Sun Microsystems, Inc.
2550 Garcia Avenue, Mountain View, CA 94043
415 960–1300 FAX 415 969–9131

For U.S. Sales Office
locations, call:
800 821–4643
In CA: 800 821–4642

European Headquarters
Sun Microsystems Europe, Inc.
Bagshot Manor, Green Lane
Bagshot, Surrey GU19 5NL
England
0276 51440
TLX 859017

Australia: (02) 413 2666
Belgium: 32–2–759 5925
Canada: 416 477–6745
France: (1) 40 94 80 00
Germany: (089) 95094–0
Hong Kong: 852 5–8651688
Italy: (39) 6056337
Japan: (03) 221–7021

Korea: 2–563–8700
New Zealand: (04) 499 2344
Nordic Countries: +46 (0)8
7647810
PRC: 1–8315568
Singapore: 224 3388
Spain: (1) 5551648
Switzerland: (1) 8289555
The Netherlands: 033 501234

Taiwan: 2–7213257
UK: 0276 62111
Europe, Middle East, and Africa,
call European Headquarters:
0276 62111
Elsewhere in the world,
call Corporate Headquarters:
415 960–1300
Intercontinental Sales

Printed in USA 01/92

A

B

C

D

E

F

G

H

I

K

Notes:

1. The IO Cache is enabled. On the SPARCserver 600MP, the IOC buffers VME transactions into and out of the Mbus, making them much more efficient. The IOC's line size is 32 bytes, meaning that the system attempts to buffer 32 bytes for transfer before requesting bus access.

2. A Virtual Address Cache (VAC) is enabled, and it operates in write–back mode. Alternatively, this message could note that the VAC is enabled in write–through mode. Systems equipped with TI SuperSPARC modules have physical address caches, and are called out as PAC.

3. The kernel probes the SPARCmodules and indicates which kinds of cpus are found. In this case, the Cypress CY605 cpu called out is on a Ross Technologies 6002 module. It includes a Cypress CY7C601 IU, a TI8847–style FPU (second–sourced by Weitek), and a Cypress CY7C605 Mbus level 2 memory management unit. Note that although the kernel is really talking about caches, it calls them out as modules. There is no way to tell what kind of physical modules are installed. The MID is the Module Id, which is an addressing mechanism on the Mbus. Each cache is assigned a MID. MID 0 is reserved as a DMA address, MID 0xf is reserved for non–coherent Mbus level 1 modules (none are supported).

4. The device driver for the SCSI host adapter (esp0 in this case) and the device driver for each device negotiate at probe time to determine how fast synchronous transfers will take place. The negotiated speed is called out in the kernel probe, as indicated here. Notice that various devices often end up negotiating different speedsm as sd0 and sd1 have here. Asynchronous devices, such as the CD–ROM and tape drives (sr0, st0, etc) transfer at whatever rate is available at the time of transfer (as implied by the asynchronous nature), hence no advance negotiation is necessary.

5. This Lance Ethernet interface is equipped with buffer memories to reduce latency between the interface and the Sbus/MSI/Mbus complex.

6. As of Solaris 1.0, the device drivers for most frame buffers call out their resolution and some other characteristics.

7. Finally, the kernel is configuring cpus as such. In the case of the Ross 6002 module, each cpu has exactly one cache, and hence appear as successive cpu numbers. If a processor were to be configured with multiple caches (as it would if it had separate I– and D– caches), each cache would consume a MID, and the cpus would be numbered 0, 2, 4, 8.

8. The system boots in uniprocessor mode; the last thing the kernel does before beginning to execute /etc/rc.boot is enter multiprocessor mode in the scheduler, as called out in this message. If for some reason you wish to boot the system in uniprocessor mode, patch the kernel variable ”uniprocessor” to 1 and reboot. (”echo ’uniprocessor?0x1 ’ | adb –w /vmunix ”)

Appendix D. Kernel Probe Messages

```
IOC  enabled                                                    (1)
VAC ENABLED in COPYBACK mode                                    (2)
SunOS Release 4.1.2 (TERABYTE) #1: Thu Oct 10 23:12:08 PDT 1991
Copyright (c) 1983–1991, Sun Microsystems, Inc.
cpu = SUNW,Sun 4/600
mod0 = Cypress,CY605 (mid = 8)                                  (3)
mod1 = Cypress,CY605 (mid = 9)
mod2 = Cypress,CY605 (mid = 10)
mod3 = Cypress,CY605 (mid = 11)
mem = 64932K (0x3f69000)
avail mem = 62951424
Ethernet address = 8:0:20:a:30:62
dma0 at SBus slot f 0x81000
esp0 at SBus slot f 0x80000 pri 4 (onboard)
esp0:        Target 3 now Synchronous at 3.572 mb/s max transmit rate
sd0 at esp0 target 3 lun 0
sd0: <SUN0669 cyl 1614 alt 2 hd 15 sec 54>                      (4)
esp0:        Target 1 now Synchronous at 5.0 mb/s max transmit rate
sd1 at esp0 target 1 lun 0
sd1: <SUN1.3G cyl 1965 alt 2 hd 17 sec 80>
esp0:        Target 2 now Synchronous at 5.0 mb/s max transmit rate
sd2 at esp0 target 2 lun 0
sd2: <SUN1.3G cyl 1965 alt 2 hd 17 sec 80>
esp0:        Target 0 now Synchronous at 4.167 mb/s max transmit rate
sd3 at esp0 target 0 lun 0
sd3: <SUN0669 cyl 1614 alt 2 hd 15 sec 54>
esp0:        Target 5 now Synchronous at 3.572 mb/s max transmit rate
sd16 at esp0 target 5 lun 0
sd16: <SUN0669 cyl 1614 alt 2 hd 15 sec 54>
sr0 at esp0 target 6 lun 0
lebuffer0 at SBus slot f 0x40000                                (5)
le0 at SBus slot f 0x60000 pri 6 (onboard)
cgtwelve0: screen pixel resolution 1152x900                     (6)
cgtwelve0 at SBus slot 0 0x0 pri 9 (sbus level 5)
zs0 at obio 0x100000 pri 12 (onboard)
zs1 at obio 0x0 pri 12 (onboard)
audio0 at obio 0x500000 pri 13 (onboard)
cpu0 at Mbus 0x8 0x1f4000                                       (7)
cpu1 at Mbus 0x9 0x1f8000
cpu2 at Mbus 0xa 0x1fc000
cpu3 at Mbus 0xb 0x200000
entering multiprocessor mode                                    (8)
root on sd0a fstype 4.2
swap on sd0b fstype spec size 259200K
dump on sd0b fstype spec size 259188K
```

Appendix C. Measuring an Existing System or Mockup

If you measure an actual system, you want to set up to record performance for each type of transaction for the cpu, memory (server working set), network load, and disk I/Os.

Using etherfind(1) can sometimes reveal a lot about how the database server interacts with its clients. In this case, you're primarily interested in the communications between a single client and the server, so the `-between` argument to restrict the output. Measure a sample of each transaction type.

include a section on measuring an obsolescent system and scaling the results to a modern one

Appendix B. Estimated NFS Client Demand

The following are *measured* demands, taken from real systems. They are intended to be representative guideposts for configuring NFS networks. While they can be reasonably used for estimation, they should not be taken as absolute truths.

Client	Action	NFSops/sec	Dominant Ops	Duration
SS2	load 30MB file	27–35	read, getattr	1.5–2 minutes
SS1	load 30MB file	25–30	read, getattr	2–2.5 minutes
SS1	write 10MB file	10–12	write	90 secconds
286 PC	load email file via PC–NFS	8–10	read, getattr	5–10 secconds
386 PC	run 1–2–3	10–15	read, getattr	4 seconds
SS1/SLC	large *make*	4–7	read, getattr, write	10 minutes
diskless SLC (12MB memory)	typical O/A	0.3–1.2	getattr, read	indefinite
HP 730	cat remote >/dev/null	70–90	read, getattr	duration of cat(1)
diskless 3/60 (8MB memory)	Interleaf	2–3	read, write	indefinite

SDM is expected to become part of the SPEC suite release 3.0 (not 2.0).

A.13. AIM−III[88]

The AIM–III is the only commonly–cited benchmarks for which the source code (and results) are proprietary. The AIM benchmark suites (there are five of them) are products of the AIM Technology Corporation. Vendors purchase the benchmark suite and have the option to have the results published in various AIM publications.

AIM–III is a synthetic benchmark which attempts to simulate the load imposed by large numbers of software development users. In this respect, it is very similar to the Kenbus test in SPEC SDM.

Like Kenbus, AIM–III runs collections of shell scripts which simulate multiuser activity. There are number of different scripts run under AIM–III, which result in four different metrics being reported. These are AIM Performance Ratings (AIMs), maximum user load, MPH (miles per hour) and maximum throughput. The AIMs number is a multiple of how fast a VAX–11/780 runs the same set of scripts, in terms of peak load. The maximum user load indicates how many simulated users can be handled “without unacceptably degrading performance”. Unacceptable throughput is defined to be less than one job per minute per user load. The AIMs number is generally *not* achieved at the maximum user load! Miles per hour is a measure of how fast the system runs certain UNIX utilities, again expressed as mutliple of how fast a VAX 780 runs the same utility scripts. The maximum throughput metric is a peak indication of how many simultaneous multitasking processes can be accommodated.

AIM–III attempts to test the same kinds of items as SDM/Kenbus, using a proprietary benchmark suite.

88. Data is from *UNIX System Price Performance Guide, Fall 1991.* Copyright by AIM Technology.

tems. At the least, the executive information system is also likely to put significant emphasis on the network interfaces. Taken together, the whole suite (TPC–A through TPC–D) is likley to be the next major frontier in system performance and tuning, for both DBMS vendors and for the vendors' operating system groups.

A.12. SPEC SDM Release 1.0

The SPEC System Development Multitasking (SDM) is a suite of benchmarks intended to predict the performance of multiuser and server systems under software development loads. Like other SPEC and TPC benchmarks, SDM includes specifications as to how the codes are to be run and how results must be reported.

The present release of SDM includes two codes, SDET and Kenbus. Both execute a number of scripts which simulate the load generated by software developers. Kenbus is a derivative of the widely–available Musbus multiuser benchmark. Results from both are described in terms of *scripts per hour at a specified number of users*, for example, "1097 scripts/hr at 160 users". Large numbers of users suggests that the system under test is suitable for high levels of multiprogramming[85]. Larger numbers of scripts per hour indicate faster throughput for each job. Thus a system which provides 1000 scripts/hr at 3 users will feel much faster than one that provides 300 scripts/hr at 120 users.

Also of pertinence are the shape of the performance curves. The following observations have been made about the shape of SDM curves[86]:

- the ideal attainable curve rises quickly to a reasonably high peak, then stays essentially flat;
- a slowly–rising curve may indicate a poorly performing or poorly tuned disk subsystem;
- a curve that quickly becomes non–linear under light load generally indicates poor tuning;
- a sharp peak probably indicates either memory contention (paging and/or swapping) or poor disk I/O tuning;
- and finally, the width of the flat part of the curve indicates a system's adaptability to sustained overloads.

The SDM suite tests the operating system, the speed of the compiler[87], and expecially the performance of the I/O subsystem. In particular, many of the commands and scripts refer to /tmp. Competition for disk drive which holds /tmp can become the limiting bottleneck; it may be helpful to use MetaDisk to stripe /tmp onto a number of drives in order to reduce /tmp's utilization. If tmpfs is used to accelerate /tmp, it may still help to use MetaDisk to spread swap space across a number of drives.

85. Note the difference between multi*programming* and multi*processing*. The former is a term which describes what is popularly referred to as multi–tasking, meaning the circumstance of having multiple processes active in a system at once. It means nothing at all about how many physical processors are present or in use. All Sun systems offer multiprogramming. Multi*processing* explicitly means having more than one physical processor available. Multiprocessing nearly always implies multiprogramming, but the converse is generally not true (today).
86. Dixit, Kaivalya. *"The SPEC SDM Release 1.0 Benchmark Suite"*. SMCC Performance Engineering Internal Report, May 31, 1991.
87. Kaivalya observes (*op. cit.*) that because the compiler is used in the scripts, the speed of compilation is measured, *but* because the generated code is not then executed as a part of the test, the quality of the code is not tested. In particular, invocation of an optimization phase in the compiler script results in *lower* SDM results!

The TPC–A test primarily stresses the database management system, the DBMS–operating system interface, and the disk write mechanism. Unlike nhfsstone, TPC–A is specifically scaled to the anticpated demand.

Interestingly, the TPC–A is similar to nhfsstone in that it heavily stresses write performance (database updates), without stressing read (database select) performance. The ATM network load is very accurately simulated – but ATMs do very few reads and relatively many writes. This doesn't even remotely approximate the vast majority of online applications, whose performance are often dominated by retrievals and sorts.

Another issue with TPC–A is that it is not representative of applications with aggressive performance requirements. Although TPC–A results of 50 to 150 transactions per second are common in late 1991, this is far in excess of the demands imposed even by large (real) ATM networks: the entire state–wide ATM network of the Massachusetts–based Bay Bank averages about 7 transacations per second on the busiest day of the year. To put this in perspective, the widely–cited American Airlines Sabre system routinely processes 1000–1500 transactions per second, with peaks as high as 2650 transactions per second. Sabre's load is approximately 70% selects.

Fortunately, the TPC–A does test a fairly large subset of the DBMS and the virtual memory management system in the operating system, especially for large tests.

There are few vendors which cite both TPC–A and TPC–B for the same system; however, for those which have provided both results, TPC–A numbers appear to fall in a narrow range of 50–55% of TPC–B numbers. Of course, these aren't even close to official results (since they aren't even run, let alone audited!), but they provide a mechanism for estimating the performance of a system without the equivalent numbers.

A.10. TPC−B

The TPC–B test is a modification of the TPC–A, in which the test is oriented toward discovering the maximum throughput of a server configuration. Instead of directly–connected terminals, the TPC–B uses network connections from client workstations. In addition, wait time is reduced to zero! As a consequence, the TPC–B is a much more intensive test than the TPC–A, particularly with respect to cpu power.

The TPC–B suffers from the same drawbacks as the TPC–A from it which it is derived. Of course, the the inverse relationship between TPC–A and TPC–B applies, so multiplying a TPC–A result by 1.9 seems to yield an approximate TPC–B number (within 10 or 15%).

A.11. TPC−C, TPC−D

Neither the TPC–C nor the TPC–D benchmarks have been formally defined at this writing (fall 1991). The TPC–C and TPC–D are intended to more accurately model different kinds of online applications, specifically a manufacturing system and an executive information system. The manufacturing system simulation will be fairly balanced between selects and updates, while the executive information system is heavily oriented toward selects. Both of these are likley to be much more realistic and useful than the TPC–A and –B. Like the –A and –B, they will exercise the DBMS, the operating system interface, and the I/O subsys-

with relatively long seek times (systems such as Auspex which use many relatively inexpensive, lower–performance disks), or those with long latencies (as do some kinds of disk arrays). Expect laddis numbers to be 30–50% lower than nhfsstone numbers on equivalent systems.

Like nhfsstone, laddis has no explicit scaling mechanism defined as a part of the test. The large data sets help ensure that large networks are reported with reasonable accuracy. It may turn out that laddis is a bit too aggressive in this respect, at least for today's networks. Few existing networks average 15MB of NFS working set per client, although when multimedia applications become more widespread this is much more likely to reflect live applications.

The same reporting deficiencies are likley to affect laddis as do nhfsstone. Caution is advised.

Laddis has been proposed to SPEC, and is expected to appear in the SPEC 3.0 suite sometime in 1992.

A.8. TP1

The TP1 is the original transaction processing benchmark. It simulates the activity of a large network of Automated Teller Machines (ATMs). The TP1 benchmark is merely a piece of code; there are no associated standards, auditing, or even rules. As a consequence, the results from the TP1 are subject to wild variations, both from innocent variations in interpretation and from also from questionable benchmarking technique. A wide variety of optimizations have been applied to TP1 codes.

Although the TP1 and the TPC–A and TPC–B are structurally very similar, the results from TP1 and TPC–A/B are *not* comparable[84]! New literature citing TP1 numbers should be considered suspect.

A.9. TPC−A

Similar to the SPEC consortium, the Transaction Processing Council (TPC) provides definitions for a variety of benchmarks designed to provide information about online, transaction–oriented database processing systems. The TPC–A benchmark is a deriviative of the famous (notorious) TP1 test. It simulates the transactions involved in managing a large network of automated teller machines. The most important difference between the TPC's benchmarks and the predecessor TP1 is the rigorous framework built around the actual benchmark definitions. In fact, for a benchmark to be considered an official "TPC" result, the entire benchmarking process must be audited by a TPC–approved auditor! Like any TP1 number, results which are "preliminary" or even merely not specifically tagged to be "audited" should be regarded with some suspicion.

Sun has traditionally not provided TPC–A results, due to the company's focus on client–server computing, although results for Sybase, Informix and Oracle are likely to be forthcoming in late 1991 or early 1992.

84. Nonetheless, some vendors have been known to cite both TPC–B and TP1 benchmarks in the same literature. This is exceptionally misleading, even though the source of each number *is* shown.

With all reads being cached on the server, the disks spend virtually all of their time performing synchronous writes, which of course are not cacheable without special arrangements such as PrestoServe or eNFS.

Even the volume of written data is very small (less than 5MB per tested file system). As a consequence, nhfsstone can substantially overestimate the performance of servers with relatively slow disk access, because seek times are necessarily very short. (A cylinder is about 700KB on the 1.3GB disk, so only about fifteen twenty cylinders are active; even with the smaller 424MB disk, only about thirty cylinders are active.) Under live application loads, writes will be spread over much more data area, resulting in much longer typical seek times. The fact that very little data is read from the disk during the test further aggravates this situation.

The end result is that the nhfsstone benchmark tests several subsystems very thoroughly: the network protocol layers, the disk write mechanisms, and to some extent the scheduler. On the other hand, the default mix probably doesn't accurately reflect the performance of current networks; and finally, the test is especially inaccurate when it comes to measuring very large, very high–throughput networks.

A deficiency in the *reporting* of nhfsstone results is that many vendors report only the maximum number of NFSops achievable, even though it more appropriate to report other information to provide a more complete picture of NFS performance. The average latency of each request, and particularly the shape of the performance curve are both important.

The latency is frequently overlooked as a metric, although it is nearly as important as the raw number of NFS ops. Most vendors report nhfsstone rates with up to 70ms latency (in particular, Auspex does this, primarily because they have lots of area under the slow end of the rate–vs–latency curve). This doesn't reflect true customer requirements, as 70ms is *very* slow for a disk access – even discount PCs rarely come with disk drives with seek times slower than 40ms. The number of operations should be specified *along with the latency at which that peak rate is achieved. Rates with latencies longer than about 35–40ms should be disregarded!* If the environment includes diskless clients, the latency requirement should be even stricter, probably on the order of 25–30ms, since virtual memory response is critical to client performance.

