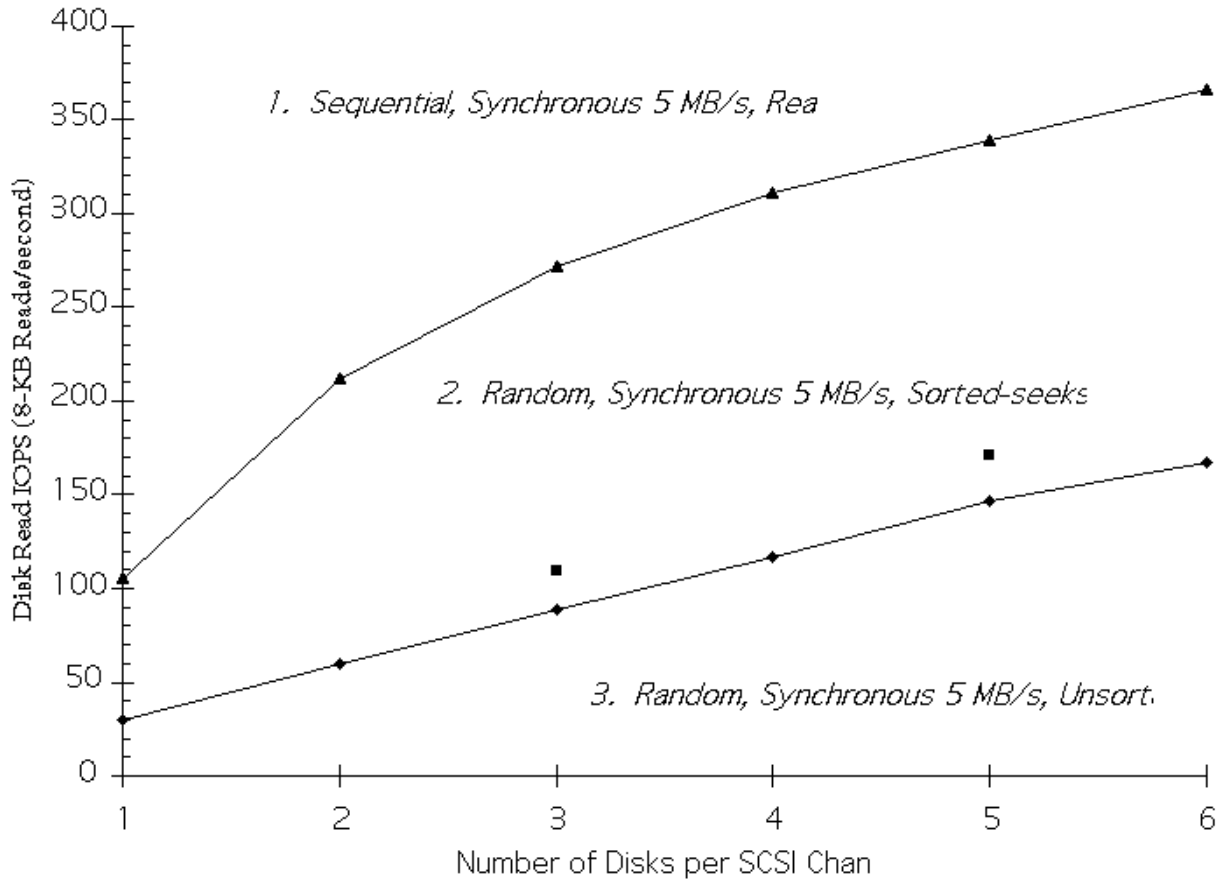

Chart 2b: A test-bench evaluation of SCSI-1 channel performance on 1-6 drives. This graphical analysis of the HP97548 drive studies the effect of many drives per controller on random and sequential read performance. Random-access throughput--for both sorted and unsorted seeks (curves 2&3)--scales linearly to 5 drives with a minor decrease at6. This is remarkable scalability. Sequential throughput for 8-KB blocks (curve 1) tapers off starting with the third drive due to SCSI bus contention in the bandwidth-intensive multi-drive sequential test.

Chart 2b studies read behavior as a function of up to six drives on one SCSI channel. The test environment is similar, but not identical to, that of chart 2a. One notable difference is that the later-model HP97548 drives used for chart 2b performed synchronous transfers at 5MB/s, not 4MB/s as in 2a. Chart 2b confirms two trends for 4-6 drives per channel that were observed above for 1-3:

Random-access throughput scales linearly for the small, 8-KB blocks measured. This is true for both sorted (curve 2) and unsorted (curve 3) seeks. Sorting gives the same performance gain as in chart2a.

Sequential throughput rapidly falls off linear when multiple, sequential-reading SCSI drives jointly contend for the 5 MB/s SCSI-1 bus (curve 1). This is because each drive is capable of bursting data at 5 MB/s from its read-ahead buffer, and *all* the drives' buffers are indeed filled and ready to transfer in this sequential-read test.

3.8 SCSI and IPI Comparison Summary

Section 3 has examined SCSI and IPI from specifications to testing, from HDA to interface, from price-capacity to price-performance, and from the cursory to the detailed. Table 4 summarizes these findings for the disk subsystem organizations shown in

figure 2.

Attribute Advantage Ë	ipi-2 Disk & Ctlr	scsi Array
Transfer Rate Unstriped (mb/s)	4	
Transfer Rate Striped (mb/s)	4	4
Disk Drive Concurrency (disk operations/second)		4
Disk Channel Utilization (disk operations/second)		4
Storage Controller or Processor (disk operations/s)		4
Price-Capacity ($/mb)		4
Volumetric Efficiency (gb/m3)		4
Reliability (disk mtbf)	4	4

Table 4: A summary comparison of SCSI-array and IPI disk subsystems. Check marks in this table are a consequence of conclusions reached in earlier sections and tables.

Some comments on table 4:

SCSI remains the volumetric-efficiency leader because of the new, 1-GB 3.5-inch SCSI drives.

Since disk striping (interleaving) can be applied to both multicontroller and multichannel IPI subsystems and also to multi-SCSI-bus subsystems, both IPI and SCSI can attain >>10 MB/s aggregate transfer rates. For those unfamiliar with disk striping, see Patterson and Katz's RAID paper [Patterson88].

At MTBF ratings of >=250K hours for both IPI and SCSI because of their like (or identical) HDAs, both drive types are extremely reliable.