The shape of the performance curve is also important, because balanced systems tend to provide quick response through a wide range of rates.

A.7. Laddis

The Laddis[83] benchmark is an attempt to address some of nhfsstone's shortcomings, as discussed above. The code is derived from nhfsstone, with substantial modifications. In particular, laddis utilizes a much larger data set – about 15 MB per client system (not per file system). This causes many more reads to be addressed to the server system, resulting in much more stress on the disk subsystems, as only very large memory configurations (more than 64MB) are able to cache a majority of the data accessed in the test. We expect that the laddis tests will more accurately distinguish weaknesses in the disk subsystems, especially those

83. The name is an acronym for Legato–Auspex–Digital–Datageneral–Ibm–Sun, the six companies which participated in the development.

A.6. Nhfsstone

The nhfsstone benchmark was developed by Legato Systems[80] to measure NFS performance. Unlike most other benchmark programs, nhfsstone is *not* run on the system under test. It is run on a group of client machines, generating a synthetic load for an NFS server. Two versions are in common circulation, versions 2.0.3 and 2.0.4. Version 2.0.4 introduced a few minor bugs, so most cited results are from version 2.0.3 (including all of Sun's current published results).

The nhfsstone code is capable of generating any kind of NFS operation mix, although the numbers normally quoted are from the so–called Legato Mix. The Legato mix represents the distribution of operations common in Sun server installations of 1987, which included SunOS 3.5, and a heavy dose of small–memory diskless clients. While this mix is useful for directly comparing with other systems, it doesn't accurately reflect what happens in most current Sun installations, where diskfull and dataless clients are the rule and read operations predominate on most servers. Sun's engineering groups study a variety of other mixes, including the Brown mix, which more closely approximates the current dataless environments running under SunOS 4.x.

The nhfsstone code makes direct RPC calls to the server, in order to avoid the effects of file cacheing on the client (load–generating) machines. This is useful for eliminating client configuration issues, making the test reasonably insensitive to which clients it is run on. However, specifically because it negates client–side file cacheing, it artificially alters the operation mix. In particular, because reads are frequently cached on the client side (in addition to possibly also being cached on the server side), nhfsstone tends to overestimate the proportion of reads and underestimate the proportion of writes, in relation to each specific mix of requests generated by an application set on the client. This is important because writes are much harder to optimize!

Another effect of bypassing the client–side cache effects is that nhfsstone underestimates the performance of a network media. Since relatively large volumes of data are requested from the server, more network bandwidth is required to support a given NFS mix.

In addition to bypassing the client biod, the design of the nhfsstone code is such that it does not scale well to high–intensity tests. All of the requests generated on the client are generated randomly on a uniform distribution basis, which means that the locality of reference for each driving process is not simulated. The result is that nhfsstone actually exercises only a very small portion of the server's disk, even when the server is subjected to very high load[81]. The active data during the test is approximately two megabytes per exported file system[82] – not per client process – which means that virtually every server is able to satisfy *all* read operations out of virtual memory cache, even on servers with very little (16MB) memory. This is true for tests of any level of load, even those simulating loads of 1000 NFSops or more!

80. Legato Systems was founded by a number of engineers from Sun. Most of those engineers were directly involved in the development of the NFS architecture and its first implementations. It is *quite* safe to say that Legato understands NFS very well.

81. This is markedly different from the TPC benchmarks, in which the operating database must be scaled with the demanded load.

82. Typical practice is to provide one server file system per client *system*, which obviously does not correspond to actual application use.

geometric mean of the ten results! Application of the KAI preprocessor to the SPARCstation–2 moved its SPECmark rating from 21.0 to 25.0, or a full 19%. Other machines with larger caches are “accelerated” to even greater extent.

While the optimizations involved are honest (in that they can legitimately be applied to an entire class of codes), they do not accurately reflect the majority of user applications. Unless the target application heavily emphasizes matrix multiplication, the distortions from matrix300 are sufficient that it may be wise to look at the entire set of SPEC results for a machine and recompute a second geometric mean with matrix300 removed.

Two other floating–point codes, nasa7 and tomcatv, are also susceptible to these optimizations, but to much lesser extent. The SPEC 2.0 Suite, expected to be released in early 1992, has dropped matrix300 (and did not include a proposed matrix1000), so SPEC 2.0 results will likley vary less widely. SPEC 2.0 has twenty codes, in the same proportions as SPEC 1.0: 30% integer and 70% floating point.

At present, with no multithreading compilers, the SPECmark measures only the performance of a single processor. The SPEC 1.0 suite does virtually no I/O and makes very few system calls. As a result, SPEC 1.0 primarily measures the main processor, memory complex, and the C and Fortran compilers. This is not expected to change until at least SPEC 3.0, expected in late 1992 or early 1993.

A.4. SPECthroughput

The SPECthroughput metric is used to characterize the performance of multiprocessor configurations. The test runs copies of the SPEC 1.0 suite for each processor; the sum of the results is reported as the SPECthroughput number. SPECthroughput always less than the number of processors times the SPECmark of an individual processor because the processors share the operating system (even in fully symmetric systems) and many other hardware resources. In the SPARCserver 600MP, the limiting resources are the Mbus and the memory subsystem, due to the small size of the caches associated with each processor. Because it depends upon the SPECmark, SPECthroughput has basically the same strengths and weaknesses. Of course, since multiple copies are being run, the efficiency of the scheduler tested. In small–memory configurations, it would also stress virtual memory management.

A.5. IOBench

The IOBench test is a Sun internal test, used to compare the disk (and virtual disk) subsystems of various platforms. It generates as many I/O operations as possible against a disk device, measuring both the speed of the operations and the amount of processor time required to accomplish them. Both raw and file system can be tested; random and sequential access of various block sizes are selectable. Multiple processes are easily accomodated.

Most of the I/O capability numbers (including Online:DiskSuite) cited in this work were obtained from IOBench.

IOBench has very restricted usefulness: it tests the I/O subsystems and their operating system support. It does a good job at this, but it tests little else, and does not generally reflect application usage.

Algebra System, and the “coded” description means that the most heavily used inner loops are coded and tuned for optimal speed on a particular architecture/implementation combination. Usually this can be translated as “hand–coded in assembly language.”

Current practice is generally to quote floating–point performance in terms of Fortran, non–coded Linpack figures. Loop unrolling is becoming less important because compilers and preprocessors are becoming more sophisticated to the point of unrolling loops. On Sun platforms, the compiler does not unroll loops, but the KAI preprocessor does, if requested. In the absence of the KAI preprocessor, the version used by Sun has partially unrolled loops, selectable by using the –DROLL compiler flag. There are no coded BLAS versions for Sun platforms, as this would not reflect actual customer use: very few if any customers are using assembly language for their own applications, and the commercial math subroutine libraries such as IMSL and NAG are already highly hand–optimized for SPARC.

The 100x100 Linpack uses only 80KB of data; normally this would fit reasonably well in a 64KB cache. However, conventional Fortran coding methods call for allocation of an array larger than necessary to run the actual case – 200x200 in the case of Linpack. This causes the array to be sparse and much larger. The 200x200 matrix is 320KB, and this *doesn’t* fit well in a 64KB cache, causing many cache misses.

Linpack stresses the floating point arithmetic unit(s), and on systems with small (64KB or less) caches, it is a good test of the memory subsystem, since it causes many cache misses. Beginning with products introduced in 1991 (by HP) and 1992 (by most others, including Sun), Linpack will essentially fit in caches and will become a cpu–only test (like dhrystone).

A.3. SPECmarks

The most commonly cited benchmark today is the SPECmark. The SPEC 1.0 suite consists of ten Fortran and C codes which are intended to test the integer and floating point abilities of a system. The codes were chosen to be large enough to stress the caches of the day (single–level, unified 64KB), and hence also the memory subsystems. The number reported is the geometric mean of the results of all ten tests. Of the tests, four are integer–oriented and six are floating–point oriented.Within reason, the SPEC 1.0 suite accomplishes its goals. With one notable exception, its results reasonably approximate the actual delivered performance of a platform on scientific codes.

The exception is a floating point code, known as matrix300, which performs some large array calculations. The point of this test was to include matrix manipulation on an array large enough not to fit into a cache, hence testing floating point ability when essentially operating from memory. Unfortunately, matrix300 has proven to be very susceptible to a class of optimizations which affect only specialized matrix operations. Every major vendor, including Sun, has either implemented this optimization in their compiler or cites results obtained by preprocessing the Fortran code with a a source–level optimizer from Kuck and Associates (KAI). In addition, IBM went to the trouble of adding a joint multiply–add instruction to the RIOS architecture specifically tailored to this kind of activity! (The multiply–add instruction is applicable to a wider class of applications than the use of the KAI preprocessor, but it still specifically intended for certain relatively small classes of matrix manipulations.) The result of these optimizations is a tremendous result on matrix300 – so large that it distorts even the

Appendix A. Standard Benchmarks – What they measure

Todays' marketplace is strewn with citations of many benchmarks, which purport to predict the performance of systems with ever–increasing accuracy. Unfortunately, the publicly–touted benchmarks are all synthetic. Although some (notably many of the SPEC offerings) include code which descends from user applications, none of them measure what is really relevant to the customer – his own applications. In fact, few of the often–cited benchmarks are complex enough to reflect more than a small part of even the facet of performance that they are intended to measure. In this Appendix, we address some of the most common benchmarks. We will attempt to provide some insight as to what they measure, what they don't measure, and what can be learned from them.

A.1. Dhrystone MIPS (also VUPs)

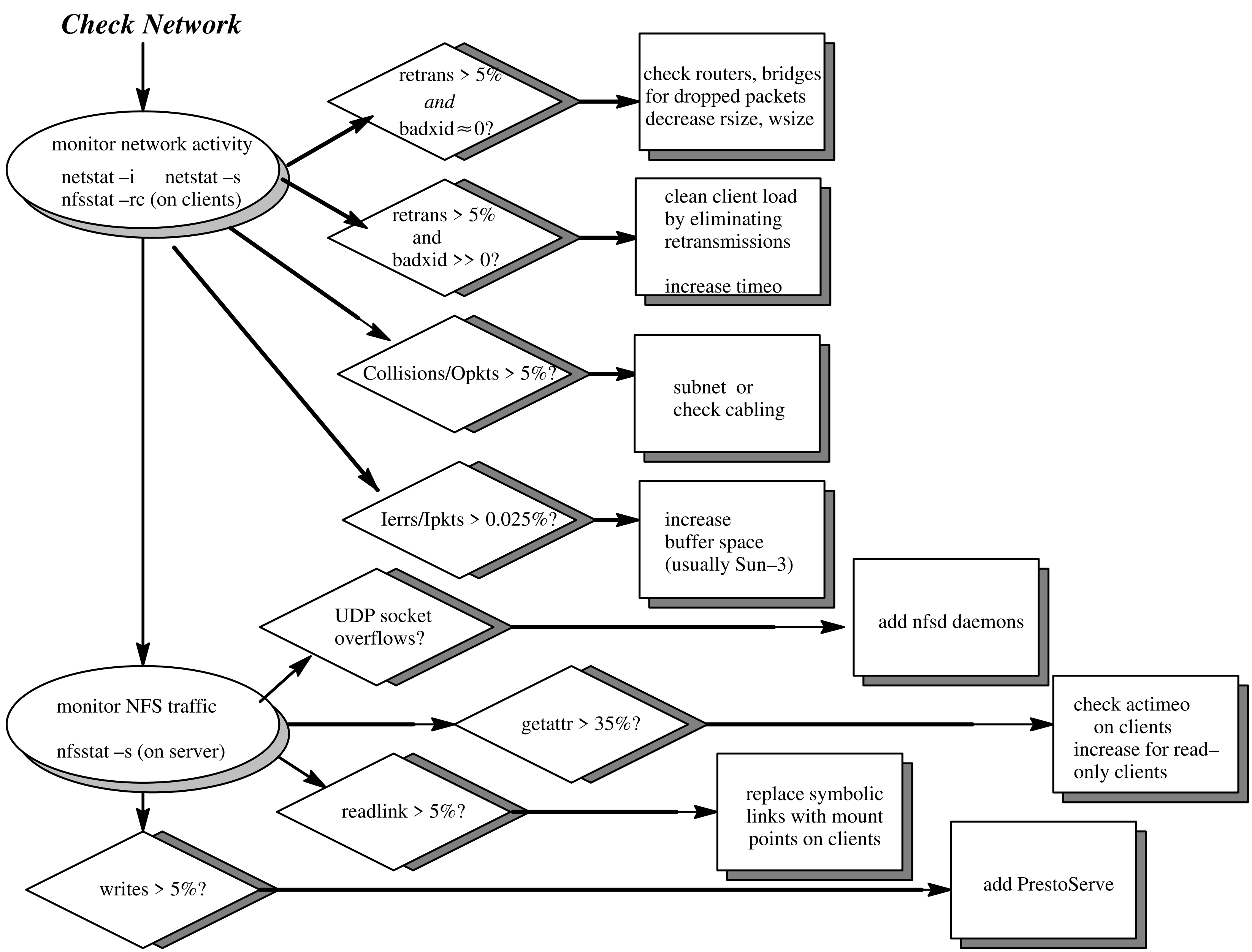
The "MIPS" metric quoted by most vendors is measured by a program known as Dhrystone. There are at least two versions, known as version 1.1 and version 2.1. Most vendors quote results from version 1.1. Sun normally quotes version 1.1 results! The dhrystone benchmark is a small, synthetic code which was intended to measure integer performance, as opposed to its predecessor, whetstone, which was more oriented toward floating–point performance. The most notable shortcoming is that, because it is a synthetic code, it normally fits entirely into a system's cache. For this reason, dhrystone normally overestimates the performance of user applications, since most modern applications are hundreds of times the size of the cache. Systems with small caches benefit the most from this (systems based upon the MC88000 frequently have only 16K or 32K of cache and probably benefit the most).

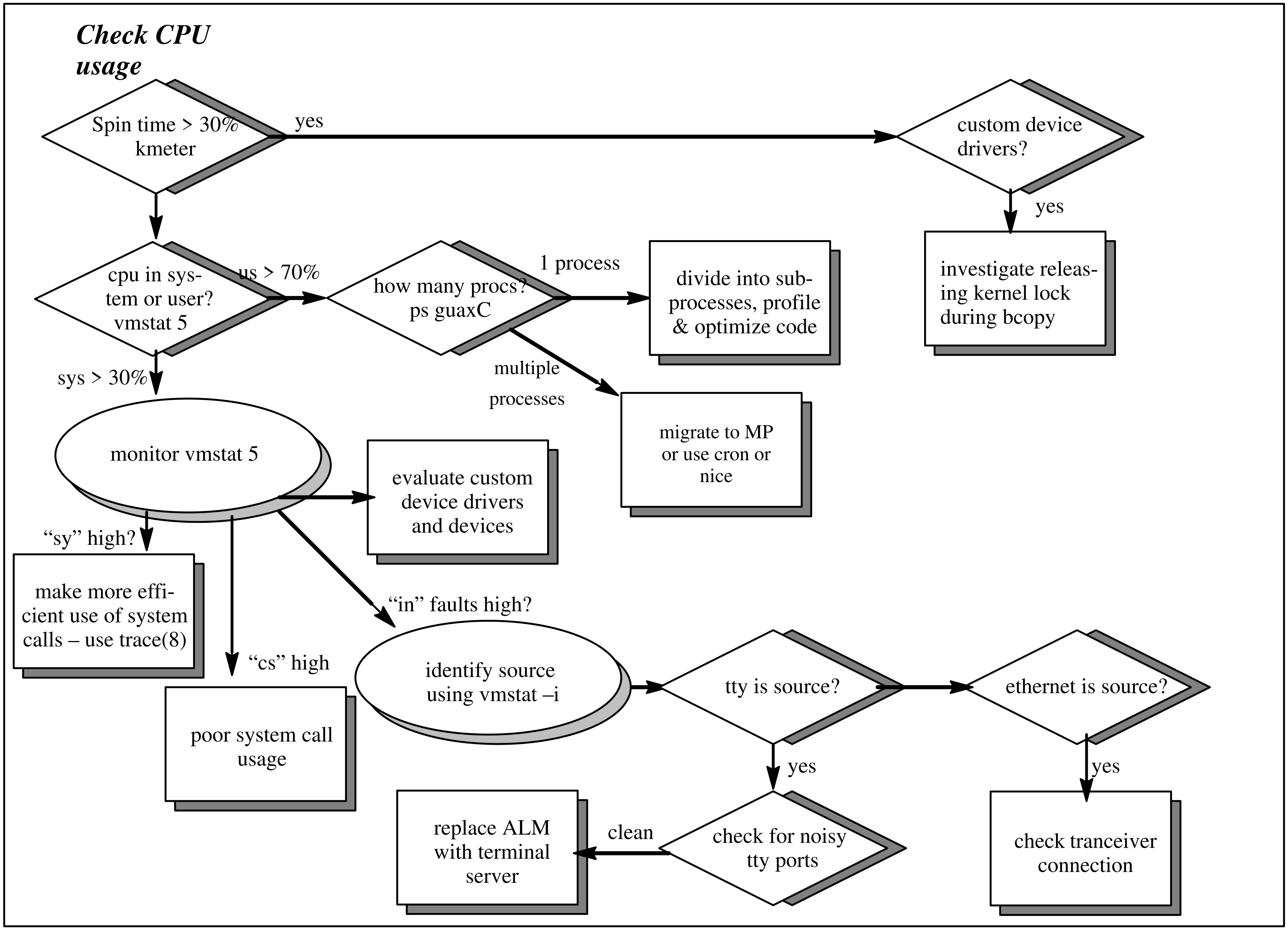
In addition, because there are two common versions, there is also confusion about comparing version 2.1 and version 1.1 results. Dhrystone 2.1 numbers are nearly always smaller than 1.1 results. The number reported is the number of dhrystones reported by the code, divided by the speed of a VAX–11/780 running Ultrix with the Ultrix C compiler. The 780 was chosen as a baseline because it was generally considered to execute a million instructions per second, and because it was a common point of reference. The 780's dhrystone speed is taken to be 1767, as measured by dhrystone 1.1. DEC uses the (equally obsolescent) MicroVAX–II as its basis, quoting the integer speed of its new processors in multiples of the MicroVAX–II and calling them VUPs (VAX Unit of Performance). The MicroVAX–II is generally accepted to be 0.9 MIPS (dhrystone 1.1).

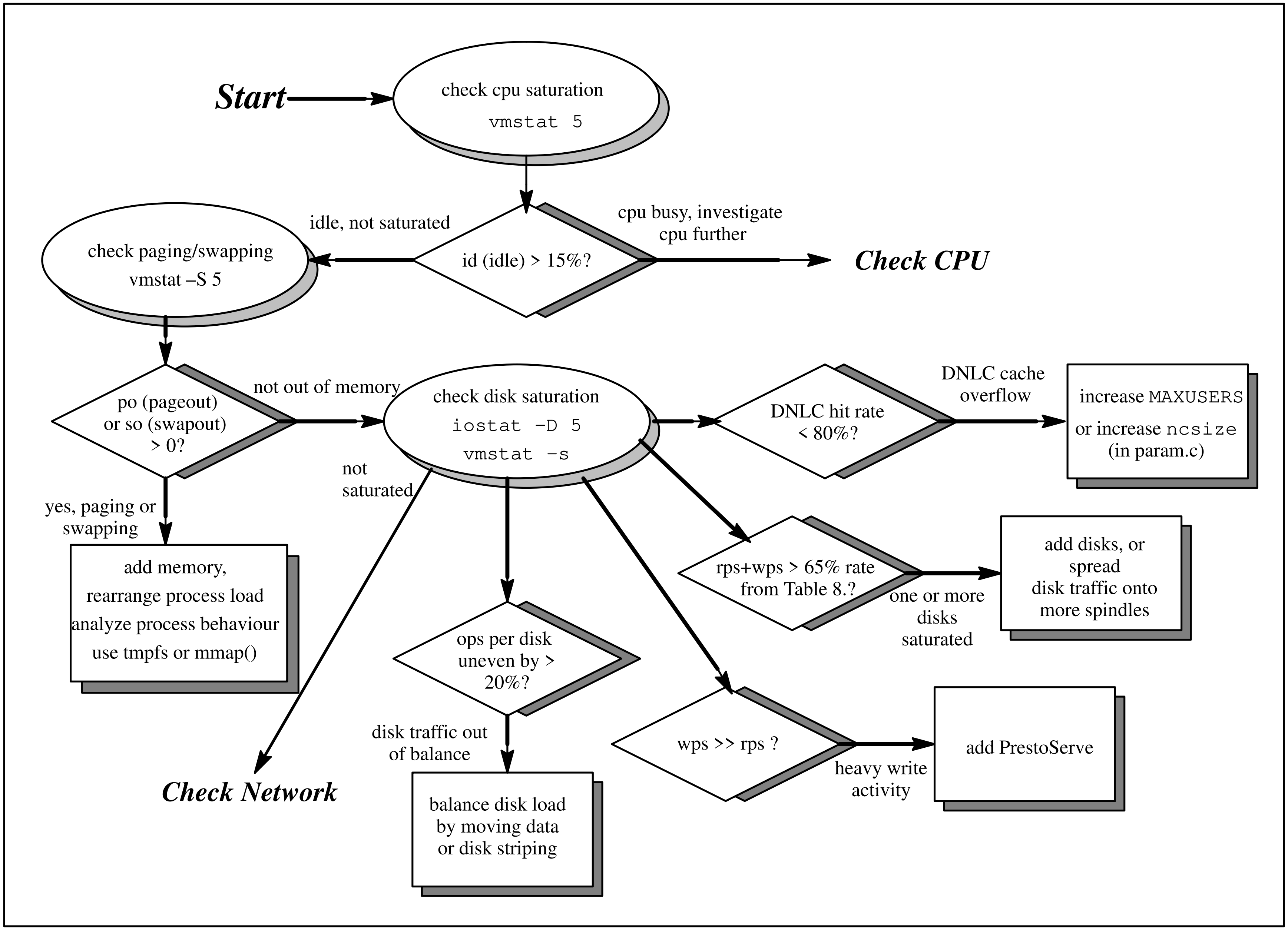
By design, the dhrystone test measures integer operation speed and the efficiency of the C compiler. Because it normally fits entirely in the cache, it measures the speed of register–to–register operations only.

A.2. Linpack MFLOPS

The Linpack metric reports the number of floating–point operations achieved per second while computing the solution of a 100x100 solution of linear equations. There are a wide variety of variants in circulation; some are coded for single–precision, others for double–precision. There are versions with loops unrolled and with loops left rolled. Finally, there are some versions known as "coded BLAS". The BLAS acronym stands for Basic Linear


Check Network
monitor network activity
netstat –i netstat –s
nfsstat –rc (on clients)
retrans > 5% and badxid ≈ 0?
check routers, bridges for dropped packets decrease rsize, wsize
retrans > 5% and badxid >> 0?
clean client load by eliminating retransmissions
increase timeo
Collisions/Opkts > 5%?
subnet or check cabling
Ierrs/Ipkts > 0.025%?
increase buffer space (usually Sun–3)
UDP socket overflows?
add nfsd daemons
monitor NFS traffic
nfsstat –s (on server)
getattr > 35%?
check actimeo on clients increase for read–only clients
readlink > 5%?
replace symbolic links with mount points on clients
writes > 5%?
add PrestoServe


Check CPU usage
Spin time > 30% kmeter
yes
custom device drivers?
yes
investigate releasing kernel lock during bcopy
cpu in system or user? vmstat 5
us > 70%
how many procs? ps guaxC
1 process
divide into sub-processes, profile & optimize code
multiple processes
migrate to MP or use cron or nice
sys > 30%
monitor vmstat 5
evaluate custom device drivers and devices
"sy" high?
make more efficient use of system calls – use trace(8)
"cs" high
poor system call usage
"in" faults high?
identify source using vmstat –i
tty is source?
ethernet is source?
yes
yes
check for noisy tty ports
clean
replace ALM with terminal server
check tranceiver connection


Start
check cpu saturation
vmstat 5
idle, not saturated
id (idle) > 15%?
cpu busy, investigate cpu further
Check CPU
check paging/swapping
vmstat –S 5
po (pageout) or so (swapout) > 0?
not out of memory
yes, paging or swapping
add memory, rearrange process load analyze process behaviour use tmpfs or mmap()
check disk saturation
iostat –D 5
vmstat –s
not saturated
Check Network
DNLC hit rate < 80%?
DNLC cache overflow
increase MAXUSERS or increase ncsize (in param.c)
rps+wps > 65% rate from Table 8.?
one or more disks saturated
add disks, or spread disk traffic onto more spindles
ops per disk uneven by > 20%?
disk traffic out of balance
balance disk load by moving data or disk striping
wps >> rps ?
heavy write activity
add PrestoServe

3. A Brief Tour of Server Tuning

3.1. Processor Complex

3.2. Memory Bottlenecks

3.3. Disk I/O Problems

3.4. Networking Issues

Room for Expansion

Interstream's product doesn't combine with the Network CoProcessor because both products install their own nfsd programs. There isn't any special reason that the two couldn't be integrated together, other than that they are products from two different (small) companies. The Network CoProcessor provides *much* greater benefit than eNFS (it accelerates all NFS operations along with TCP, IP and UDP processing). The Network CoProcessor is recommended over eNFS if there is a choice to be made.

Rule XXVIII. Configure eNFS if NFS writes are more than 6% of the mix, or if NFS writes are the limiting aspect of performance, and when PrestoServe cannot be configured.

2.13. SunDBE Database Excellerator Software

Designing and tuning an operating system is a delicate balancing act, especially when the system is intended as completely general purpose system. Many engineering tradeoffs must be made to permit reasonable performance for as wide an audience as possible. Unfortunately, this may mean less than optimum performance for individual classes of application. SunDBE is a mechanism for realigning some of those engineering tradeoffs to skew the performance of Solaris toward optimizing for database servers.

SunDBE 1.0 offered the following features:

- support for very large shared memory partitions and improved virtual memory management via software PMEGs. Affects all Sun–4 and especially Sun–4c machines.
- improved, lower–overhead context switching.
- faster, specialized asynchronous I/O for operations < 63KB. Used by Oracle, Sybase and Informix to increase performance when operating on raw disk partitions.
- 1024 file descriptors per process (instead of 256). Used by Informix to handle large numbers of clients.

in addition to all of SunDBE 1.0, SunDBE 1.1 offers the following features:

- optimizations to the system call dispatcher
- optimizations to the read and semop system calls
- "intimate" shared memory segments (shared page tables). Used by Oracle and Informix to avoid excess consumption of PMEGs or software PMEGs.
- improved UFS update performance, eliminating excess updating of certain inode information during heavy updates

quests to be serviced without intervening sync()'s; the utilization savings can be as much as 40% in heavily write–bound environments[76]. It also reduces (improves) write latency, at some cost in cpu power – about 12% in update–oriented environments, and about 20% in write–oriented situations[77].

A word of caution about eNFS: while it is effective in real–world environments, *it has little or no effect on nhfsstone benchmarks!* This is because nhfsstone is written in such a way that the driving client process cannot issue any new NFS requests until the first one is acknowledged[78]. As a result, the eNFS–modified server usually times out waiting for additional write requests (writes are only about 15% of the common Legato mix), negating eNFS's primary optimization. This does not happen in normal situations, where multi–tasking clients typically have 4–8 biod's (*b*lock *i/o d*aemon) operating simultaneously. The biod daemons work together to issue parallel NFS requests. For example, the typical file system operation is an 8K operation, but under disk clustering (which of course works just as well on clients as servers), most disk write requests will be clustered into 56KB blocks. This is normally broken down into seven 8KB NFS write requests and distributed to the various biods for processing in parallel.

The eNFS product can be combined with the PrestoServe, if sufficient cpu resources are available to drive both. This is only desirable if the write portion of the NFS mix is primarily writes, as opposed to updates. A study at Sun indicates that the combination of eNFS and PrestoServe provides approximately four times the performance *on NFS writes* as compared to a base system without either PrestoServe or eNFS[79]. If the write mix is mostly updates, there is little to be gained by additionally configuring eNFS, as the benefit over configuring with only PrestoServe is only 8%. This combination is only appropriate in circumstances which have an abnormally high proportion of NFS writes, causing the performance bottleneck to be in the disk subsystem even with PrestoServe.

	Sequential Writes	Random Updates
base system	20	30
base + Presto	55	51
base + eNFS	66	54
base + Presto + eNFS	82	55

Table 14. Relative NFS (nhfsstone) performance of PrestoServe, eNFS, and combinations.

76. Khemani, Rajiv. "eNFS: How and by How Much? The Architecture and Performance of Yet Another NFS Accelerator". SMCC Performance Engineering Internal Report, March 29, 1991.

77. In this context, "write" means appending to the end of the file, "update" means a write–in–place. The difference is that writes involve the allocation of new disk blocks, while updates to not.

78. Nhfsstone completely bypasses the biod daemons. This means that the benchmark *underestimates* the server's performance in a real–world environment, because normally the biod's will permit eNFS to cluster writes on the server. This is even true without eNFS, because the biod's also perform client–side cacheing of both reads and writes. Nhfsstone bypasses all of these optimizations. See Appendix A.6. for additional information about the implications of the nhfsstone benchmark.

79. *op. cit.*

As with disk mirroring, it is usually best to permit a database management system's I/O manager manage spreading data across multiple disk spindles if it uses raw partitions and has the facilities to support disk striping and/or concatenation, and for the same reasons.

2.11.3. Disk Concatenation and Very Large File Systems

The meta driver also permits simple concatenation of disk partitions; this is simply a mechanism to permit larger partitions, either for the purpose of storing files which are larger than a single partition, or for the convenience of having to manage fewer partitions. (Imagine trying to keep manage 40 disks, each with 4 partitions!) For reasons which are not well understood yet, the present implementation seems to impose very high seek overhead on file systems which are concatenated but not stripped. As a consequence, we recommend that all concatenated partitions actually be stripped with an interlace factor of between 8KB and 32KB. Without the stripping, disk I/O performance is degraded by 10–30% on the SS690MP, 5–72% on SS630MP or SS670MP, and 5–42% on the SPARCstation–2.

> **Rule XXVII. Configure any concatenated partition as a stripped partition. Never configure "raw" concatenated partitions without striping.**

2.12. eNFS – NFS Acceleration Software by Interstream

Interstream offers software which accelerates NFS servers, known as eNFS. This package addresses many of the same issues as the PrestoServe board, but in very different ways. The most fundamental issue is the implementation of stateless NFS writes. As noted above, the PrestoServe accelerates this process by intercepting the disk write request in the device management layer of the kernel and committing it to non–volatile memory. The eNFS software operates by accepting the NFS write request in the NFS protocol layer. It then issues a normal UNIX write to send the write to the disk; however, instead of issuing a noncacheable, synchronous write (as normal NFS does), eNFS's request is for a cacheable, asynchronous write. The server does *not* acknowledge the request yet, since it isn't actually committed to the disk. Instead, the server looks in the input queue for more requests. If other write requests are found to be pending, the server goes on to process them. Finally, either the queue of write requests will be exhausted, or some amount of time elapses. In either event, a sync is issued, which will commit all of the cacheable writes that were issued. After the sync returns successfully, the server sends acknowledgements to the requesting clients.

The net effect is to handle groups of writes in a single batch, rather than to issue a number of individual synchronous writes, essentially (re–)implementing the I/O clustering which is normally defeated on NFS servers. This strategy works best when there are relatively large numbers of biod's running on the clients issuing the requests, which permits the clients to generate more requests in less time.

In addition to the acceleration of NFS writes, eNFS includes some improvements to the Ethernet and NFS subsystems. These include extending the Ethernet (both le and ie) and NFS input queues, expanding the loan buffers for ie0 interfaces, and similar optimizations.

As far as system performance is concerned, eNFS has many of the same effects as PrestoServe, both positive and negative. It reduces disk utilization by permitting multiple write re-

2.11.2. Disk Striping

Disk striping permits a logical disk partition to be spread across a number of physical disk devices. The partition's address space is divided into long strips, called a *stripe*. Each device then takes its share of stripes. The "width" of the stripes is called the *interlace*; ideally it should be smaller than the average size of an I/O operation. In such a circumstance, the data request can be divided into as many sub–operations as there are physical devices; each operation can proceed in parallel. Striping can be thought of as a software mechanism for constructing a simple disk array.

While the software permits striping together of mixed kinds and sizes of disks, an important fact is that the components of a stripe should be the same size. If the components are not the same size, each of the components are truncated to the size of the smallest component before striping. Thus striping together two 1.3GB disks and one 669MB disk yields a partition of only 3 * 669 = 2 GB.

In the current release, only specific applications and configurations will benefit from striping. In particular, SS690 systems doing essentially random I/O will see about 30% increase in disk performance; SPARCstation–2's equipped with 669MB disks work well in all applications. Other configurations are to be avoided, at least for performance reasons. Other than as a logical outgrowth of disk concatenation (see below) it is difficult to foresee other circumstances under which striping should be used. These results are described in Table 13.

	Random Read	Random Write	Seq. Read	Seq. Write
SS2 1.3GB	0	+45	–21	–60
SS2 669MB	+10	+17	+15	+55
SS630/670 SCSI	–3	+2	–40	–55
SS690 IPI	+28	+30	–32	–30

Table 13. Performance impact of two–disk striping on disk I/O performance. The numbers in this table indicate the change in IO operations per second, relative to a single constituent disk. For example, a SPARCserver 690MP doing random writes on a pair of striped 1.3GB IPI disks is able to perform 30% more writes than the same platform writing randomly to a single IPI disk.

The Meta driver permits various interlace sizes to be configured. The default is one cylinder, which does not seem to be optimal for any configuration or access type (a cylinder is about 700KB on the 1.3GB disks, 240KB on the 424MB disk). The minimum interlace is 8KB, which is the size of a single file system block. Interlaces up to 32KB (for two–drive stripes) or so are reasonable, since the file system will be making access in units of 56KB if it can. Adjust accordingly for stripes consisting of more than two drives. The data reported here is for an 8KB interlace; interlaces of 56KB are within 5% of the 8K numbers. The default cylinder–size interlace is slower than 8KB or 56KB interlaces in all cases.

Rule XXVI. Configure striping for maximum random (server) I/O performance.

which limits the loss to about 20%. Just exactly why this option improves write performance has not been determined, although the empirical evidence is quite clear.

The SPARCserver–2 equipped with the 1.3GB SCSI disks makes out the best of the three platforms. Mirroring with the geometric read (–g) option actually improves performance by 10% for random reads and by nearly 15% for random writes. Sequential access loses only 1% for reads (again using geometric reads); a penalty of about 22% is imposed for any of the modes in sequential writing. These results can be summarized in Table 12.

	Random Read	Random Write	Seq. Read	Seq. Write
SS–2 SCSI	+10	+15	–1	–22
SS630/670 SCSI	–5	–7	–10	–20 (–r only)
SS690 IPI	–2	–15	–12	–30

Table 12. Summary of performance impact of two–way disk mirroring. Figures quoted here are number of IO operations performed under mirroring, relative to the same platform performing the same kinds of operations against a single like similar disk.

It is important to put these figures in perspective. These represent the performance impact of mirroring on a process which does *nothing* but disk I/O. Very, very few applications behave in this manner. Shuff's study investigated the impact of mirroring on multiuser applications, with both the application data and the system's disks (both root and swap) mirrored. Under these conditions, even the SS690MP suffered overall degradation of only 2–6%. A SPARCserver–2 under similar load benefitted 3–6%. Other kinds of load can expect similar results. For example, the nhfsstone benchmark typically has the disks doing random writes about 40% of the duration of the benchmark, so the overall degradation would be less than 7% in the worst case[75]. Even database applications typically are cpu–bound, rather than disk–bound (discussed in *Disk Subsystems* above). Consider the performance penalties associated with mirroring to be the cost of higher reliability.

> **Rule XXV. Configure mirrored partitions to increase random access performance of SPARCserver–2's. Otherwise, configure mirroring only for reliability/availability reasons.**

When configuring disks for a database management system, it is generally wise to permit the DBMS to provide mirroring if it has facilities to do so. The database management systems usually use the raw disk interface and assume much of the responsibility for optimizing I/O. Operations on the UNIX file system are more diverse than operations generated by the DBMS I/O managers, and the I/O manager usually can optimize on this information. The DBMS internals generally take into account mirroring when such formulating optimizations.

75. Nhfsstone underestimates the proportion of disk reads because it only accesses about 2MB of data per client, considerably less than found in actual practice. As a consequence, nhfsstone tends to exercise the virtual buffer cache rather than the disk reading mechanism. However, mirroring has virtually no impact on reading, so even more realistic loads such as the laddis benchmark (which accesses a very conservative 15MB per client) would probably suffer no serious degradation.

operation cost is 3.3ms (4.21 ms normalized to a 28.5 MIP cpu) * 1.33 (the overhead for striping on IPI disks) = 4.39 ms. Two stripped reads must be performed, so this is 8.78 ms per operation. The overhead for the mirroring is 27%, or 4.39 * 0.27 = 1.19 ms, so the overall expense is 8.78 ms + 1.19 ms = 9.97 ms. This is example is more complex than practiced by most sites, which either stripe *or* mirror specific partitions, but generally not both. The overhead associated with hot sparing is so small that it can be safely ignored.