4 SCSI and IPI Disk Subsystem Performance Comparison

4.1 The Anatomy of an NFS Operation--Measuring Transfer Rate & NFS Throughput

Figure 3 dissects the anatomy of an NFS *Read* operation from when the *Read* request is initiated on a client workstation until the *Read*'s 8 KB of data returns to the client from a disk. The resulting timing analysis is done in the context of an Auspex NetServer [Hitz90, Nelson91], a functional asymmetric multiprocessor NFS file server that uses SCSI disk arrays for its NFS storage subsystem (figure 2). The goal of this timing analysis, presented in detail below, is to clearly demonstrate that the actual data-transfer time of an 8-KB NFS *disk* read is such a small part of overall *system-level* NFS *Read* time that the disk's transfer rate has negligible performance impact.

The analysis in figure 3 explicitly ignores all system-, controller-, and drive-level caching optimizations, both for reads and writes. This is necessary to study the influence of actual disk transfer rate: If a disk is not involved in an I/O operation because a cache-hit stepped conveniently into the way--at any level in the system--then transfer rate doesn't matter at all. Caching is indeed crucial for system performance and is used in all real systems. But this study examines what happens when there is a data cache miss on a read (or write).

Finally, the NetServer timings shown for the NS5000 Ethernet and File Processors in figure 3 are from a two-year-old software release. As such, because of substantial software tuning and hardware upgrades to these processors, the timings shown are probably 40-60% high (the NS5500--with new processors--has about 80% higher maximum NFS throughput than the NS5000). However, since individual NS5500 I/O processors have not yet been measured, actual--if older--NS5000 times are used.

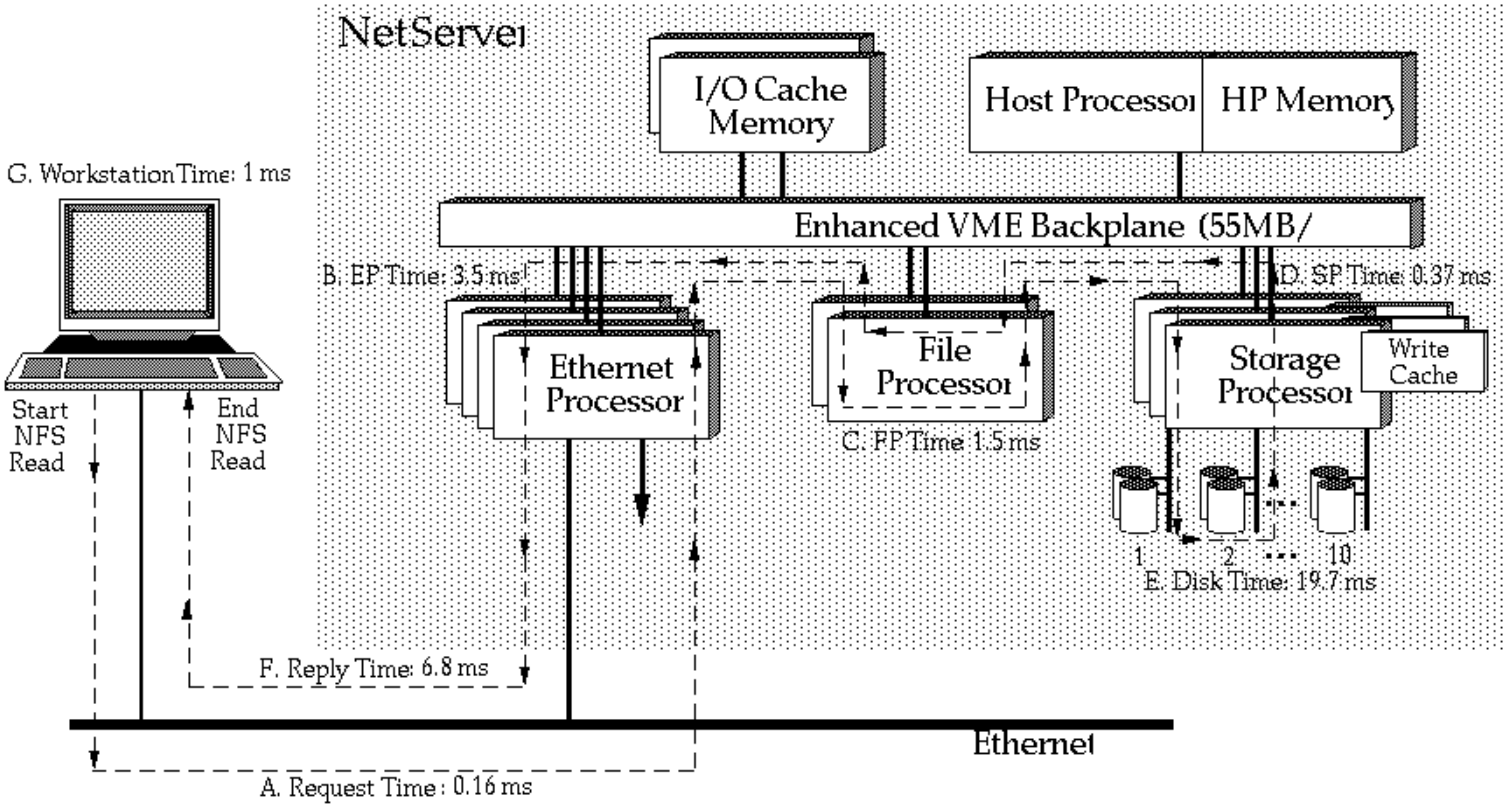

Figure 3: The anatomy of an uncached NFS *Read* operation. The indicated 19.7 ms of total disk time dominates the overall 32.0ms required for the complete client-to-server-to-client NFS *Read*. However, the disk data-transfer time for an 8-KB NFS block is only 2.0ms of the 19.7 ms total disk time, or just 6% of the entire 32.0ms client-to-server-to-client NFS *Read* time. This 6% transfer-rate contribution is not significant, and becomes *less* so as disk transfer rate increases further beyond 5.0 MB/s. Client workstation time is not included in the total time. Detailed discussion and further conclusions appear in the text. The timings indicated for the NetServer are not estimates but were measured with software monitors and logic analyzers on an NS5000 running Auspex software release 1.1. The disk timings are based on current-generation HP C3010 SCSI-2 drives [HP92] of tables 1-2. Ethernet timings are for transmission times assuming no access deference or collisions.

Let's examine each step in the timing path of figure 3's non-cached *Read* operation, and make some key observations along the way:

A *Ethernet request time: 0.16ms, inbound to server only*. This is the Ethernet transmission time required to send a single IP packet (UDP datagram) that contains the NFS *Read* remote procedure call (RPC). This is a best-possible time that assumes no Ethernet contention or collisions. *Read* requests are about 100bytes long and contain IP, UDP, and RPC headers, NFS file handle and offset information, and user identification and (optionally) authentication information.