The version of MetaDisk in the Online:DiskSuite product also includes the functionality of QuickCheck 1.0, which permits very fast – usually trivial – checking of file system consistency after a reboot. This functionality is also now bundled into SunOS 4.1.2.

	Random Read	Random Write	Seq. Read	Seq. Write
SS690MP, IPI				
Striped, 8K	33	27	57	50
Concatenated	25	20	64	54
Mirrored, –Sg	27	35	14	34
Mirrored, –g	30	49	14	37
SS630MP or SS670MP, SCSI				
Striped, 8K	34	44	80	34
Concatenated	35	38	81	56
Mirrored, –Sg	36	44	17	19
Mirrored, –g	27	46	18	18

Table 11. CPU cost of various MetaDisk options. Numbers cited are percentage of additional cpu required to complete given I/O, relative to the amount required without MetaDisk (see Table 7.) The –g option selects geometric reads; the –S option selects serial writes.

2.11.1. Disk Mirroring

Mirroring using MetaDisk permits a disk image to be duplicated onto two or three (not necessarily physically identical) disks. A recent study at Sun[74] revealed that performance of MetaDisk file system partitions varies substantially depending upon base platform and even upon disk. This study addressed only two–way mirroring (expected to be the most common case).

On the SPARCserver 690 with IPI disk subsystems, mirroring imposes a performance penalty of 2–5% for random file system reads, and a penalty of 10–20% on random file system writes. For the less important sequential cases, the read penalty is about 12%, for writes nearly 30%.

On the SPARCserver 630/670MP with 1.3 GB SCSI disks, the penalties for random access are slightly smaller: random reads lose about 5%, random writes about 7%; unfortunately, serial access loses 10% for reads and nearly 50% for writes. In this configuration, the loss of substantial write performance can be reduced by using the –r (balanced read) option,

74. *op. cit.*

driver provides new kernel services which provide support for very large file systems – up to 1 terabyte – and limited support for very large files (via the lseek2(2) system call). In general, most of the services provided by MetaDisk should be considered for their functionality rather than for potential performance gains, although in some cases performance can be increased through judicious use of the MetaDisk driver.

The functions provided by the MetaDisk can also be implemented in hardware or firmware, as some other vendors do. The advantage of performing these functions in hardware is that the main processor is not burdened with the computation associated with this functionality. However, the cpu cost is quite small (refer to Table 11.), and the software nature of the MetaDisk driver provides an important measure of flexibility. The MetaDisk driver implements all of its functions without regard to the specifics of the underlying disk drives. Used within reason, MetaDisk permits dissimilar types and sizes of disk drive to be operated together (*e.g.,* mirror a 1.3GB IPI onto a concatenated, striped pair of 669MB SCSI drives). Currently, all currently supported IPI, SCSI and SMD drives and controllers[72] are fully functional with the MetaDisk. This flexibility protects the user's investment in disk technology, where disk sizes and capabilities change frequently. Most hardware implementations require all constituent disks to be of the same type (sometimes identical disk sizes are also required).

An important performance note: after installing Online:DiskSuite 1.0, be certain to tune any file systems created on metapartitions. MetaDisk sets the maxcontig and rotdelay parameters to defaults which cause serious performance loss. This performance penalty is on the order of 60% on most file system operations. Specifically, use the command:

```
tunefs –a 7 –d 0 /fileSystem
```
[73]

These values are efficient for current disks (1.3GB SCSI or IPI and 424MB SCSI) with Solaris 1.0. The default values effectively prevent the file system from performing disk clustering, which forces substantial extra physical I/O to be done. (See section 2.4., *Disk Subsystems*.)

The following discussion of the MetaDisk driver presumes that any file systems mounted on a metapartition have been tuned in this or similar way. (If the partition is to be accessed via the raw disk interface, there is neither any need nor mechanism to perform such tuning, as the application has accepted the responsibility for optimizing access to the disk by using the raw interface.)

Use of the meta driver imposes additional burden on the processor(s), since the driver must perform various calculation and coordinate multiple physical I/O's. This load varies according to which operation is being done and which peripherals are involved. The results are summarized below in Table 11. When using more than one option, calculate the overhead for each option individually, then add the overhead to the basic operation cost. For example, the cpu required to process an 8KB random read from a striped file system which is mirrored with another stripped file system (using the –Sg options) is computed as follows. The basic

72. The 104, 207, 327, 424, 669 and 1.3GB SCSI drives, 280, 575, 688 and 892MB SMD drives, and 911MB, 1.0GB, and 1.3GB IPI drives are supported. All disk controllers that are supported with these drives are fully functional with MetaDisk.

73. Shuff, Pat. "Performance of MetaDisk on SPARCstation–2 and SPARCserver 600MP". SMCC Performance Engineering Internal Report, September 3, 1991. The file system is specified by naming metapartition (*e.g.,* /dev/md2f) upon which the file system is created.

very high disk utilization[70]. For diverse communities, this is clearly a resource which bears close monitoring.

2.10.5. BSD type 4.2 File Systems

The BSD 4.2 file systems take advantage of the physical disk's geometry by dividing a disk drive into *cylinder groups*. When files are created, an attempt is made to allocate them completely within a cylinder group. This strategy works well for the small files typical of traditional UNIX systems, by keeping the average seek distance very low – as noted earlier, the amount of time required to perform a seek is roughly proportional to the length of the seek. Over time,sectors are used and reclaimed in essentially random pattern. When a file doesn't fit in the cylinder group, is simply flows over into another one. Eventually, many files cross cylinder groups, with a corresponding performance loss. In this circumstance, the best way to reorganize the disk is to do a complete dump of the disk partition, recreate the file system, and finally restore the data (dumping to a temporary disk area, even over a network, can speed this up dramatically).

The optimizations incorporated into the file system cannot ordinarily optimize across disk partitions, even though all file systems may well be located on a single disk – for the simple reason that file systems do not cross partitions. This can be of some impact when laying out partitions on the disks. If two (or more) partitions are expected to be heavily accessed, put them on two different disks. Since accesses to those partitions is likely to be interleaved, the head will often be moving across many cylinders. The resultant degradation can be severe. This is likely to happen when a large disk is divided up into several partitions for administrative convenience; one very common – poor – configuration has two 1GB drives, one full of home directories, the other holding the server's various utility partitions: /, swap, /usr, /var/spool, and the group's application binaries. Since access to swap, /var/spool and applications all happen frequently, this disk will likely spend a lot of time seeking between partitions[71].

Rule XXIII. Configure only one "hot" partition per disk drive.

Rule XXIV. If one disk must provide both a heavily–used file system *and* swap space, configure a single partition *and place a swap file in the heavily–used partition*, rather than making partitions for both swap and file systems.

2.11. MetaDisk Logical–Level "Disk" Device Driver

The Online:DiskSuite product offers a high–level disk "device" driver which permits several useful operations on disk drives: mirroring, concatenation and striping. This driver, known as the MetaDisk driver, functions by being interposed between the user of the disk resource (generally the UFS file system or an application which accesses the raw disk interface) and the actual disk device driver. From such a position, it can duplicate I/O requests (for mirroring) or issue interleaved requests (for striping). In addition to these services, the MetaDisk

70. Stern, Hal and Brian Wong. "Impact of Network Load on NFS Server Performance". SMCC Performance Engineering Internal Report, September 20, 1991.

71. An unsupported disk activity monitoring package *dwtool*, including specially instrumented device drivers for disks and controllers, is expected to be available internally soon.

cache all of the accessed data, a quick check of the buffer cache hit rate may reveal a need to tune to a higher `nbuf`. This can be set as high as 255, representing 255 buffer pools of 8KB, or 2MB of meta–information. Acceptable values for the buffer cache hit rate vary, but clearly rates of less than about 50–60% warrant some attention.

2.10.4. Directory Name Lookup Cache (DNLC)

Another of the operating system's internal caches is known as the directory name lookup cache. It caches the names of resolved file system names, in order to reduce the amount of time involved in locating a file system entry. Without the DNLC, locating the file /usr/etc/dmesg requires four directory name lookups: one each for /, /usr, /usr/etc, and /usr/etc/dmesg. The DNLC caches file system information necessary to resolve paths without resorting to the file system meta–information and possibly to physical I/O. The DNLC is constructed in such a way that it has a maximum path length which it can resolve.

Like the buffer cache discussed above, the DNLC can be too small, resulting in a very similar situation: the user data is substantially or completely cached, yet the disk subsystem is excessively busy (re–)retrieving DNLC entries[69]. This can be particularly severe on NFS servers, where nearly 50% of all operations may involve the DNLC (since database servers usually operate on raw disk partitions, a minimal DNLC size is generally sufficient).

If a system seems to be doing "too much" disk I/O and the buffer cache hit rate is high, check the DNLC, which is reported by *vmstat(8) –s*. DNLC hit rates of less than 85% or so warrant attention.

The DNLC is based upon the setting of MAXUSERS in the kernel configuration file, specifically it is 80 + 17 times MAXUSERS. This formula seems to be fine, except that the default value for MAXUSERS is a paltry 8 in SunOS and only 16 in Solaris 1.0, leaving nearly any NFS server or multiuser machine with a DNLC operating at very low efficiency. Although the size of the DNLC can be patched into the kernel in the same way as the buffer cache size, the best way to set it is to increase MAXUSERS in the kernel configuration. For optimal NFS performance, [Mehta 91] recommends setting MAXUSERS to 64 for servers with 16–32 MB of memory, to 128 with 64–96 MB memory, and to 160 for larger systems. One of the reasons for increasing MAXUSERS instead of simply patching the DNLC size is that more file references typically is associated with larger consumption of other resources such as file and inode tables, which are also sized according to MAXUSERS.

> **Rule XXII. Configure approximately one MAXUSERS for each MB of main memory configured. Monitor DNLC hit rate and increase MAXUSERS if hit rate is below 80%.**

In general, the DNLC hit rate goes down the more files are accessed; for servers providing NFS service to large communities, a very large DNLC may be needed. A recent study at Sun, spreading a constant network–wide nhfsstone load over a larger and larger community of client workstations caused the DNLC hit rate to drop dramatically, eventually resulting in

69. Mehta, Varun, Maneesh Dhir, and Rajiv Khemani. "Networks and File Servers: A Performance Tuning Guide". Sun Microsystems white paper, December 1990.

MMU. Previously context switch time became expensive under heavy multiprogramming loads because of the necessity to spill an overflowing MMU into memory, consuming memory bandwidth and smashing the cache. On the SS600MP, however, this process is not necessary. Unfortunately, as this is written (at SunOS 4.1.2–Alpha–3.0), the cost of creating and breaking down virtual–to–physical memory mappings (a part of the context–switch process) is much higher than in previous releases and/or hardware. This topic is presently the subject of intense research.

2.10.3. Buffer Cache

Since the dark ages of UNIX, memory has been used to cache disk operations; at least since the Berkeley flavor of UNIX split off from the AT&T version at V6, this cache has been known as the buffer cache. In SunOS 3.x and in most Berkeley–derived implementations, the buffer cache was reserved out of main memory at boot time; SunOS reserved 10% of physical memory for this purpose. With the release of SunOS 4.0 and its single–level virtual memory model, the primary purpose of the buffer cache became obsolescent. In current releases, all user data stored in the file system is automatically cached in memory subject to the normal virtual memory replacement algorithms[66]. The buffer cache, however, was not removed from SunOS (or Solaris). Instead of reserving a large area for any and all disk I/O operations, Solaris reserves a much smaller area for cacheing file system meta–information: the contents of directories, inode entries, superblocks, cylinder group information, *etc.* This space is still known as the buffer cache, even though its specific function is rather different from the original buffer cache.

In Solaris 1.0 and SunOS 4.x it was allocated from the kernel map; in Solaris 1.0 it is allocated from the secondary kernel map, an area created primarily for the purpose of managing a much larger buffer cache than the original kernel map (which is limited to 16MB by the address space of some common chips, such as the AMD Lance Ethernet). In SunOS 4.x and Solaris 1.0, the size of the buffer cache (represented by the kernel variable `nbuf`) was determined by a complicated formula which nearly always left nbuf and the buffer cache far too small for efficient server operation. The result was that user data stored in the file system might be completely cached in memory on a large server, yet the system would resolutely be hammering on the disk drives to retrieve the same meta information over and over again – even though there might well be vast amounts of unused memory.[67]

In older releases (pre–Solaris 1.0), one had to patch the kernel variable `nbuf` to fairly large values (say, 0x70 on a server with 64MB of memory) to achieve good performance from the buffer cache. Beginning with Solaris 1.0, `nbuf` is dependent only upon the size of physical memory, and the size of the buffer cache is substantially larger than previously.

The kernel keeps statistics about the buffer cache, but these are not available from the normal kernel monitoring utilities. Only *statit(8)*[68] reports the buffer cache hit rate. If a server seems to be doing more disk I/O than expected, for example if physical memory is sufficient to

66. Gingell, Rob, *et. al.* ”Single–Level Virtual Memory Model in SunOS 4.0”. Sun Microsystems Internal Report, 1988.
67. Mehta, Varun and Rajiv Khemani. ”Tuning the SPARCserver 490 for Optimal Performance”. SMCC Performance Engineering Internal Report. February 4, 1991.
68. statit(8) is available internally from newstop.ebay; a SunNet Manager statit agent may also be available in the near future.

another, necessitating that the processes virtual memory mappings and cache context be moved (or effectively moved, in the case of the cache). Ordinarily, the round–robin scheduler would simply assign the next process on the run queue to the next available processor; however, in Solaris 1.0, an attempt is made to reschedule a process on the same processor as it last ran, if possible, in an attempt to minimize cache thrashing. Staying on the same processor is profitable because each processor has its own cache, whose content is a function of what memory that processor has referenced recently. When a process is switched to another processor, references to data resident in another cpu's cache imposes Mbus activity and consequent delays. The scheme used in Solaris 1.0.1 seems to work fairly well, keeping Mbus traffic to a minimum. Sometimes, in *infrequent* circumstances, performance can be improved by forcibly "sticking" a special process to a given processor, using the `TIOCGPAM` and `TIOCSPAM` ioctl(2) calls. This is known as processor affinity. An unsupported utility[63] is available internally to launch a process and its children onto a specific set of cpus.

☞ ***Caution! The processor affinity interface and capability are otherwise completely undocumented and are ➧ expected ➧ to change in future releases of Solaris!***

The use of the processor affinity utility has never been observed to provide more than approximately 5–6% in performance, although it has not been extensively tested or benchmarked (nor has it been observed to degrade performance by more than about 15%).

2.10.2.2. Context Switches and the Scheduler

The Solaris 1.0 scheduler is responsible for allocating user processes to the available physical processors. Depending upon the system's configured purpose, the scheduler can be tuned to slant toward overall system throughput or toward response time for individual processes. The scheduler is set to interrupt a process after a tunable period (the quantum). The default configuration is to interrupt 40 times per second, which means that each processor (in a 4–processor system) gets interrupted ten times per second. Sometimes, especially under heavy multiprogramming loads, it helps to set the interrupts to be more frequent. This increases response time (makes the system feel slower), but sometimes can improve the overall throughput of the system. The kernel variable `rr_ticks` can be patched patched, either with adb(1) or with an XView–based tuner[64]. The value defaults to 40; lower values tend to improve responsiveness and lower throughput. The interrupt rate can range from 10 per second to 160. Remember that the interrupts are equally distributed across all of the configured processors; a 4–processor system and a 2–processor system may behave very differently under the same workload with the same setting. We expect that tuning this parameter will have approximately the same effect on uniprocessor machines as on multiprocessors, but we haven't tried it!

☞ ***Caution! This tunable parameter is completely undocumented and may well disappear in future releases of Solaris!***[65]

The reason that this is effective appears to be the expense of a context switch on the SS600MP systems, despite the presence of many hardware contexts in the SPARC Reference

63 *oncpu(8),* available internally from newstop.ebay.
64 *xvtr(8),* available internally from newstop.ebay.
65 This may not be serious; the scheduler in Solaris 2.0 is a loadable kernel module, which can be replaced to handle this kind of tuning, if appropriate.

top group of four LEDs indicates which processors, if any, are executing in user code. The bottom group of LEDs indicates which processors, if any, are executing in system code. The middle group shows the system heartbeat, with a single LED lit, walking up and down the group. This is shown in Figure19.

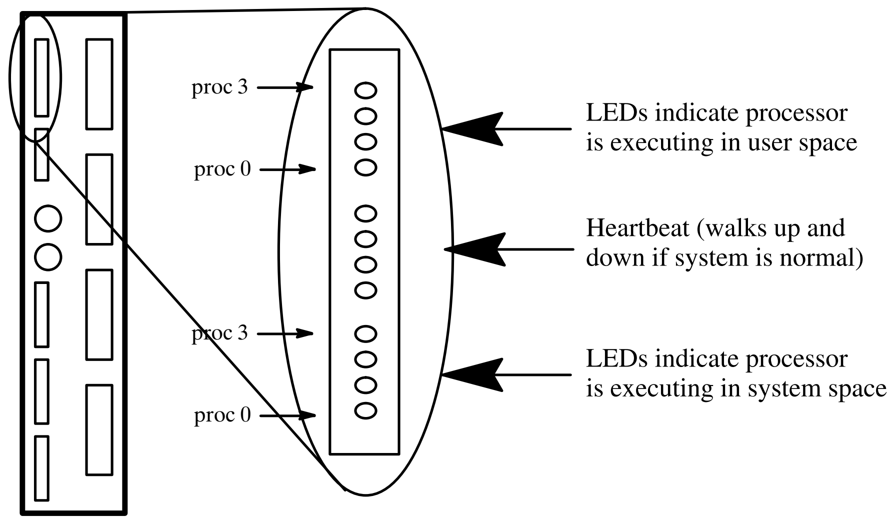


Figure 19. Map of LEDs on SS600MP processor complex board and their meaning.

2.10.2. The Solaris Scheduler

The UNIX scheduler is responsible for distributing processor complex resources fairly among the various competing processes. Since the processor complex resources are especially critical in the SPARCserver 600MP series, the behavior of the scheduler can have a significant impact on the system's overall performance.

2.10.2.1. Cache Flushing and the Scheduler

When a process is running on a given cpu, its memory reference pattern causes a memory image to be stored in the cache which is tightly bound to that processor. Clean (read–only) cache lines can be shared between the processors, but cache lines dirtied (written) by access from a processor will update *only* that processor's cache and invalidate any copies of that cache line which are stored in the caches of other cpus. When a process is moved to another cpu, that cpu's cache is generally organized to support the address space of a completely unrelated process, necessitating substantial overhead as the switched process proceeds to fill the cache on its new processor. When the system is cache–limited, such as the SS600MP when equipped with the 64KB caches in the Ross 6002 modules, this overhead can be excessive, effectively negating the write–back cache organization and imposing very high demands on the memory bus. This is fundamental to the design of any shared memory multiprocessor with cache memory.