B *Ethernet Processor (EP) time: 3.5ms, inbound and outbound*. This is the complete NFS protocol processing time required for the *Read* (or of many other NFS operations). In the NetServer, protocol processing includes Ethernet, IP, UDP, RPC, XDR, and NFS processing time. It also incorporates preliminary local file system (LFS [Schwartz90]) processing time (equivalent to much of what happens in a conventional NFS server's NFSDs) and block-transfer DMA time between I/O cache memory and the Ethernet Processor.

Observe that NFS protocol processing time (EP time) dominates all other NFS file server software overhead--the EP's 3.5ms exceeds the FP's 1.5ms or the SP's 370us. (This is why NetServers integrate protocol processing directly into the network interface--for network scalability with very low response times.)

C *File Processor (FP) time: 1.5ms, inbound and outbound*. This includes all file-system processing time, including cache management. In the NetServer, it also includes the time for kernel-level message-passing communication [Hitz90] between the Ethernet, File, and Storage Processors, which takes about 100us per one-way message.

There are two request-reply message pairs exchanged for an NFS disk *Read* (four one-way messages): a *Read* request from the EP to the FP, an 8-KB disk-read request from the FP to the SP, a trivial "done" reply from the SP back to the EP, and a *Read*-data reply from the FP back to the EP.

D *Storage Processor (SP) time: 370us, inbound and outbound*. This includes all Storage Processor kernel message-processing time (230us), elevator queuing software overhead (67us), SCSI bus arbitration (1us), and SP SCSI command overhead (70us). Put in conventional SCSI terms, the NetServer's SCSI host adapter overhead is 138us, an exceptionally low figure since many host adapters have overheads >1 *milli*second.

E *Disk time: 19.7 ms total*. This total disk-read time for a randomly-accessed 8-KB NFS data block includes six components:

1. *70us of SCSI device selection and command transfer* (overlapped with SP overhead in D above).

2. *500us of SCSI drive command decode and processing*;

3. *11.5ms average seek time*;

4. *5.56ms average rotational latency*; and

5. *2.0ms data-transfer time* for 16 sectors (8 KB), all assumed to be on the same disk track (a realistic best case).

6. *260us of SCSI cleanup and handshake* (some overlapped with the SP in D).

Observe that this 19.7 ms of disk time overwhelms all other times, and that transfer time is a small fraction--10%--of disk time. We will study the influence of increasing this transfer rate below in table 5. Note that items 1-6 sum to 19.89 ms; the 19.7ms total excludes the overlaps mentioned in items 1 and 6. The drive-specific timings for items 2-5 above are for the HPC3010 2.0-GB SCSI-2 drive (table 2) [HP92].

F *Ethernet reply time: 6.8ms, outbound from server only*. This is the Ethernet transmission time required to send the 6 IP packets that contain the server's reply to the client's NFS *Read* RPC. This is again a best-possible time that assumes no Ethernet deference or collisions. *Read* replies are about 8,500bytes long and contain IP, UDP, and RPC headers, NFS file information, and the actual 8 KB (8,192bytes) of file data. Although the *Read* RPC reply is a single UDP datagram, it is too large to be encapsulated in a single IP packet. Thus it is fragmented into 5 full size (1,512-byte) IP packets and a final, sixth partial packet.

Observe that at nearly 7ms, Ethernet transmission time is a significant factor (22%) of overall *Read* time. On an Ethernet experiencing moderate loads (>30%), deferred transmissions and collisions can easily add 10-30ms to this 7ms of unhindered transmission time [Boggs88]. Moral: don't overload your Ethernets or both throughput and NFS response times will suffer.

G *Client workstation time: 1ms total*. Client workstation time is highly variable, and depends not only on workstation CPU power, but also on the capabilities of the workstation's Ethernet interface, software driver, and Ethernet-to-memory I/O datapath. As a consequence, this analysis assumes an optimistic 1ms total client NFS processing time. This is not obtained in previous generation 10-MIPS CISC workstations, but may well be in contemporary 100-MIPS RISC workstations.

[[sigma]] *Total time: 32.0ms*, not including client workstation time.

Table 5 shows the performance impact of increasing disk drive transfer rate on the system measured in figure 3. Note that a 9-MB/s SCSI drive would only improve NFS read performance by 3%, and a 12-MB/s drive by only 4%. But even these improvements would be worthwhile *only on SCSI drives*: if an IPI subsystem were used instead of a SCSI disk array, random-access throughput would suffer greatly because of IPI channel-access contention, as discussed in section 3.5.2 and shown empirically later in section 4.3.

Raw Transfer Rate	8-kb Transfer Time	Total nfs Read Time	[[Delta]] Performance
4.86 mb/s	2.0 ms	32.0 ms	Baseline
6.0 mb/s	1.6 ms	31.6 ms	+1 %
9.0 mb/s	1.1 ms	31.1 ms	+3 %
12.0 mb/s	0.8 ms	30.8 ms	+4 %

Table 5: The influence of disk data-transfer rate on system-level NFS performance. The 4% gain--i.e., reduction in total *Read* time--from using even 12-MB/s drives is negligible. The transfer times shown are the data-transfer times for a 16-sector, 8-KB block. The total NFS *Read* time consists of a fixed, 30.0ms portion (the sum of all but the data-transfer and client-workstation times in figure 3), to which the data-transfer time is added.

4.2 NFS Benchmarking, NFS IOPS, and the NFS Operation Mixture

Some simple fundamentals of NFS benchmark methodology are needed for the following sections that examine disk subsystem performance in the context of system-level NFS testing. A complete discussion of accurate NFS benchmarking is a complex subject beyond the scope of this paper. This section briefly repeats some definitions and descriptions from Horton's NFS performance study [Horton90].

NFS IOPS--I/O operations per second--are an NFS-specific measure of NFS load. NFS IOPS are computed by dividing the total number of NFS operations executed by a server (or client) in a given time interval by that time itself. These NFS throughput results are plotted as a function of average response times (e.g., chart 3). Each type of NFS operation--*Read, Write, Create, Remove, Lookup, GetAttr*, *SetAttr*, etc.--is counted as a single operation in the total operation count. For reference, a single DECstation 3100 or SPARCstation1 performing moderate NFS I/O (e.g., compiling remote sources) generates a load of 10-20 IOPS to an NFS server, depending on local disk and memory configuration.

nfs Operation Mix	GetAttr	SetAttr	Lookup	ReadLink	ReadDir	Read	Write	Create	Remove	FSStat	Total
nhfsstone	13%	1%	34%	8%	3%	22%	15%	2%	1%	1%	100

Default											%
Software Company	14%	1%	59%	2%	9%	8%	4%	1%	1%	1%	100 %

Table 6: Two distinct NFS operation mixtures. NHFSStone's default operation mix is widely used [Lyon89]. It is moderately I/O intense and characterizes a diskless-workstation software development environment. On the NHFSStone mix, *Read* and *Write* combined constitute 37%, or just over one-third of all operations. Directory operations (*Lookup, GetAttr, SetAttr, ReadLink*, and *ReadDir*) constitute 59%, or approach two-thirds. The Software Company's mix is particular to chart 4. It has an extremely high *Lookup* fraction and unusually low *Read* and *Write* ratios--a lightweight mix reflecting a low-use dataless software development environment with very deep directory trees. The remaining seven (of the seventeen total) NFS operations unrepresented in this table are executed so infrequently as to be negligible.

The benchmark workloads used in this study consist of synthetic NFS traffic. This traffic is carefully generated to correspond to average NFS operation mixtures actually encountered on servers handling client workstations executing software development applications. The server disk traffic resulting from these synthetic NFS workloads is distributed equally across initially-empty file systems on all tested disks. The two sets of NFS operation mixes used for this paper's tests appear in table 6.

It is often useful to derive a single NFS throughput figure-of-merit from the two-dimensional throughput versus response-time curves shown in the accompanying charts. In this report, we measure a server's NFS throughput (in NFS IOPS) at either 50 or 70 milliseconds response time. The 50- or 70-ms response time cutoff is a comfortable maximum: workstation users prefer 20-40ms and notice a marked slowdown at >=100ms.

4.3 Why IPI is Poor at NFS--Measuring Multiple IPI Drives per Controller

4.3.1 Multiple Drives per Controller without Write Caching

In section 3.5.2 we predicted that adding drives to the same IPI disk channel would less-than-linearly increase random-access throughput. Chart 3 is empirical proof of this counterintuitive hypothesis. For reasons of convenient access, Sun's SPARCserver IPI disk subsystem was used for this test. Similar results are expected from any sequentially-optimized IPI (or SMD) disk subsystem, although the relative ratios would change depending on a particular vendor's implementation.

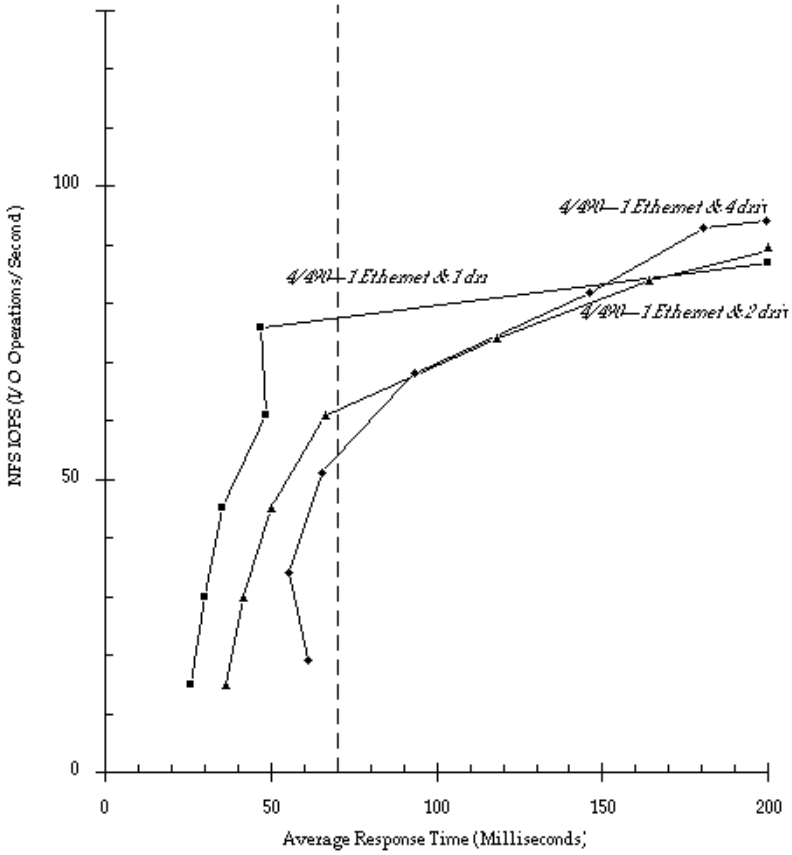

Chart 3: An NFS performance comparison of multiple IPI drives on a single IPI controller *without* write caching. Notice how the 4/490's NFS throughput actually *decreases*--and response times *increase*--as more disks are added to the same IPI channel. This is because of bottlenecked IPI channel access (probably including excessive and unnecessary read-ahead). Both this negative throughput scaling and the saturated-disk, 150-ms throughput crossover are explained in the text. All disk writes performed in this

test are synchronous and not buffered asynchronously through fast stable storage such as a non-volatile RAM cache. This test uses the default NHFSStone NFS mix (table 6). The test configuration was a Sun 4/490 running SunOS 4.0.3 with one ISP-80 IPI-2 controller and 1-4 Seagate 8-inch IPI-2 disks.

This comparison is a system-level test--not an isolated disk subsystem test. The NFS load driving the server originates on workstations, so server network and CPU characteristics will play a part in the results. But from test trial to test trial in chart 3, only the number of disks was varied. Three drive configurations were tested on a 4/490--1, 2, and 4 drives, all on the same IPI controller. No asynchronous write-caching (buffering) was used--NFS *Write* operations are performed synchronously to disk.

Here are the critical observations in chart 3:

With only 1 drive, response times stay between 30-50ms until about 80 NFS IOPS (measured at a response-time threshold of 70ms) and then throughput levels off.

With 2 drives, response times increase to 40-70ms and throughput drops to about 65 NFS IOPS at a 70ms response time. Random-access throughput and response time is worse for 2 drives than 1.

With 4 drives, response times increase further, to 50-100ms, and throughput drops further to about 55 NFS IOPS. Four drives are worse than 2 drives, because there are now 3 drives waiting for the 1 active drive on the IPI channel to finish.

In total overload, at the response time region of 150-200ms, 4-drive throughput exceeds 2-drive throughput, which exceeds 1-drive throughput. This is probably because each disk finally has enough queued work (in the controller) that whenever one drive finishes a data transfer, one of the waiting disks (probably) has queued data under its heads and can begin transferring it immediately. But the high response times of this overload region are unacceptable for most applications.

The negative throughput scaling in chart 3 is worse than expected. In general, a small *positive* but ever-diminishing (i.e., markedly sublinear) throughput *increase* should occur for each extra disk on a single channel. We speculate the tested 4/490 was optimized for "excessive" sequential read ahead, with very detrimental results on this small-block 8-KB NFS test as explained at the end of section 3.5.2. We did not have access to a system with such better behavior when preparing the paper.