The Solaris 1.0 scheduler is implemented in such a way as to tend to stick a process to a specific processor. This minimizes the number of times a process is moved from one cpu to

nel at a time. Because there is just one lock, systems which spend most of their time executing in system code are effectively limited to less than the throughput of a single processor. (Remember that it is nearly always possible to configure enough of any other resource to make the processor the bottleneck.)

There are a few exceptions to this organization. Under certain restricted conditions, it is possible to release the kernel lock while performing kernel activities. One of the most expensive kernel tasks is a memory copy via the kernel bcopy(), kcopy() or uiomove() routines. All of these routines are highly optimized, but the typical bcopy associated with I/O is 2KB–8KB, which can take some noticeable number of *milli*seconds (on a 28.5–MIP machine, one millisecond is 28,500 instructions!). For the duration of the bcopy, there is nothing else that can be done in the kernel other than to service other interrupts; it is safe to release the kernel lock, permitting other processors to enter the kernel while the current processor finishes the bcopy.

The most dramatic instance where a bcopy can release the lock is the one in the PrestoServe driver[60]. This single instance, however, is sufficient to provide a 7–8% improvement in NFS performance in multiprocessor mode on the SS600MP systems (this is 7% with the Sbus PrestoServe and 10–11% with the VME PrestoServe). Recall that on uniprocessor systems, the PrestoServe imposes a 5 to 10% cpu overhead due to bcopy overhead. However, on a multiprocessor, the effect is to permit one processor to execute the bcopy while another one continues on to execute other code in the NFS, disk or network kernel code.

At this writing, there are eight instances where an expensive kernel operation can release the kernel lock. The net result is an increase in kernel efficiency of approximately 35%. Even though there is still only one lock implemented, this clever workaround essentially has the effect of a multiple locks, and provides a glimpse of how the multithreaded, fully symmetric Solaris 2.0 will perform[61]. The degree to which multiple processors can safely execute within the kernel is called the parallelism efficiency. At present this is has not been determined for either Solaris 1.0 or 2.0. VMS 5.2, known for its excellent symmetry, has a parallel efficiency of about 90%.

Because most applications perform some significant portion of their time executing in the system, the kernel lock limits the extent to which a multiprocessor system improves performance. The amount of system time can be determined in several ways. The *vmstat* utility shows the total amount of time being spent in the system; this is most effective when measured on a uniprocessor, in an attempt to estimate the proportion of speedup on a Solaris 1.0 multiprocessor. If the system is already running on a multiprocessor, the *kmeter* utility indicates the proportion of time spent in system code[62]. In addition, kmeter indicates the amount of time spent in “spin” mode – when a processor is blocked waiting to get into the kernel; sometimes this can be very interesting information! The unsupported utility *statit* provides tabulated form of user, system, and spin time. Finally, in the absence of a display, the LEDs on the cpu board provide an indication of what is going on in the processor complex. The

60. While there are many calls to bcopy in the kernel (it is the most heavily used routine in the Solaris kernel), the unprotected bcopy may only be called when the kernel can guarantee that there can be no other kernel access *on behalf of the current process* for the duration of the bcopy. This is relatively infrequent.

61. Solaris 2.0 will have approximately 130 discrete kernel locks, in addition to these (and possibly other) released–lock situations.

62. *kmeter* is available from newstop.ebay.

system and enable it only for those partitions for which it is appropriate. For example, a server which supplies virtual memory to a population of diskless clients stores NFS–accessible data on dual–ported disks, it is reasonable to configure the PrestoServe to operate on the client's root and swap partitions and *not* to affect the data partitions.

2.9. Serial Line Connectivity

Serial line connectivity can be accomplished in a number of ways. Sun offers both the 8–port Sbus Asynchronous Line Multiplexor or the 16–port VME ALM–2. Choosing between these is a matter of cost and required expansion capability. For servers which can accept either, it is probably safer to configure the VME version in SS670 and SS690 systems, since they have many free VME slots and relatively few Sbus slots. Additionally, the SS690 has physical provisions for neatly mounting the interface panel of an ALM–2. From a performance standpoint, there is little to choose, since they both operate in the same interrupt–per–character mode and the bus traffic generated by either is vanishingly small.

For situations in which more serial lines must be supported than are physically configurable in the system, third–party terminal servers such as those offered by Xylogics (among others) may be the best choice. These devices are essentially serial–to–ethernet concentrators. They have several advantages over direct serial line multiplexors, although they cost rather more. First, from a performance standpoint, they impose less overhead on the host system, because they can collect multiple characters together into a single packet, resulting in fewer interrupts. Second, the terminal servers can be obtained in very large configurations – 256 serial ports are reasonably common. Finally, because they connect to the hosts via Ethernet, they save bus slots.

> **Rule XX. Configure the VME ALM–2 over the Sbus Serial/Parallel interface, due to availability of VME slots.**
>
> **Rule XXI. Configure ethernet terminal servers if more than 30–50 serial lines are required, due to the terminal server's lower overhead.**

One other option has recently become available: serial line multiplexors which connect to systems via SCSI. Although the authors have not benchmarked these items[59], they appear to offer an interesting mechanism for working around terminal connectivity issues. Due to their SCSI architecture and off–board processors, they are likely to offload a reasonable amount of interrupt processing that Sun's ALMs pass on to the main cpu.

2.10. Kernel and Operating System Resources

So far we have talked a lot about hardware. But the hardware is controlled and shielded by the operating system, and respecting the operating system's limitations is just as important than obtaining the fastest hardware. We will look at a few of the most important constructs in or associated with Solaris.

2.10.1. The Kernel Lock

Probably the most limiting item in the Solaris 1.0.1 kernel is the kernel lock. This is the construct which enforces the restriction that only one processor can be executing in the ker-

59. One notable offering is from Central Data, which offers 4–, 8– and 16–port expansion per SCSI target address. One of the 16–port items has an AMD29000 RISC processor for interrupt processing!

fer must be initiated by either the Sbus target or the main cpu. For this reason, an Sbus PrestoServe must be used in the circumstance that both SCSI and IPI disks are connected to the same system (which works, although it is not officially supported).

In higher–availability configuration, the use of PrestoServe must be carefully considered. There are two options which provide higher disk availability: Online:DiskSuite and the Dual–Port Disk kit.

The PrestoServe may be used in combination with the striping and concatenation options of DiskSuite without any problems, logical or operational. However, the combination of PrestoServe and disk mirroring option of DiskSuite must be chosen on a case–by–case basis. The goal of disk mirroring is to provide higher data availability in the event of a disk drive failure by keeping two copies of a disk. In the process of accelerating synchronous writes, the PrestoServe works to opposite purposes. If a synchronous write is committed to a mirrored pair in a Presto–equipped system, both copies will be intercepted an committed first to the NVRAM in the PrestoServe. Both copies then are subject to the failure of the PrestoServe board. This perverts the intent of mirroring, but only to a limited degree. First, the PrestoServe is much less to fail than the disk controllers, because it is a *much* simpler piece of equipment. And because it is completely solid–state, it is orders of magnitude less likely to fail than the disk drive itself. Finally, the matter of a PrestoServe as being in the critical dual–pathed disk subsystem is one of philosophy. The PrestoServe resides in the bus with the disk controllers, and there is only a single bus, a single memory subsystem, and (for failure avoidance purposes) only a single cpu. If the PrestoServe is viewed as a part of the cpu complex rather than in the disk subsystem, it makes sense to configure a PrestoServe in applications which will commit many synchronous writes to a mirrored partition.

Rule XVII. Configuration of PrestoServe with disk striping or concatenation if necessary. A striped or concatenated disk partition may be treated simply as if it were a normal disk partition.

Rule XVIII. Configure PrestoServe in combination with mirrored disks if the customer can reconcile the philosophical issues. Note that the PrestoServe will be twice as important on a mirrored partition because twice as many writes must be committed!

On the other hand, the use of a PrestoServe option cannot be reconciled with the use of dual–ported disks. The problem here is that there are two systems involved, and if a system equipped with a PrestoServe fails, it will contain information which is logically “on” its disk(s) in the PrestoServe’s NVRAM. Because this information is not on the media of the dual–ported disk, it is not available to the fail–over system. Even worse, if the disk contains file systems, it is the most critical information that is missing: inodes, superblock information, indirect blocks – which is why the writes were requested synchronously in the first place. For this reason, PrestoServe is incompatible with dual–ported disks.

Rule XIX. Do *not* configure PrestoServe to operate on dual–ported disks.

It is possible that some applications will require the use of a PrestoServe, while other applications on the same machine are incompatible with the use of that option if the machine is to be a multipurpose server. In these cases, it is possible to configure the PrestoServe into the

cepts such writes and commits them to non–volatile battery–backed memory, temporarily saving the physical time associated with actually committing the write to a disk. The PrestoServe device then posts the precommitted writes to the disk when it is convenient to do so.

From an NFS client's point of view, the net effect is to turn a non–cacheable synchronous write into a cacheable, asynchronous one. From the server's perspective, the effect is that when many synchronous writes are concentrated to a relatively small area of a disk, most of the physical writes can be avoided completely, since the PrestoServe driver typically flushes its data to the disk subsystem at intervals sufficient to capture a number of write requests. UNIX performs all modifications to a directory synchronously, so any process which creates and/or deletes many files benefits from a PrestoServe. Servers and workstations that support CASE environments nearly always benefit substantially from a PrestoServe. Sun's NSE is a spectacular example! All NSE servers should be equipped with PrestoServe[58]. Also, some electronic mail environments store each message in a separate file (USENET News is a good example of this), causing very heavy synchronous disk traffic. Under these circumstances, the PrestoServe can dramatically lower the utilization of a disk. There is a slight cpu overhead for using a PrestoServe, primarily attributable to the expense of an addition data copy.

Rule XIV. Configure PrestoServe any time nfs writes exceed 6%.

Rule XV. Configure PrestoServe if you are using a database management system which places its data in a UNIX file system.

Rule XVI. Configure PrestoServe if your application creates and/or destroys many small files (e.g., store–and–forwarding email systems).

Sun now offers both Sbus and VME versions of the PrestoServe "NFS" accelerator. The two versions differ primarily in form factor. When used in the recommended configurations, the biggest performance difference is that the VME version imposes slightly more processor overhead than the Sbus version (about 5% for Sbus, about 10% for VME). On a uniprocessor system, this is a clear limitation on when a PrestoServe can be configured: if the processor is already be saturated, the PrestoServe degrades performance. However, recent experiments with the SS600MP systems appear to have weaseled around this – see section 2.10.1. on the Solaris 1.0 kernel lock.

Sun recommends that a PrestoServe be configured on the same I/O bus as the system's disks. Since the PrestoServe can master a bus and initiate data transfers, it is able to keep cpu overhead to a minimum. Accordingly, a SPARCserver 690MP with its VME IPI controllers should use a VME PrestoServe, and other machines (630MP, 670MP, SS2) with SCSI disks connected via Sbus host adapters should use an Sbus PrestoServe. If for some reason there are insufficient VME slots in an SS690, an Sbus PrestoServe can be used. In such a configuration, the PrestoServe's ability to provide fast response times is reduced (it has to copy data an extra time), and the cpu overhead required to support such a configuration is about double that required for normal configurations. Unfortunately, the reverse configuration (Sbus SCSI disks with VME PrestoServe) is not possible: a VME bus master cannot initiate an independent transfer to an Sbus target (and an attempt to do so hangs the system). Such a trans-

58 NSE apparently creates thousands of one–byte files in various stages of its processing. When NSE was recently implemented throughout engineering, the cost of some remote (over INR at less than T1 rates) operations dropped from nearly two hours to about five minutes when a PrestoServe was enabled on the server!

2.6. Internetworking: High Speed Serial Interface (HSI/VME, HSI/S)

Both HSI boards permit the transmission of high–speed (up to 1.5Mb/sec) serial internetwork connections, and are typically found in servers which are used as communications servers as well as some other function, such as file or DBMS service. The HSI/VME board permits connection of two T1 serial connections, while the HSI/S accommodates four T1 interfaces, providing a total of 3.5Mb/sec. across the four ports. Like other network interfaces, the handling of protocol stacks and routing are the primary consumers of server resources. The main processor complex is responsible for these functions. The HSI/VME is known to consume about 3 MIPS per T1 line for protocol handling and routing at full T1 load. The HSI/S is a new board and has not been formally benchmarked, but preliminary experience with the HSI/S has shown that is is more efficient than the HSI/VME by 15–20%[56]. For now, expect to consume about 2.5 MIPS per T1 line on a HSI/VME port, and about 2 MIPS on an HSI/S port.

2.7. IBM/370 Data Interchange: Channel–to–Channel Adapter (VME)

Sun offers a Channel–to–Channel Adapter option for connecting IBM–compatible mainframes to Sun networks at high speeds. The CTCA is a two–board VME set which can sustain transfer rates of 3MB/sec. In the past, this has been the option of choice for high speed data transfer, but recent product introductions by both Sun and IBM have combined to make this a much less attractive alternative. IBM mainframes can now connect to Ethernet, FDDI, and Token Ring networks, all of which are much lower cost and complexity. Because FDDI has the potential to transfer speeds well in excess of the existing CTCA (reasonable expectations are 4–8 MB/sec with sufficiently powerful cpus, available on the Sun side with Solaris 2.0 or with a cpu upgrade in the SPARCserver 600), FDDI should be considered the high–performance interconnect of choice. A lower–cost option is the use of a 16Mb/sec token ring, which can be expected to transfer around 1MB/sec.

2.8. PrestoServe (VME and Sbus)

The PrestoServe accelerator is a VME or Sbus board which contains 1MB[57] of battery–backed non–volatile memory. The device driver ensures that data committed to the non–volatile memory are never lost (unless the battery is physically removed from the board). In principle, non–volatile RAM could be built onto the main processor board, rather than placing it out on a peripheral bus (DEC provides this on the DECstation 5500).

The PrestoServe improves disk performance by intercepting and cacheing synchronous writes, *whether or not they are remote operations generated via NFS.* Because NFS is a stateless protocol, the server must guarantee the viability and correctness of any writes; thus all remote writes must be performed synchronously. This involves sending the request(s) down through the device driver and then waiting for the hardware to physically perform the I/O and return an indication that the write was competed successfully. The PrestoServe inter-

56. This is apparently due to lower bus communication overhead and limited on–board specialized packet processing, although the HSI/S does not have a generalized processor to handle protocol stacks.

57. Note that although Sun OEMs the PrestoServe boards from Legato, Legato boards are not necessarily equivalent. The Legato Sbus board has only 512KB of memory.

2.5.4. FDDI/DX (VME)

The FDDI/DX is a 9U VME board with a Class–A FDDI interface. Class A means that the one interface board connects to two separate counter–rotating rings; the rings are used to provide a failover mechanism in the event of a ring break. Wiring a Class–A network takes the physical appearance of appearance of a ring: the boards are connected by a ring of fiber. No hardware other than the fiber connections and the interface boards is required to build a complete network.

In addition to the two fiber–optic interfaces, the board has a dedicated 68020 and buffer memory. The 68020's job is to manage the various input and output packet buffers. Like Ethernet and token ring, the primary bottlenecks in managing FDDI transactions are getting the data onto and off the interfaces and especially processing the protocol stack. In data–intensive NFS environments, the FDDI/DX provides about three times the network bandwidth of an Ethernet. This directly attributable to the maximum transmission unit of 4500 bytes, which is three times that of Ethernet. When data transmission is primarily data (for example, when saving a 70MB financial model, or loading a 450MB geologic data set), virtually all data going over the network is user data, which arrives in one third as many packets as on Ethernet.

Because the FDDI/DX does not provide the 10x improvement in network performance normally associated with FDDI, a number of other vendors have been promoting the Sun board as "slow". It is worth pointing out that other boards which have no on–board protocol processing deliver approximately the same performance as the FDDI/DX, and for the same reasons. Third–party boards which do include protocol processing may well compare to the FDDI/DX in the same way as the Network CoProcessor compares to a built–in Ethernet: its overhead is lower, latency is lower (better), but given a relatively slow processor its overall throughput is likely to be limited[55].

2.5.5. FDDI/S (Sbus)

Sun also offers an Sbus Class–B FDDI interface board. Class–B interface type means that this board connects to only one ring. Wiring normally appears to be a star configuration, although each branch of the star is actually a dual fiber carrying the ring to and from the hub. In Class–B networks, the normal configuration includes a concentrator hub in a central wire closet. This arrangement is very similar to current twisted–pair Ethernet installations. Don't forget the concentrator hub when specifying a configuration!

The Sbus FDDI interface has not been formally benchmarked yet. However, it is likely to perform at a level comparable to that of the FDDI/DX, again because most of the expense of using the network is spent processing the protocol stacks. The FDDI/S has no on–board protocol processing engine. The earlier comments about comparisons with third–party FDDI interfaces apply equally to the FDDI/S as to the FDDI/DX.

55. As a guess, it probably takes an AMD29000 or another fast RISC processor to make an FDDI interface very fast. Sight unbenchmarked, virtually any 68020– or 68030–based interface isn't likely to keep up with protocol processing on an FDDI network. The NC400 is already limited by a 16Mhz 68020 on a single Ethernet.

overall performance. In this case, although the AMD Lance interface is capable of sustaining 400 NFSops/sec (as demonstrated by most Sun systems), the Network CoProcessor is limited by its protocol processor to about 290 NFSops/sec. Although Sun has not rigorously studied the Network CoProcessor's non–NFS performance, observation indicates that its typical TCP/IP performance is also limited to lower levels than the built–in Lance interfaces, with the accompanying reduction in cpu requirements and request latency.

Rule XII. Configure a Network CoProcessor for highest performance NFS, or when the server performs both NFS and other processing.

Rule XIII. Configure networks connected to a Network CoProcessor for a maximum of 290 NFSops/sec.

2.5.3. Token Ring (4Mb, 16Mb)

Sun offers a token–ring network interface as an alternative for customers who prefer this alternative networking media. There are two forms of token–ring adapter available, 4Mb/sec and 16Mb/sec. There are several differences between the two variants, in addition to the obvious difference in the raw transmission speed of the two networks. Specifically, stations utilizing the 16Mb version have the ability to put more than one packet onto the ring each time it obtains the token. This enables a server or a heavily–loaded client to transmit more data each time it has control of the ring, making bulk transmission more efficient. In addition, the 16Mb card has more intelligent firmware onboard for processing TCP, IP and UDP headers (but not NFS), further enhancing performance. In some ways, the 16Mb token ring card is similar to the Network CoProcessor, in that it performs some off–board protocol processing. Of course, the token also goes around the ring more quickly in the 16Mb version than it does in the 4Mb version.

Studies using the Nhfsstone benchmark have shown that a 4Mb token ring provides responsiveness (16–40ms latency) similar to that of Ethernet, but that it permits roughly half as much data to be transferred – approximately 200 NFSops/sec. (See table 2.) On the other hand, the 16Mb ring presently offers performance (400 NFSops/sec) and responsiveness (latency between 14ms and 40ms) which are nearly identical to that of an Ethernet, at least for small numbers of clients. When the network is heavily loaded, the 16Mb token ring degrades more gracefully than an Ethernet. This due to the token–passing design: even if everyone on the ring wants the token, only one station has the right to attempt to transmit. Ethernet, on the other hand, has no such governing mechanism, and as a consequence, an Ethernet tends to degrade very abruptly under heavy load.

Interestingly, although the token ring boards have on–board TCP, IP and UDP decoding, they do not appear to impose lower overhead than the built–in Sun Ethernet interfaces, which do not have intelligent processors. In addition, the maximum transmission unit is 30% larger, which would imply less protocol overhead. All of this implies that there is a different bottleneck (perhaps in data copy management) that is limiting token ring performance. In several weeks of testing we were unable to drive the ring loading up over about 41% (as measured by a Network General Sniffer), even with 18 systems on the ring. This is also an indication of a bottleneck somewhere in the token ring subsystem, since Suns can easily drive an Ethernet to over 95% utilization.

2.5.1. Ethernet

Current ethernet interfaces are capable of sustaining 220–400 NFSops per second. Interfaces built around the Intel 80385 Ethernet chip can sustain around 230 NFSops/sec, those built around the AMD Lance ethernet chip can sustain 400 NFSops/sec. At these rates, the Ethernet itself is approximately 35% utilized, close the to practical limit. More active stations stations on such a network are likely to interfere with each other's ability to transmit, resulting in significant delays.

2.5.2. Network CoProcessor (VME) Ethernet Interface

Sun offers the Interphase Network CoProcessor intelligent Ethernet interface as an option for high–performance, low–overhead network–oriented applications. The Network CoProcessor is built around the same AMD Lance Ethernet chip used in all of Sun's Ethernet products, but adds a MC68020 processor whose responsibility is to recognize incoming TCP, UDP, IP and NFS packet headers and decode them. Outgoing packets are similarly encoded by the protocol processor. NFS requests are passed directly to and from the Network CoProcessor device driver to the NFS handling code. By relieving the main processor(s) of the protocol–processing code, the Network CoProcessor makes additional processor power available. The Network CoProcessor reduces the utilization of the main cpu by approximately 40% in a pure NFS environment. In addition, the Network CoProcessor substantially reduces the latency involved in network operations – by about 40% for NFS operations. Because a substantial proportion of NFS processing is done on the Network CoProcessor, it accelerates NFS requests by the largest amount. Nonetheless, servers which handle heavy Ethernet loads benefit from an Network CoProcessor. This is shown in Figure 18.

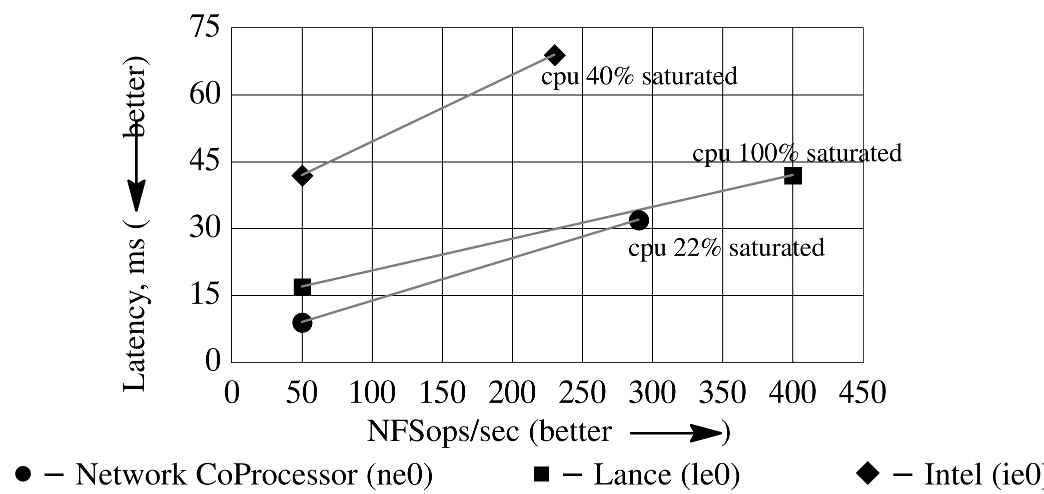


Figure 18. Latencies of various Ethernet interfaces, as measured by nhfsstone. The Network CoProcessor displays the lowest latencies at all performance levels attainable by the Network CoProcessor. The Network CoProcessor's onboard 68020 limits the Network CoProcessor's throughput to 290 ops/sec. Note that the main cpu is the limitation with the Lance interface, while the Network CoProcessor's cpu is the limitation with the Network CoProcessor interface. The bottleneck with the ie0 interface is the Intel 80385 itself.

One of the disadvantages of the Network CoProcessor is that it suffers from the same limitation as most special–purpose processors: its relatively old processor limits the product's

power consumption. Most important, they can spin at higher and higher rates – 5400 RPM is common in the smaller sizes, and even higher speeds are under development. When combined with multiple head assemblies, small–form–factor drives will provide substantial performance improvements over the next year or two.

2.5. Network Interfaces

Sun's local area network offerings include Ethernet (by default), token ring, and FDDI. Having the right network configuration is crucial to getting the true potential out of a server. Many, many workstation users have complained about the "performance of the server", only to discover that a jammed network was the bottleneck, not the server. Comparing the networks at a user's level is easy, since the higher–level software layers – TCP, UDP, IP, NFS and the like – are the same over all of the media. The relevant differences are the speed at which things travel, the method of interface, and particularly the degradation characteristics of each media.

One thing to keep remember when comparing network types is that protocol overhead is quite substantial. Applications which transfer many small packets will usually gain little performance by migrating to a high–performance network, because most of the time involved in such network activity is associated with processing the software layers of the protocol stack. NFS is especially susceptible to this, since more than 70% of all NFS requests are typically attribute lookups and other small packets. Recent trends in NFS usage (and NFS benchmarks), seem to point toward more data–intensive applications and longer packets. Database applications tend to send relatively long packets already; they too are tending toward larger amounts of data.

Rule XI. Configure Ethernet for most applications, except: (a) when the previously–installed networking is token ring, or (b) most network traffic consists of large data packets (as opposed to attribute fetches), or (c) a single network must service over a hundred workstations/clients. Configure FDDI in these cases.

	Media Speed	Effective Speed	Max Trans. Unit	Maximum Length
Ethernet	10 Mb/sec	440 KB/sec	1500 bytes	500 meters
4 Mb token ring	4 Mb/sec	320 KB/sec	2048 bytes	100 meters
16 Mb token ring	16 Mb/sec	500 KB/sec	2048 bytes	100 meters
FDDI	100 Mb/sec	900 KB/sec	4500 bytes	62.5 KM
T1 line via HSI	1.5 Mb/sec	1.5 Mb/sec	N/A	N/A
T3 line	45 Mb/sec	45 Mb/sec	N/A	N/A

Table 10. Characteristics of various Sun's various network media. Note that FDDI's effective speed today is limited by the speed at which the protocol stack can be traversed. When faster cpus become available, or when the multithreaded Solaris 2.0 makes additional cpus available, this effective speed should increase greatly.

slower than ones outside of them; cylinders are grouped into *zones*, which classify cylinders by their performance and capacity. Moving the more frequently demanded data to the fastest zones has the dual effect of permitting the data to be read faster and causing fewer seeks (because more data fits into a cylinder). Zone information is shown in Table 9.

	1.3 GB SCSI or IPI		424MB SCSI	
Zone	Start Cyl	track size	Start Cyl	track size
0	0	78		
1	626	78		
2	701	76		
3	801	74		
4	926	72		
5	1051	72		
6	1176	70		
7	1301	68		
8	1401	66		
9	1501	64		
10	1601	62		
11	1801	60		
12	1901	58		
13	2001	58		

Table 9. Zones on 1.3GB IPI, 1.3GB SCSI, and 424MB SCSI disks.

Rule X. Configure the most heavily–used partitions in the fast zones of disks. Consider striping fast zones together (see Section 2.11.2.) for high performance applications requiring larger partitions.

It is worth noting that disk drives are not typically designed with a specific performance–to–storage ratio in mind. Generally, the design of disk drives is driven by the currently–available technology for packing as many bits into as few dollars as possible. This is wonderful for storage capacity, but configuring for performance often results in far too much disk space or (more often) inadequate performance for the right amount of disk.

Disk drive technology is advancing rapidly. Connor Peripherals, for example, is developing a line of disk drives with two independent disk arms on a single platter. These drives are not like the CDC/Imprimis 9722 – both disk arms can be seeking and/or transferring independently. In effect, it’s two disk drives in a single package. Additionally, smaller and smaller physical platter sizes are becoming available: 2.5” and even 1.8” disks are becoming available or are in advanced stages of development. The advantages of these drives, in addition to the obvious packaging advantages, are that their moving parts are smaller, resulting in less

Another factor affecting disk performance is where data is physically located on the disk platters. Each disk “drive” is actually a series of individual disk platters stacked together, all rotating around a common center spindle. Both the top and bottom surface of each platter are used for data storage. Some drives use one or more surfaces for recording control information. Data is actually read and written by “heads” mounted on a moving arm, which moves radially across the disk platters.

All of this adds up to a geometry in which the data which is most quickly available is that which is located on the same “ring” as the current location of the read/write head(s), since no seek is involved. Each ring is called a *track*, which is divided into 512–byte *sectors*. The collection of tracks on each surface with the same radial address is called a *cylinder*. Since the activation of the read/write head(s) is electronic, access to data which is in the same cylinder as the head is also very fast. Although there is one one read/write head for every surface, normally only one head is able to transfer data at a time. The notable exception is the 911MB IPI drive, which has sufficient electronics to enable two heads to be simultaneously active. The second active head provides very fast transfer rates *as long as the both heads are trying to transfer to or from the same cylinder. If a seek is required, before the next operation can begin, the second head is of little use.*

The method used to encode the data onto the disk media can also have a performance impact. Most inexpensive disks (less than about $20K/drive) use one of three encoding mechanisms: MFM, RLL, and ZBR.

Modified Frequency Modulation (MFM) drives are by far the oldest, and are normally connected via an interface known as the ST506. They are the least expensive and provide the least performance. MFM disks are common in the low end of the PC marketplace: the ST506 interface is limited to 0.625 MB/sec transfer rate. Sun has not offered an MFM–format disk since the 71MB disk was discontinued in 1988 (the Sun 71MB disk used an ESDI interface at up to 1.2MB/sec, rather than the ST506).

Most current disks use Run Length Limited (RLL) encoding. Disks using RLL offer about twice the performance of an otherwise equivalent MFM drive, and are interfaced with a variety of techniques, including ST506, ESDI, SCSI and IPI.

MFM and RLL drives store data with *constant radial density.* This means that the same amount of data is recorded on the outermost cylinder as on the innermost cylinder, despite the fact that the outer cylinder is larger than the inner one. This is done to simplify the read/write head: data travels past the head at a constant rate, regardless of the head’s radial position. Under these arrangements, the head need be equipped to read data at only a single speed.

Zone Bit Recording (ZBR) is the most recent innovation, and is the encoding mechanism used in Sun’s latest disk offerings, the 424MB SCSI and both 1.3GB drives. The new ZBR drives record in *constant linear density.* Data is encoded at the same density on all cylinders. As a result, more data is stored in the outer cylinders than in the inner cylinders – about 50% more. Performance is affected because the disk platter always spins at the same speed, so 50% more data flies under the head in a given time period when the arm is positioned over the outside cylinders as when the arm is over an inside cylinder. The read/write heads are then configured to transfer data at variable rates. On current drives, the outer cylinders are about 50% faster than the inner cylinders. Cylinders toward the inside of the drive are smaller and

is specified by the vendor; this is useful information, since the UNIX file system arranges so that most seeks are quite short[52].

Both the maximum data transfer rate and the rotational latency are primarily dependent upon the rotational speed of the disk platter. Most existing drives rotate at 3600 RPM. Newer drives spin faster, meaning that data arrives under the heads more quickly, and that once found, the data is transferred to the read/write heads at higher rates. In some cases, notably with Sun's 911MB IPI drive, two read/write heads can be active simultaneously, providing for very high *serial* data rates. See Figure 17.

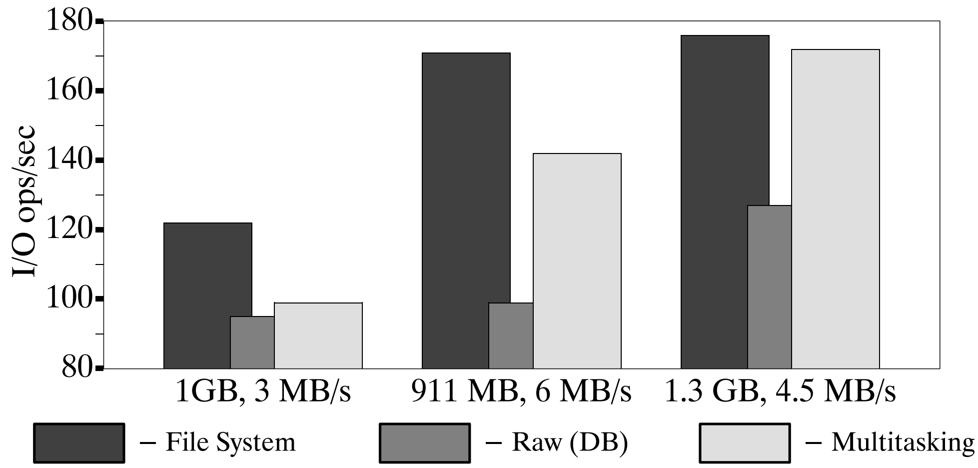


Figure 17. Relative IPI disk performance. File system IOPS are 8KB; raw IOPS are 2KB. These rates were achieved on a SPARCserver 490.

Depending upon the disk, controller/host adapter, and bus interface, the amount of time spent on any single category may increase or decrease. For example, Sun's IPI drives have rotational position sensing. The IPI controller uses this knowledge to prioritize data requests for sectors which have the least rotational latency. This is particularly effective in environments where there are many accesses to disparate areas of the disk – as is typical in high–volume servers. Both SCSI and IPI controllers have information as to the radial location of the read/write arms and use this information to reorder a sequence of pending I/Os to minimize seeks[53].

The random I/O rates listed above are the maximum sustainable rates, presuming 100% utilization. In reality, this doesn't ever happen. Patterson and Hennesey suggest that disks should be seeking no more than 60% of the time[54] ; and studies at Sun have suggested a disk utilization (such as reported by iostat –D) should average less than 60–70%. The last column in the table shows the 65% utilization figures.

52. In some informal investigations at Sun, more than half of all seeks are less than five cylinders. The methodology used to obtain this data is decidedly less than scientific, although they agree with results previously reported by [Patterson 1990] and others.

53. Due to SCSI's simplified disk layout (SCSI disks appear to be a linear progression of data blocks, while SMD and IPI drives present a three–dimensional geometry), mapped–out replacement blocks and other internal discontinuities mean that upper layers of the SCSI software can be mislead on infrequent occasions. This doesn't happen under IPI or SMD.

54. *op. cit, p. 545.*

As with SCSI host adapters, current disk packaging places constraints on how disks can be organized on the string controllers. Specifically, each disk tray on the SS690MP has exactly two IPI channels. Each tray must be therefore configured to one string controller (the second channel is for the dual–port option and cannot be used to increase performance). This limits a system without an expansion cabinet to only two controllers. Under some circumstances an expansion cabinet may be advisable in order to gain enough disk trays and disk controllers. This was not true of Sun's previous IPI disk offerings.

> **Rule IX. Configure up to three disks per ISP–80 controller when doing primarily random I/O. When configuring for primarily serial I/O, configure only a single active disk per controller. Additional drives per controller will not increase performance. Note that this is the *opposite* of the rule for configuring SCSI.**

2.4.4. Disk Drives

Disk drives can also be bottlenecks, especially under conditions which are heavily oriented toward random I/O – typical of server environments. To understand the limitations of a disk, let's look at the average time necessary to perform a physical disk I/O. There are four main components of a physical I/O: seek time, rotational latency, data transfer (from the disk platter to the read/write head), and controller overhead. In Sun's current product line offering, these statistics are listed in Table 16.

Disk Size	Interface	Format	Cache Size, KB	Ave. Seek	Rot. Latency	Xfer Speed, MB/sec	Typical 8K Xfer	Max Rnd I/O rate	65% of Capacity
104 MB	SCSI	RLL,2	8	23 ms	8.3 ms	1.0	23.7 ms	42/sec	27/sec
207 MB	SCSI	RLL,2	8	18 ms	8.3 ms	1.2	20.8 ms	48/sec	31/sec
424 MB	SCSI	ZBR	64	14 ms	6.8 ms	2.2–3.0	12.7 ms	78/sec	50/sec
669 MB	SCSI	RLL,2	8	16 ms	8.3 ms	1.8	17.9 ms	56/sec	36/sec
1.3 GB	SCSI	ZBR	192/48	11 ms	5.53 ms	3.0–4.5	10.9 ms	92/sec	60/sec
911 MB	IPI	RLL,2	N/A	15 ms	8.3 ms	6.0	14.5 ms	69/sec	45/sec
1.0 GB	IPI	RLL,2	N/A	15 ms	8.3 ms	3.0	14.9 ms	67/sec	43/sec
1.3 GB	IPI	ZBR	N/A	11 ms	5.53 ms	3.0–4.5	10.9 ms	92/sec	60/sec

Table 8. Characteristics of current and recent Sun disk drives. The column labeled "typical 8K transfer" is the typical time required to perform a single 8K transfer, assuming a controller cache miss. Note that although the IPI drives do not have embedded disk controllers and hence no caches, the IPI–2 string controller has a 1024K cache to devote to the entire string.