These three IPI performance curves show our prediction that IPI disk subsystems do not exhibit good random-access performance unless there is a low drive-to-controller ratio--e.g., 1:1 or 2:1. This is an expensive proposition.

4.3.2 Multiple Drives per Controller with Write Caching

This section continues the previous study of the Sun 4/490 IPI subsystem, this time with system-level write caching (described below) included. While 4 IPI controllers were available in the test configuration, only 1 and 2 drives per controller were tested. However, as chart 4 shows, write buffering does not effectively mask the multiple-drive-per-controller contention problem on the 4/490 under heavy NFS loads. Two configurations of SCSI-array NetServers are included for comparison.

In detail, this section repeats the test of the last section with three major changes:

The IPI disk subsystem is "fronted" by a stable-storage write buffer that permits the disk subsystem to perform writes asynchronously from the system (and the user's) point of view. Asynchronous NFS writes are usually not performed by NFS servers because strict NFS semantics require synchronous writes to maintain a stateless server guarantee. However, a write cache (or write buffer) that can survive crashes--and therefore appear like a zero-latency disk--is completely acceptable [Lyon89]. Since a write cache is a random-access as well as zero-latency device, it should increase the random-access performance--i.e., boost throughput and reduce response time--of (sequentially oriented) IPI subsystems.

Third-party Ethernet controllers that incorporate some NFS protocol processing are used instead of the 4/490's built-in Ethernet interface. This change should not influence the disk subsystem, although it may free some 490 CPU cycles for extra NFS processing.

The NFS operation mix (section 4.2) is *significantly* different and less server-intensive in these tests, so that the total NFS loads measured are *much* higher. Because of the different mixes used in charts 3 and 4 (see table 6), *their throughput curves cannot be compared directly*. (Chart 4 uses benchmark data based on a different mix because write-cached data using the same mix as chart 3 is not available.)

Consider these chart 4 observations:

With only 1 drive per channel (4 total) on the 4/490, NFS response times stay between 20-40ms until about 380 NFS IOPS, and then throughput levels off. These response times are relatively lower than in chart 3--the write cache is working.

With 2 drives per channel (8 total), *once again response times increase, and NFS throughput decreases*. Under heavy NFS load, we speculate that the Prestoserve write cache has a high miss ratio, and that the writes going directly to the disks are immediately stalled because of multiple drive-per-controller contention, as before.

This leads to the hypothesis that write caching will be most effective for low to medium NFS loads. This is tentatively confirmed by the enhanced 490's better performance than the 4-drive NetServer. Write buffering has masked the IPI random-access shortcomings in a small, 4-drive configuration. Because of the zero-latency "disk" access, the 490 response times are shorter than the NetServer. Maximum NFS throughput of both the buffered 4/490 and unbuffered NetServer is similar--within 8%--showing that both systems are probably disk limited as they enter overload.

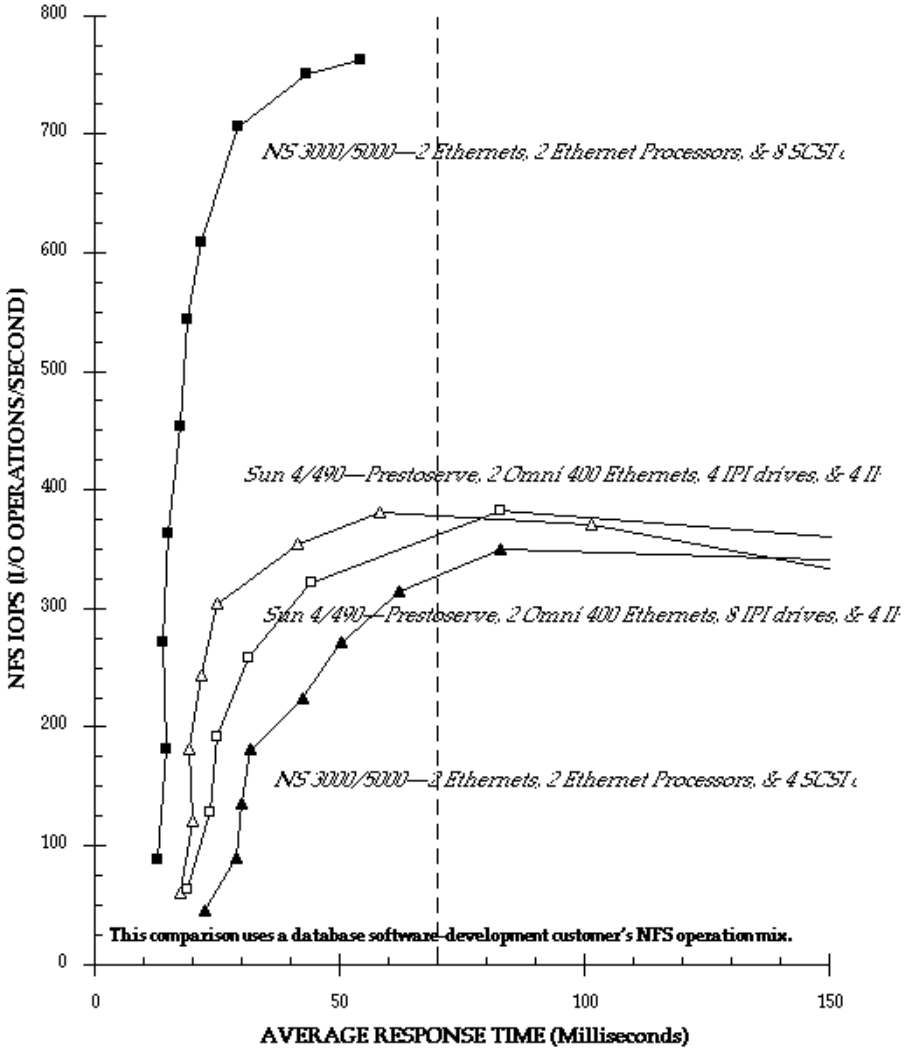

Chart 4: An NFS performance comparison of 1-2 IPI drives on 4 IPI controllers *with* write caching. The 4/490 has 4 controllers and is tested either with 1 drive per controller ("4 drives" in the chart), or 2 per controller ("8 drives"). Notice how the 4/490's NFS throughput still *decreases* as a second disk is added to each IPI channel (from 4 to 8 total). This is because even the Prestoserve write cache cannot totally overcome IPI's serial-channel transfer contention under heavy NFS load. On the other hand, notice that SCSI disk array throughput on the NetServer doubles as disks are doubled. Further conclusions appear in the text. All 4/490 disk writes performed in this test were buffered asynchronously through a Prestoserve write cache. This test uses a software company's specific NFS operation mix (table 6). This mix is so different from that used in chart 3 that *throughput curves from the two charts are not directly comparable*. The test configurations were: a Sun 4/490 running SunOS 4.1 with a Prestoserve 2.0 write cache, 2 Interphase NC400 Network Controllers, 4 ISP-80 IPI-2 controllers, and 4-8 Sun IPI-2 disks; an Auspex NS5000 NetServer running version 1.2 software and SunOS 4.0.3 with an Auspex Storage Processor and 4-8 1-GB SCSI-2 disks.