Disk manufacturers generally quote the "average seek time" (the sum of all possible seek lengths, divided by the number of possible seeks). For capacity planning purposes, however, it is more appropriate to take a quarter to a third of this figure as the "typical" seek time (the 1/3rd figure was used to compute the above table)[51]. Sometimes the track–to–track seek time

51. Patterson and Hennesey. [1990]. *Computer Architecture: a Quantitative Approach.* p. 516.

2.4.3. IPI–2 String Controller: the ISP–80

The ISP–80 IPI string controller fulfills the same logical combined functions as the host adapter and the embedded disk controllers in a SCSI system: it provides a nexus point on the I/O bus (VME in this case) for a string of disk devices. However, it differs in some points of operation. Specifically, the string controller has considerably more information about the actual status of each disk unit than does a SCSI host adapter. In particular, it has rotational position sensing, which enables the string controller to reorder the input queue to take advantage of both radial arm position and rotational platter position. The net effect is to reduce both the typical seek distance and the number of revolutions wasted waiting for the right part of the disk platter to fly under the read/write heads. In addition, the high–speed buffer memory on the ISP–80 is generally larger (1MB) than the combined embedded–controller caches on all of the drives typically connected to a SCSI host adapter[50]. This capacity, combined with a dynamic buffer reconfiguration scheme permits the ISP–80 to provide higher throughput and generally better response time than SCSI, even with the same disk head/platter mechanisms. Note that IPI is the only recourse when dual–porting is necessary (with current electrical and cooling technology, there is space on the disk drive's embedded interface for either SCSI controller logic and memory, or for the second IPI port, but not both). The loading ability of the ISP–80 is shown in Figure 16.

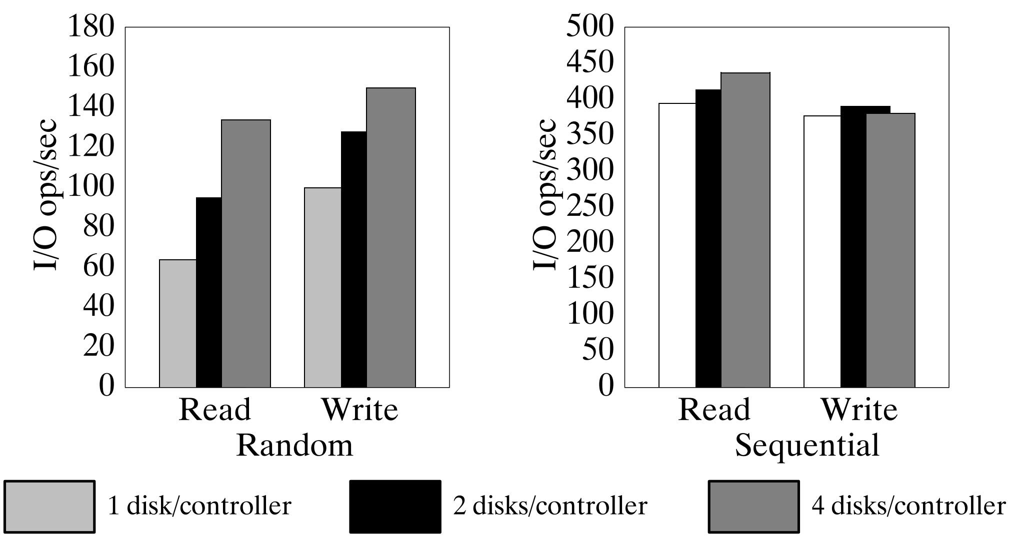


Figure 16. Loading ability of the ISP–80 string controller. The bars represent the number of I/Os available through the string controller when the controller is loaded with the indicated number of drives. For random access activity, the controller scales close to linearly from one to four disks (linearity is actually maintained very close to three drives), while a single drive is sufficient to saturate the subsystem when access is completely sequential. The data underestimates the random access capacity of the subsystem, because only one process is generating the I/O load in all of these cases. These rates were achieved by a SPARCserver 490; the disks are the 1.3GB ZBR drives.

50. Most SCSI disks have 8K or 32K cache; the 1.3GB SCSI has 192K read cache and 48K of write cache. Even with the 1.3GB drive, it takes more than 4 drives to equal the ISP80's cache.

to the host adapter. If you are out of slots for host adapters, it is worthwhile trying to arrange to have only two (or four, under sequential loads) heavily used disks per host adapter.

Exhaustive benchmarking has shown that all of the current (late 1991) crop of synchronous host adapters are essentially the same speed. This includes the built–in host adapters on the SPARCstation–IPX, SPARCstation–2 and SPARCserver 600MP, as well as the Sbus SCSI host adapter and the SCSI/Buffered Ethernet (SBE/S) Sbus card. There are slight differences due to the varying transfer rates available from each board (the maximums range from 4.1 MB/sec to 5.0 MB/sec), but these differences are negligible since the longest individual transfer requested by Solaris is 56KB[49].

SCSI host adapter loading is important when determining what kind of packaging the system should be configured into, especially for the SPARCserver–2. While the SCSI disk farm pedestal offers a convenient and attractive package for many peripherals; unfortunately, it can be hard to configure the right number of host adapters. Sometimes a "junky–looking" system (one that has many boxes scattered around on the desktop) might perform much better if the devices are more appropriately distributed across host adapters.

Surprisingly, one time when it pays to watch transfer rates is when configuring backup devices. In general, if you have a SCSI bus configured to support two fast randomly–accessed disks, it may be wise to consider moving a slow device to another bus if it is likely to be heavily utilized. As noted above, responsiveness from randomly–accessed disks requires very low host adapter utilization. However, a tape drive is nearly always configured to operate in asynchronous transfer mode (all Sun tape drives operate asynchronously). If it is used frequently, its slow transfer speed can occupy the SCSI bus for long periods of time, dramatically raising host adapter utilization and lowering the perceived response time of the disks.

The best (worst) example of this would be a typical entry–level server which has one SCSI bus with several disks and a helical–scan tape drive which is used by the Online:BackupCoPilot to perform online backups. In this configuration, the large amounts of data being transferred to the tape drive can drive the host adapter's utilization well above the 20% critical level and impose significant additional disk response time. Because of the asynchronous nature of the tape devices, sustained backups can nearly consume a host adapter: the 8mm Exabyte transfers at 220KB/sec, which represents 18% loading on the host adapter. The 1/2" tape drive is even worse: its 1MB/sec transfer rate represents 87% utilization! Clearly, any system which expects to perform online backups should configure the backup media on a SCSI host adapter which has few or no disk latency requirements. Systems configuring an IBM3480–compatible tape drive on a SCSI bus may also encounter this issue since those drives are typically very fast (750KB–1.2MB/sec) and have the potential to drive bus utilizations far into the critical zones.

Rule VIII. Avoid configuring "online" backup devices on the same SCSI bus as normally–active disks (except when the disks are half of a mirrored pair, and the normal mode of operation is to detach the mirror for dumping).

49. This maximum is expected to increase at Solaris 2.0.

S_i is the average transfer size for each disk
R_i is the average number of transfers for each disk.

Generally speaking, *this works out to 2 disks per host adapter for random I/O, and 4 disks for sequential I/O*. These critical areas are illustrated in Figure 15.

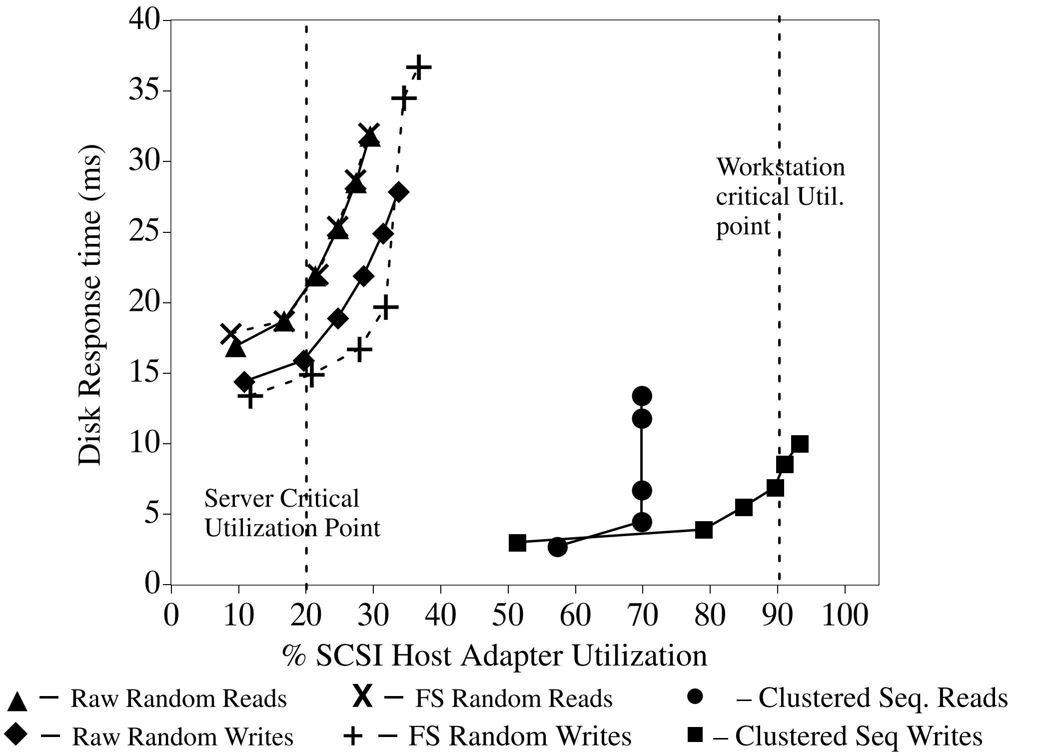


Figure 15. SCSI host adapter utilization and disk response time. Both curves go up sharply fairly quickly. The clustered read case is actually limited by the cpu in this example. The data points correspond to connecting one to six 1.3 GB disks to the SPARCstation–2's on–board host adapter. Random writes are faster than random reads because the embedded controller's cache can consume the writes; the sequential cases also take advantage of the controller cache. The curves for slower drives are even steeper.

Rule VII. Configure two disks per host adapter for random I/O (generally servers), four disks per host adapter for sequential I/O (generally workstations). Exceeding these limits can drastically reduce performance.

Unfortunately, the fact that random writes are less stressful on the system isn't of much use in practice, because very few raw disks are write–only (database log files are the only common example). In fact, the majority of disk accesses are reads. Effectively, this means that for random I/O – typical for nearly any kind of server – you probably want two disks per host adapter. For sequential I/O – typically the case on a datafull workstation – you can reasonably utilize four disks per host adapter. This is the primary reason that Solaris kernels support "only" 4 disks per SCSI host adapter. Of course, if your application is less demanding, or if your bottleneck is elsewhere (such as in the central processor), you can attach more disks

trollers of the SCSI disk drives have buffer memories which are used to pre–read data; the embedded controller for the 1.3 GB SCSI disk also has a write buffer.

All of this presumes that data is being transferred in relatively large blocks. If the access is essentially random and in typical 8KB file system blocks, most of the buffering mechanism is defeated: any data that was read–ahead is likely not used, and even worse, physical I/O must be performed to satisfy the request.

2.4.1. Host Adapters and Controllers

The number of disks attached to a SCSI host adapter or to an IPI–2 string controller can be a limiting factor when the application can request enough disk I/O to keep a number of disks busy. Both the SCSI host adapter and the IPI string controller are responsible for arbitrating access to the peripheral bus, in addition to processing commands being sent from the host, buffering and optimizing bus access, as well as managing its own data transfers to and from the host system's backbone bus. In addition, the peripheral bus itself may saturate. Fortunately, the peripheral bus and its nexus controller generally clog up at about the same time, and we can safely ignore the cables.

2.4.2. SCSI Host Adapter Loading

Studies at Sun suggest that the randomness of I/Os also drastically affects the utilization of SCSI host adapters. Requirements for fast response time for random I/O require very low average utilization on the host adapter and SCSI bus – as low as about 20%. In most server systems, the multi–client nature of service typically causes I/Os to be randomized, even if most clients are accessing data sequentially, since requests from the clients tend arrive at the server in random order. The actual bottleneck is the SCSI bus; when the disk activity is either random or write–intensive in nature the data is unlikely to be in the embedded controller's cache, forcing a physical I/O to be performed. The delay associated with the physical I/O (see the section on disk drives below) means that transferring the data as soon as it is available on the controller becomes paramount to keeping disk latency to a minimum. When the disk activity is primarily serial reads, the disk drivers schedule disk read–aheads; the data is then typically already in the embedded controller's cache. Since this usually eliminates physical I/O time, contention for the SCSI bus isn't as important.

Benchmarking data indicates that optimum response time can be obtained with host adapter utilization of approximately 20% for random I/O and 60%–70% for sequential I/O. Once these critical points are exceeded, response time increases dramatically. A good rule–of–thumb for determining the number of disks that can be optimally attached to a SCSI host adapter is:

$$\text{Disks}_{\text{channel}} = \frac{U \quad T}{\Sigma \ (S_i * R_i)}$$

where: U is the critical utilization
.65 for sequential reads or
.20 for any other I/O),
T is the speed of the SCSI bus,

IPI disks by Solaris[46]. Clustering works by arranging to cache and transfer as much data as reasonable.

Presently, since most devices are limited to a 64KB transfer block (even though SCSI defines the maximum transfer to be 1MB), 56KB is the largest data cluster that is actually moved in a single block. This represents seven 8KB data blocks (the size of a file system block in the BSD 4.2 file system) and enough control information for the host adapter or disk controller to perform the I/O. If the file system is appropriately laid out on the disk[47], all of the data blocks in the cluster can be transferred with a single command setup, eliminating overhead for setting up and taking down the SCSI or IPI data bus. Hopefully, the data is obtainable from (or destined for) adjoining sectors and/or the same data cylinder, which would permit physical I/O to be performed without waiting for multiple head seeks. With all this in mind, the file system and device drivers arrange to maximize data transfers to 56KB whenever they can. Disk clustering in Solaris 1.0 works very well when data is read or written serially.

Clustering works very well when the application permits the file system to optimize on its behalf. Efficiency is noticeably impaired when doing many short random accesses, such as the 2K blocks normally associated with database operations. Using short blocks such as this usually defeats the various optimizations attempted by the file system and disk subsystems. Fortunately, database management systems normally use the raw disk interface, and in so doing they accept responsibility for optimizing access to the physical device. Most of the database management systems recommend the use of raw disk partitions, and have tuned their I/O strategies in many of the same ways as the UNIX file system, as well as using some other (relatively new) tricks, such as asynchronous I/O[48], in which the database manager also assumes responsibility for initiating I/O operations and ensuring their successful completion. This is normally handled by the UNIX file system; asynchronous I/O permits a single–threaded database server process to initiate an I/O and continue to process on behalf of other clients during the I/O's physical motion time. While the overall trend in software has been toward higher and higher levels of abstraction, the ever increasing requirements for performance have pushed very low levels of operating system functionality into relatively high levels of abstraction represented by specialized software such as DBMS and multimedia servers.

While the first optimizations are done in the operating system, the physical disk subsystems are also able to optimize for serial access. The ISP–80 IPI controller and the embedded con-

46. Note that some unbundled software, when combined with SunOS 4.1.1 and earlier, have the effect of turning off disk clustering. In particular, QuickCheck 1.0 and SPARCserver Manager 1.0 have this effect, because they install new copies of the UNIX file system kernel modules which do not have clustering implemented. Some patches may have the same effect.

47. see `tunefs`(8); this is particularly important when using the MetaDisk device driver of the Online:DiskSuite package. Refer to Section 2.11., *MetaDisk*.

48. The terminology can get confusing. In this case, we are referring to asynchronous notification of I/O *completion notification.* "Asynch I/O" in this context means essentially non–blocking I/O: the requesting process need not wait for the I/O to complete before continuing; the process is notified asynchronously when the request is finished. In other contexts (such as when discussing PrestoServe), a synchronous I/O means that the I/O is to be completed before the system call returns. This is *synchronous completion.* It is quite possible to use both for a single I/O request: by opening a file O_SYNC, write/update operations are committed synchronously; if the file is also opened O_NBLOCK, and the write is issued via the `aiowrite`(2) system call, the process is permitted to proceed after issuing a write, thus expecting asynchronous notification of a synchronous write!

8K iops	47	89	86	368	363	358
cpu util	12%	24%	26%	89%	91%	97%
ms/iop	4.21	7.61	5.71	1.52	1.43	1.21
1.3GB IPI, raw disk operations (SPARCserver 490)						
	RRR	RRU		RSR	RSW	
2K iops	74	83		505	85	
cpu util	7%	7%		34%	6%	
ms/iop	0.95	0.84		0.67	0.71	
1.3GB SCSI, file system operations (SPARCserver 2)						
	FRR	FRU	FRW	FSR	FSU	FSW
8K iops	49	61	86	446	262	259
cpu util	11%	13%	16%	91%	48%	52%
ms/iop	5.91	8.31	8.67	2.65	1.88	1.56

Table 7. Performance and processor expense of various disk drives. In this table, the first letter of the column heading is F for File system or R for raw, the second letter is S for sequential and R for random, and the last letter indicates Read, Update or Write. These performance figures were obtained from the IOBench test. The operations cited here are user–level logical operations, and do not account for file system internal overhead (such as updates to the directory entries, allocation of new inodes, *etc.*

Finally, the extent to which other non–UFS processing is involved can also have a significant impact on I/O throughput. Common examples of non–UFS processing are the use of the MetaDisk logical disk driver, the PrestoServe accelerator, and network file access.

Assuming that sufficient disks and controllers are configured, the performance of the disk subsystems as a whole is governed primarily by the randomness of the data access, the number of data channels to memory (*i.e.*, number of host adapters or controllers), and sometimes by the presence of a PrestoServe accelerator. Note that random access I/Os represent substantially greater workloads to the disk subsystem than serial I/Os. Unfortunately, the interleaved nature of disk requests on a server system nearly always means that the disk access pattern seen by the server will be random, even when all of its clients are making serial requests. For carefully–balanced systems, the response time for a serial disk request is nearly three times faster than an otherwise equivalent random disk request (see Figure 17.).

The reason that random reads are more work for the disk subsystem is that they are hard to predict. When the file system or disk subsystem can predict (guess) that a particular block is likely to be used, it is often pre–read from the disk and stored conveniently in memory. Unfortunately, the only time this can be forecast reliably is when performing serial reads. In addition, if sufficient data is being read or written in a cluster, both the file system and the disk subsystems perform read–aheads and write–behinds, up to the largest feasible transfer permitted by the hardware. This is known as disk clustering, and is applied to both SCSI and

the speed at which the system can copy data from cache to the user's buffer[45]. This is illustrated in Figure 14.

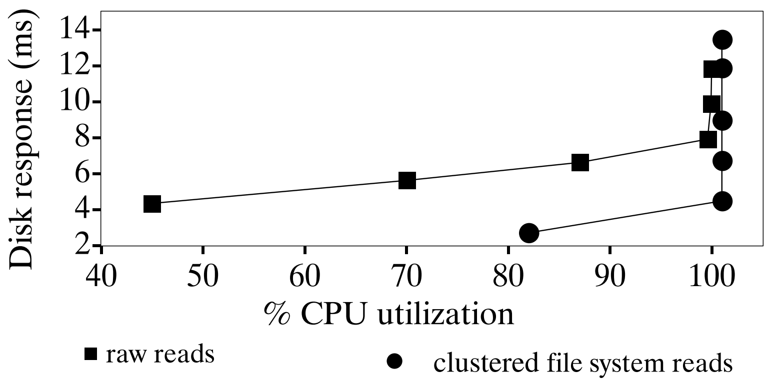


Figure 14. The perceived ability of disks to provide results is dependent upon cpu utilization. When the cpu saturates, response becomes very poor and the disks become nearly idle. The data points are for one disk to six 1.3 GB disks on the SPARCstation–2's built–in SCSI host adapter. Note how quickly the curve for file system reads becomes asymptotic.