The NetServer scales well, more than doubling throughput as the number of disks double from 4 to 8. This is in stark contrast to the 490, where moving from 4 to 8 drives decreases throughput. With SCSI disk arrays, write caching is not needed to achieve high overall throughput on normal operation mixes. However, there is merit in using a write cache to reduce latencies at low load levels, or for certain write-intensive applications.

For a further discussion of the systems in chart 4, including NFS performance on the standard NHFSStone operation mix, read Auspex's Benchmark Brief 5 [Auspex90].

In conclusion, IPI disk subsystems will benefit from write caching on most configurations and for low to moderate NFS loads. But our initial premise--that IPI random-access throughput is limited by serial IPI channel contention--has been empirically validated even with system-level write caching. With or without write caching, IPI is poor at heavy-load NFS.

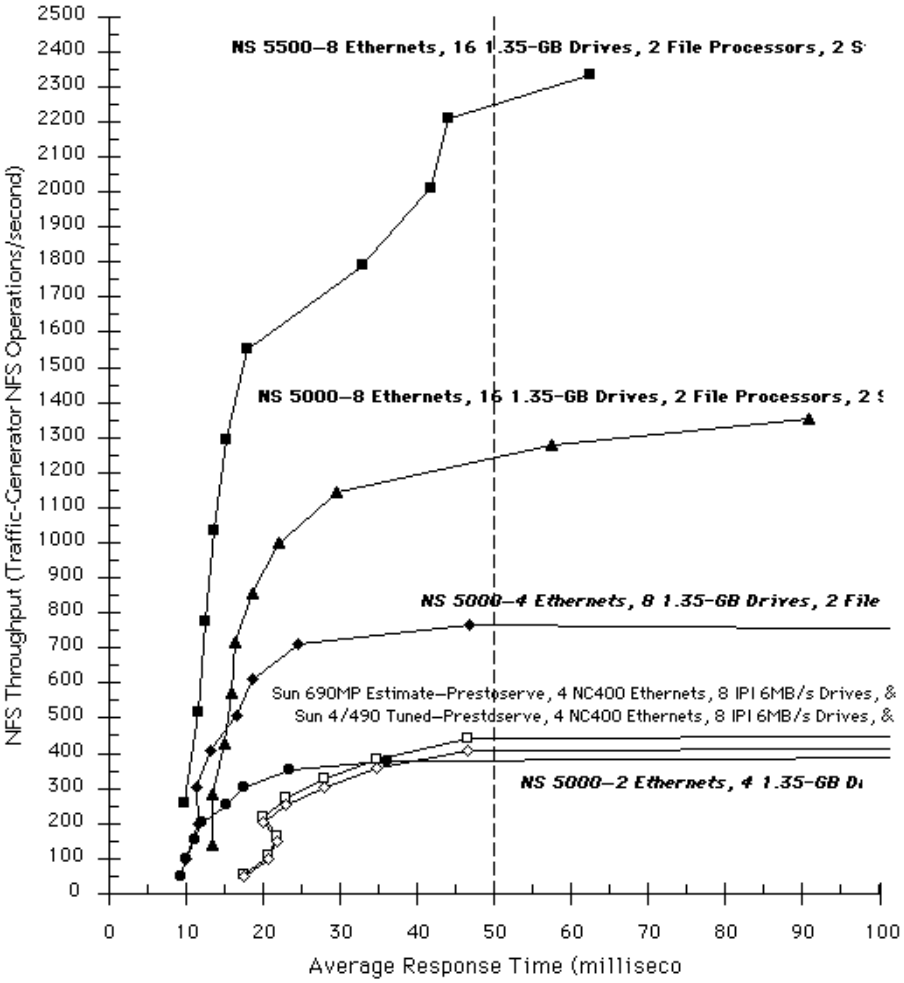

Chart 5: A scalability study of the SCSI disk arrays. In scaling a NetServer (the NS5000 curves) from a 2-Ethernet, 4-drive system, to 4 nets and 8 drives, and 8 nets and 16 drives, the NetServer has a remarkably linear NFS throughput increase of 1.9 (geometric mean) per step. This demonstrates that SCSI disk arrays can provide almost perfectly scalable random-access throughput. The disk array can respond to this scaled-up load because drives are balanced with channels in a 2:1 ratio (and note that chart 2b lets us predict similar results for up to 6:1 ratios). This is possible because SCSI controllers are built into each drive and 10 channels are available on one board. (Notice that to supply scaled-up NFS workload to the server, extra Ethernets are needed as well, although they do not effect the disk subsystem.) Sun SPARCservers 490 and 690MP are shown for NFS comparison in the curves with "hollow" data points (the 690MP estimate is based on Sun's own 8%-improvement figure). This test uses the NHFSstone default NFS operation mix (table 6). The Auspex test configuration was an NS5000 running version 1.4 software and SunOS 4.1 with 2-16 HP 97560 1.35-GB SCSI-2 disks. (See the comment on NS5000 versus NS5500 NFS throughput at the beginning of section 4.1. The NS5500, for which only an 8-net, 16-drive curve is shown, achieves over 2,200 NFS IOPS with similar SCSI scaling.) The SPARCserver test configuration was a 4/490 running SunOS 4.1 with 24 NFSDs, Prestoserve 2.0, NC400 software 1.1, 3 Sun ISP-80 IPI-2 disk controllers, and 8 911- MB 6-MB/s IPI-2 disk drives.

4.4 The Advantages of SCSI Disk Arrays for NFS

This section finishes our empirical disk subsystem investigations by examining the scalability of SCSI disk arrays as more drives are added. As chart 5 shows, it's possible to achieve nearly linear throughput scalability on NFS workloads.

Once again, the benchmarked SCSI disk array is within an Auspex server. For readers familiar with Patterson and Katz's work on Redundant Arrays of Inexpensive Disks (RAID [Patterson88]), Auspex's disk arrays are organized in the simplest fashion: RAID 0 (independent disks) and RAID 1 (dual-disk mirroring)--no striped parity scheme is used. For NFS, which requires small-block random I/Os, RAID 0 and 1 work extremely well. Questions of file-system load balancing across a multiple-disk array are addressed below.

Chart 5 shows the scalability of the NetServer SCSI disk array over 4-16 disks and very high offered NFS loads. Disk array

throughput scales nearly linearly (1.9:2.0, geometric mean) because of the disk-to-channel balance. This is outstanding: There must be no bottlenecks in the entire system to achieve this overall client-to-server NFS throughput linearity.

4.5 Balancing Disk Array Load

Organizing a file server's disk arrays as independent disks (RAID 0) begs the question of how a system administrator balances file system I/O loads across those many disks. In practice, Auspex has found that system administrators have good intuition about their user's requirements.

System	Disk Drive Ë	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Adobe	Queue time (avg. ms)	18	17	27	22	25	24	23	38	14	27	17	27				
	Service time (avg. ms)	20	20	21	20	18	16	19	17	18	19	18	16				
Auspex	Queue time (avg. ms)	7	13	8	15	9	31	22	21	21	17	0	46	34	9	29	18
	Service time (avg. ms)	22	21	21	20	19	17	17	19	16	17	26	22	21	15	22	19
3Com	Queue time (avg. ms)	13	12	6	45	33	92	24	33								
	Service time (avg. ms)	22	15	19	19	20	20	33	18								

Table 7: Sample disk-array queue times and disk service times from operational NetServer systems. In general, each system's individual disk loads are very well balanced (the two exceptions, the bold 0 and 92, are discussed in the text). This indicates that experienced system administrators have good intuition about how to assign swap partitions and file systems to disk drives for even load balancing. In addition, the low disk service times indicate that there is a mild and unexpectedly pleasant degree of locality of disk reference, even for heavy NFS loads. See the text for further discussion.

Table 7 shows the disk activity levels of several NetServer customers. These levels are long-term (several day) averages. They were measured by special software instrumentation inside the Storage Processor. Here is a brief interpretation of this operational data:

The queue and service times are fairly uniform across all disks. This shows that the disks are evenly loaded, and that the respective system administrators have done good jobs. Two major exceptions are shown in bold: Auspex disk 11 has 0 queue time, so it is probably very infrequently used or read-only (i.e., never written; see below). 3Com disk 6 has a deep queue, so it is probably heavily written (e.g., an overloaded swap disk, since service times are so short that the writes are close together on the disk).

Average disk service times are 15-22ms, less than the 29-ms average random-I/O service time for the HP 97548 663-MB SCSI-1 drives used in these systems. This means that most NFS I/Os probably require less than an average seek, and that the NetServer's SCSI-drive read-ahead is finding *some* sequential access traffic to optimize. So it appears that even large NFS servers will see some locality of disk reference, probably because the Unix File System's cylinder group notion is working well.

The average queue time is 10-30ms--about the above-measured time of a single 8-KB I/O. This subtle finding probably indicates that two disk I/Os frequently arrive as pairs. We speculate that they are often writes, because an NFS *Write* is usually transformed into two disk writes--one for the file's inode, and another for the 8-KB data block.

The previous point implies that the NetServer disk array usually encounters NFS *Write* operations, that such writes are usually not themselves queued (i.e., that the disk array is not heavily loaded), and that most NFS *Reads* come from the server's file cache. Other operational statistics indicate that the average read-to-write ratio *at the disk* is 1:3, rising to as much as 1:6 for /tmp file systems and swap partitions. This shows the effectiveness of the primary memory buffer cache, which services most reads without any disk activity. These conclusions are tentative and need more study, but are our best estimates at present.

In Auspex's current server software, two built-in capabilities make it easy for system administrators to balance their disk and file-system loads:

Disk concatenation, striping, and mirroring are fully supported as *virtual partitions* [Cheng91]. Both mirroring (for reads) and striping (for reads and writes) distribute disk traffic across multiple physical disks and eliminate hot-spot disks. The two techniques can be used together to construct a mirrored, striped, maximal-size file system--larger than a physical disk--which puts multiple actuators to work on critical, high-traffic data.

Auspex's servers come with a comprehensive performance monitor [Hitz91] that, in the storage area, shows disk I/O rates, I/O types, and virtual partition access information. Similar information is provided for file systems, including file-operation rates, cache hit rates, and cache age distributions. This makes it easy to spot currently overloaded drives or file systems, and to predict forthcoming overloads.

5 Conclusion

This paper began with two thrusts: show that NFS file server disk traffic has a random--not sequential--access pattern and explore

and explain the complex behavior of modern disk drives. The conclusions from these apparently dissimilar investigations were checked in a performance evaluation of IPI and SCSI disk subsystems in high-performance NFS file servers. The result is that SCSI disk arrays easily outperformed a contemporary, optimized IPI drive subsystem on NFS workloads. The SCSI subsystem also demonstrated nearly linear throughput scalability, a remarkable achievement given its low cost. This result has now moved beyond upstart startup Auspex to traditional Unix vendors such as Data General, Silicon Graphics, and Sun. These companies now offer medium-capacity SCSI disk subsystems--in addition to IPI--for their file server products.

Some further conclusions and comments:

Disk transfer rate is the *least* critical aspect of disk performance for NFS workloads in file servers using conventional extent-based file systems like the UFS Unix file system. Transfer rate is important for data-intensive compute-server applications where a small file is >>100 KB.

High per-disk random I/O rates and fully concurrent disk activities are essential for good NFS performance. SCSI disks have built-in controllers for concurrent, autonomous activity. IPI-2 disks do not.

Disk array organization gives superior random I/O rates and can deliver extreme transfer rates (20-40 MB/s) with striping. Merchant-market SCSI disk-array controllers supporting this functionality can be readily built because of the sophisticated, standard SCSI interface ASICs available. These controllers have 5-10 SCSI channels per controller and are available from OEM vendors like NCR and Array Technologies.

SCSI disk arrays have exceptional scalability, capacity, and performance. In large part, this is because of their built-in controllers and track buffering. But attractive pricing of both the SCSI disks and SCSI interface components--induced by high-volume shipments--is key as well.

These are the reasons why SCSI *is* better than IPI for NFS.

6 Acknowledgments

Many people contributed to the reviews or results of this paper. As well as unnamed Usenix referees, they include: Eric Allman, Dave Anderson, Art Beckman, Larry Copp, Andrew Foss, Raphael Frommer, Bill Horton, Ken Osterberg, John Ousterhout, and John Williams. Our special thanks go to Anderson, Copp, and Osterberg--from either Hewlett-Packard or Seagate--who provided timely, comprehensive, and accurate advance specifications.

7 References

[Auspex90] *The Auspex NS 3000/5000 vs. the Sun/Presto/Omni 4/490 in Customer NFS Benchmark Showdown #1.*
Benchmark Brief 5, Auspex Systems Inc., December 1990.

[Boggs88] David R. Boggs, Jeffrey C. Mogul, and Christopher A. Kent.
Measured Capacity of an Ethernet: Myths and Reality.
Proceedings of the SIGCOMM '88 Symposium on Communications Architectures and Protocols, ACM SIGCOMM, Stanford, CA, August 1988.
Also Digital Western Research Laboratory Research Report 88/4.

[Cheng90] Yu-Ping Cheng.
[SCSI] Disk Drive Performance.
Internal Memorandum, Auspex Systems Inc., 22 June 1990.

[Cheng91] Yu-Ping Cheng, Guy Harris, Bruce Nelson, and William M. Pitts.
Using Virtual Partitions for Disk Storage Administration in Network Servers.
Proceedings of the Sun User Group, San Jose, CA, 4-6 December 1991.
Also Technical Report 8, Auspex Systems Inc., July 1991.

[Hitz90] David Hitz, Guy Harris, James K. Lau, and Allan M. Schwartz.
Using Unix as One Component of a Lightweight Distributed Kernel for Multiprocessor File Servers.
Proceedings of the Winter 1990 USENIX Conference, Washington, DC, 23-26 January 1990.
Also Technical Report 5, Auspex Systems Inc., January 1990.

[Hitz91] David Hitz.
A System Administrator's Performance Monitor for Tuning NFS Network Servers.
Proceedings of the Sun User Group, San Jose, CA, 3-5 December 1990.
Also Technical Report 7, Auspex Systems Inc., May 1991.

[Horton90] William A. Horton and Bruce Nelson.
The Auspex NS5000 and the Sun SPARCserver 490

in a One- and Two-Ethernet NFS Performance Comparison.
Performance Report 2, Auspex Systems Inc., May 1990.

[Howard88] John H. Howard, Michael L. Kazar, Sherri G. Menees, David A. Nichols,
M. Satyanarayanan, Robert N. Sidebotham, and Michael J. West.
Scale and Performance in a Distributed File System.
ACM Transactions on Computer Systems 6(1): 51-81, February 1988.

[HP89] Hewlett-Packard Company.
HP 9754XS/D SCSI *Disk Drives OEM Product Manual*.
Manual Order Number: HP 19530, August 1989.

[HP91] Hewlett-Packard Company.
HP 97556/58/60 5.25-inch SCSI Disk Drives Technical Reference Manual, Edition 2.
HP Part # 5960-0115, Disk Memory Division, Boise, Idaho, June 1991.

[HP92] Hewlett-Packard Company.
HP C3007/09/10 5.25-inch SCSI Disk Drives Technical Reference Manual.
Second edition, HP Part # 5960-8345, Disk Memory Division, Boise, Idaho, April 1992.
The track-to-track seek time specification has been reduced to 2.0ms (from 2.5ms) since the second edition of the manual was printed.

[IBM91] Todd Anderson and Wesley Cook.
0663 Hardware Specification, Preliminary Version 1.2.
IBM Corporation, High Density Storage Products, Reference # 41B/114-2, 4 April 1991.

[IPI2] American National Standards Institute.
*American National Standard for Information Systems--Intelligent Peripheral Interface 2 (*IPI-*2)*.
Documents ANSI X3.129-1986 (physical) and X3.130-1986 (device).

[Kleiman86] Steven R. Kleiman.
Vnodes: An Architecture for Multiple File System Types in Sun Unix.
Proceedings of the Summer 1986 USENIX Conference, Atlanta, Georgia, June 1986.

[Lyon89] Bob Lyon and Russel Sandberg.
Breaking Through the NFS Performance Barrier.
SunTech Journal 2(4): 21-27, Autumn 1989.

[McKusick84] Marshall K. McKusick.
A Fast File System for Unix.
ACM Transactions on Computer Systems 2(3): 181-97, August 1984.

[Nelson88] Michael N. Nelson, Brent B. Welch, and John K. Ousterhout.
Caching in the Sprite Network File System.
ACM Transactions on Computer Systems 6(1): 134-54, February 1988.

[Nelson91] Bruce Nelson and Raphael Frommer.
An Overview of Functional Multiprocessing for NFS Network Servers.
SunTech Journal 4(1): 84-89, January 1991 (edited version).
Also Technical Report 1, sixth edition, Auspex Systems Inc., July 1992.

[Patterson88] David A. Patterson, Garth Gibson, and Randy H. Katz.
A Case for Redundant Arrays of Inexpensive Disks (RAID).
Proceedings of the ACM SIGMOD Conference, pp. 109-16, 1-3 June 1988.

[Rosenblum90] Mendel Rosenblum and John K. Ousterhout.
The Design and Implementation of a Log-Structured File System.
ACM Transactions on Computer Systems, to appear in 1992.
Also *Operating Systems Review* 25(5): 1-15, October 1991.

[Sandberg85] Russel Sandberg, David Goldberg, Steve Kleiman, Dan Walsh, and Bob Lyon.
Design and Implementation of the Sun Network File System.
Proceedings of the Summer 1985 USENIX Conference, pp. 119-30, Portland, OR, June 1985.

[Schwartz90] Allan M. Schwartz, David Hitz, and William M. Pitts.
LFS--A Local File System for Multiprocessor NFS Network Servers.
Proceedings of the IEEE Systems Design & Networks Conference `90,
Santa Clara, California, 8-10 May 1990.
Also Technical Report 4, Auspex Systems Inc., December 1989.

[SCSI1] American National Standards Institute.

American National Standard for Information Systems--
Small Computer System Interface (SCSI).
Document ANSI X3.131-1986.
This SCSI specification should be read in conjunction with the *Common Command Set (CCS) of the Small Computer System Interface (SCSI),* ANSI document X3T9.2/85-52 (revision 4B).

[SCSI2] American National Standards Institute.
Draft Proposed American National Standard for Information Systems--
Small Computer System Interface (SCSI).
Documents ANSI X3T9.2/86-109 (revision 10C) and X3T9/89-042.

[Seagate91] Seagate Technology.
Seagate Product Family, Elite Family Specifications.
Publication # 1326-005, 1992.

[Srinivasan89] V. Srinivasan and Jeffrey C. Mogul.
Spritely NFS: Experiments with Cache-Consistency Protocols.
Operating Systems Review 23(5): 45-57, December 1989.

[Starr90] Daryl Starr.
SCSI Bus Performance.
Internal Memorandum, Auspex Systems Inc., 25 October 1990.