Second, the extent to which the operating system must involve itself to locate or present the data typically has a significant impact upon the speed at which disk I/O is serviced. Virtually any server activity involves UNIX file system (UFS) processing, except for database management systems which use the raw disk interfaces. UFS processing is the most critical factor in the performance of a server. Detailed measurements in the UNIX kernel indicate that for file system ("cooked") I/O, processing in the UFS layer can consume 80%–90% of the non–physical time spent processing a disk I/O (the vast majority of the remainder is spent in interrupt processing). Unfortunately, there is little that an end user can do to improve UFS performance in the kernel, other than arranging for minimal fragmentation and seek times (see *BSD 4.2 File Systems*, section 2.10.5.) These facts and a variety of other interesting data are summarized in Table 7. For the 1.3GB IPI drive, the cpu cost of a "cooked" file system operation is 2.5–7 times the cost of a corresponding raw operation, due to the necessity of reading the superblock, directory entries, and following the inode chain to the data. Also, note that the cost of a sequential operation is about a third of the cost of an otherwise equivalent random operation, and the relative difference in performance between random and sequential operations. Finally, this data indicates that SCSI disks impose about 40–120% higher cpu overhead than IPI disks, primarily due to much higher cost for interrupt processing (a file system random read with the 1.3GB IPI consumes the equivalent of 0.00421 seconds * 22 million instructions/sec = 92,600 instructions, while the same operation on the 1.3GB SCSI takes 0.0591 * 28 million = 165,500 instructions).

1.3GB IPI file system operations (SPARCserver 490)						
	FRR	FRU	FRW	FSR	FSU	FSW

45. If the user has control over the I/O logic of his application, the `mmap`(2) system call can be used to avoid a data copy imposed by the semantics of the `read`(2) and `write`(2) system calls. The regular system calls obtain access to the data stored on disk and copy it into the user's buffer. If the access will be read–only or can be written directly into the file from a memory image, `mmap`(2) is considerably more efficient. Basically, mmap can be used to get a virtual memory pointer directly to the stored data on the disk, eliminating a copy operation.

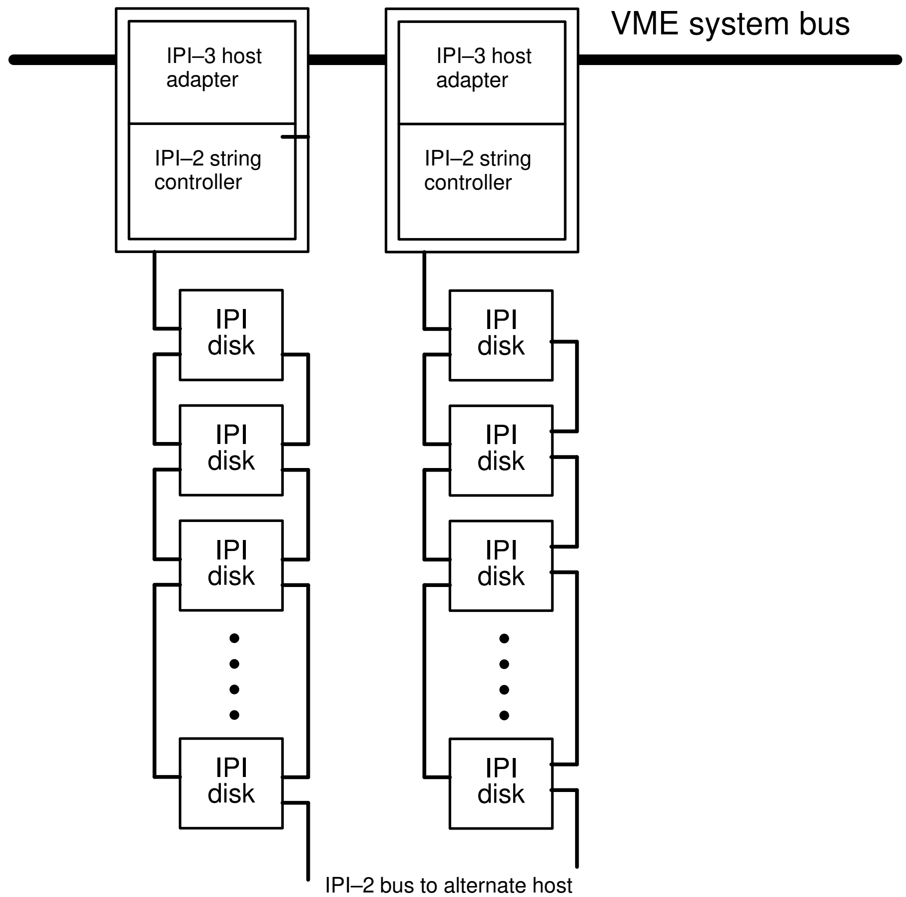


Figure 13. The simplified IPI storage organization implemented by Sun in the 4/390, 4/490 and 4/690.

ly by the speed of the cpu[43]*!* This surprising (and counterintuitive) circumstance is due to several factors. First, one of the fundamental design decisions incorporated into SunOS 4.0 and later takes into account the fact that cpu speeds and especially affordable main memory size are increasing much more rapidly than disk performance, resulting in a tradeoff of relatively plentiful cpu and memory to avoid disk accesses. Under normal circumstances, disk I/O – especially disk reads – are cached in main memory, where access is determined by the speed the main processor can copy[44] the data. When the cacheing mechanism is successful in retaining data in memory, the time required to satisfy a user I/O request is dominated by

42. In fact, most OEM disk vendors now offer their high–performance and/or high–capacity disk modules with a choice of SCSI or IPI interfaces. The actual disk platters, head assemblies, surfaces and recording techniques are identical; only the interface is different. This is the case with Sun's 1.3GB IPI and SCSI disks.

43. Ng, ChakChung: "SCSI Performance Simulation Based on Instrumented Data from SPARCstation–2". SMCC Performance Engineering Internal Report, May 30, 1991.

44. In the SPARCsystem 400 and some Mbus SPARCmodules for the SPARCsystem 600MP, special *bcopy* (*b*lock *copy*) hardware accelerates this process, but even this is fundamentally dependent upon the raw speed of the central processor(s).

In order to provide sufficient information for the string controller to sensibly organize a queue of requests, IPI disks normally provide the rotational position of their platters and the radial position of the read/write head(s) to the string controllers. Each string controller then has enough information to sort a series of disk requests. Because so many conditions must be satisfied for a transfer to occur, effective use this information is vital to high performance.

Sometimes a string controller may be faced with a choice between obtaining data from two drives on its string which are about to become ready to transfer. In these circumstances, an IPI–3 channel may be able to route new requests to an alternate string if the drive in question is connected to multiple strings. A channel can even transfer an existing request from one string to another (even if the request is the next one the queue, it could take as long as 8–25ms to satisfy all of the transfer criteria, even with very fast disk drives). By maintaining state information about each string, and by possibly executing parts of the host system's operating system, the IPI channel can substantially offload the main processors and simultaneously optimize access to the disks.

Sun's present IPI implementation is considerably less extensive than the fully–implemented standard. The IPI–3 facility is not physically implemented; instead, each string controller is treated as being paired to exactly one host adapter, and the facility is emulated (trivially) by a device driver without associated hardware. The IPI–2 strings are directly connected to the main system backbone bus (VME). Because each string is connected to only one host adapter, and because Solaris has no facilities for managing (let alone optimizing) access to a single drive by multiple strings, the second port on present IPI disks is useful only for connection to a second system. The simplified Sun IPI implementation is shown in Figure 13.

IPI subsystems, especially the full–blown implementations, are designed to perform best in situations where massive loads cause fairly long queues to form for each I/O device. If requests form queues which are short or non–existent, the intelligence of the various devices in the IPI hierarchy is largely wasted. IPI excels in heavily loaded servers where I/O requests are the primary work. IPI is generally not appropriate in a workstation, where a single user is unlikely to generate sufficient load for the IPI processors to perform much optimization. In a workstation, the usual case is to have one or two active processes, only one of which generates substantial disk activity. Furthermore, most processes which generate disk activity do so a one at a time, waiting for the current request to complete before issuing another. In such a circumstance, it is difficult to create long queues. When little queue optimization can be effected, the performance of SCSI and IPI drives are very similar[42]. This is why Sun's high–end servers are configured with IPI and all other systems are configured with SCSI.

2.4. Disk Subsystems

Although the disk subsystems are typically thought of as the most performance–limiting factor in servers, *it turns out that it is normally possible to configure sufficient disk and controller resources into the system that the system's capacity to perform disk I/O is limited primari-*

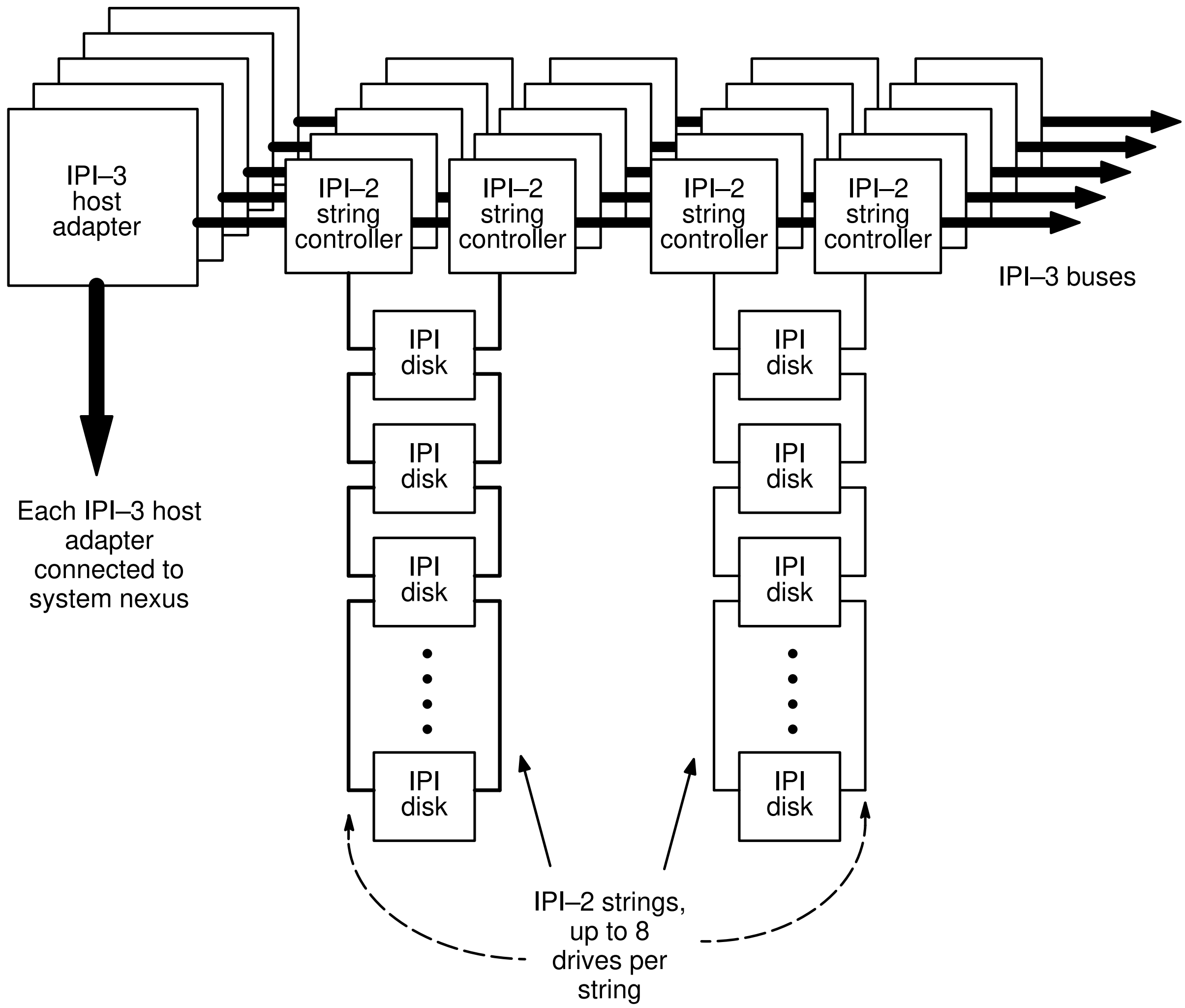


Figure 12. The idealized IPI storage hierarchy. Each IPI–3 host adapter is connected to the host system, typically by the fastest available bus. The IPI–3 bus connects up to 16 string controllers, and each string connects up to eight drives. In this example, each disk shown as dual–ported, although implementations exist with drives from one to four ports. When each disk is dual–ported and maximum IPI–2 string controllers, a system can connect to 256 drives with only four IPI–3 host adapters. Sun's current implementation is considerably simplified. The IPI–3 host adapter's functions are emulated by a combination of the ISP–80 IPI–2 string controller and a high–level IPI–3 host adapter device driver (which has no associated hardware).

The disk drives themselves have no intelligence, nor do they possess local caches. As a result, an IPI disk must have a variety of conditions satisfied in order to perform a data transfer:

- the read/write head must be positioned over the correct cylinder,
- the data must be about to rotate under the head,
- the requesting string must be free, and finally,
- the string controller must have a free buffer.

All of the intelligence in an IPI subsystem is vested in the host adapter and in the string controller. Even relatively primitive operations such as bad–block mapping must be performed by the string controller.

SCSI is intended to provide an inexpensive, flexible mechanism for interconnecting a relatively small number of devices of quite diverse type, IPI is explicitly intended to address the needs of high–performance, high–capacity disk subsystems such as those found in current mainframes and high–end minicomputers. Other devices may be accommodated – but only if they essentially appear to be disk devices, such as “solid–state” semiconductor disks.

As its name indicates, the IPI standard is designed to permit the application of processing power to take advantage of as much optimization as possible, resulting in the highest possible throughput. Toward that end, global optimization information is available to the highest levels of processing. The idealized IPI model is defined in four levels:

Level 0: Mechanical and electrical, which define the cables, connectors, and low–level electrical details;

Level 1: Primitive transfer details, including the bus protocols and the finite state machines;

Level 2: Disk String Protocols, defining device–specific commands, timing, physical device and volume addressing;

Level 3: I/O Channel Protocols, which define logical–level commands, command queuing and stacking rules, buffering mechanisms, timing–independent transfer characteristics, and logical volumes and addressing.

Like SCSI commands, IPI commands defined in the channel protocol are quite high–level; they are intended to provide a mechanism for widely distributing I/O processing in a very complex, extensive computer system. On the other hand, the commands defined in the string protocol are quite low–level, as they are intended to control a completely unintelligent disk device.

In a fully–implemented IPI system, a number of IPI–3 host adapters connect to the main system busses (in a Sun system, they are connected to the VME backplane bus). Unlike the SCSI host adapter, which is essentially a buffered data portal, an IPI–3 host adapter is normally a complete I/O computer (IBM mainframers refer to these as channels; the IPI standard refers to them as either host adapters or facilities), responsible for the management of a number of logical disk strings. Each disk string consists of one to eight disk drives, interfaced to the host adapter by an IPI–2 string controller. The string controllers are connected to the host adapter via a high–speed IPI–3 bus, with a maximum of sixteen strings per bus. Like the IPI host adapter, the string controller is an intelligent device, responsible for optimizing access along its string. Both the IPI–3 bus and the IPI–2 string reflect the massive facilities envisioned for IPI installations: both permit cable runs of up to 50 meters.

Sophisticated IPI implementations permit a disk drive to be connected to multiple strings, via additional ports. Multiported disks may be used either for connection to multiple string controllers in a single system[41], for connection to string controllers in another system, or both. When two ports on each drive are connected to a single system, maximum (16) string controllers are placed on each IPI–3 bus, just four IPI–3 host adapters can connect 256 drives (over 330GB)! The idealized IPI storage hierarchy is shown in Figure 12.

41. Some vendors offer four or even eight ports on various models of disk drive. There are even implementations which bind together several platter/head assemblies into a logical “disk pack”, addressed as a single entity. These options are, of course, quite expensive.

the target *for each word that is transferred!* This imposes a strict practical limit on the maximum transfer speed to and from the host adapter: about 1.2 MB/sec[39]. While this is quite adequate for low–speed devices such as tapes, CD–ROM and WORM devices, it's not very useful for the high performance disks now available. Accordingly, most current host adapters resort to synchronous transfers for high speed devices. The speed at which the host adapter communicates with each target is negotiated when the target is set up and remains constant while the system is in operation. If for some reason a synchronous speed cannot be negotiated with a target (*e.g.,* the target doesn't support synchronous transfers), the host adapter will normally fall back to asynchronous transfers. For Sun devices, the device driver reports the synchronous data transfer speed when it is probed during the initial boot phase. If a device does not report a synchronous transfer speed, it's not operating in synchronous mode. Note that a single host adapter normally negotiates a different speed with each target; while there are differences between adapters, the minor discrepancies in data transfer rate are overwhelmed by all of the other costs of doing physical I/O. It is usually safe to ignore these differences.

Some vendors have begun to implement yet more varieties of SCSI bus. In particular, the SCSI–2 standard includes the notions of *wide SCSI, fast SCSI, and fast–and–wide SCSI.* The SCSI–1 standard defines a peripheral bus with the addressing and transfer characteristics noted above, along with an eight–bit data bus, all carried on a 50–pin shielded cable. SCSI–1 limits transfer speeds to 5.0 MB/sec. Fast SCSI retains the 8–bit nature of the bus and so can utilize the same physical cabling as SCSI–1; it differs by permitting 10.0 MB/sec synchronous transfers. Wide SCSI quadruples the width of the data bus to 32 bits, permitting 20.0 MB/sec synchronous transfers. However, since there simply aren't enough wires in the now–traditional 50–pin cable, the SCSI committee chose to add a second cable (with 66 pins) to the configuration, known as the B–cable. The B–cable carries the additional data lines, as well as some other signals that permit the use of both fast *and* wide SCSI on a single transfer. In fast–and–wide mode, transfers could occur at up to 40.0 MB/sec. At this writing, the authors know of no current implementations of wide or fast–and–wide SCSI peripherals, and very few implementations of fast SCSI[40].

In addition to defining new varieties of interconnection strategies and protocols, the SCSI–2 standard defined an extension to the original SCSI command set. Sun began implementing SCSI–2 commands in SunOS 4.1.1 and the SPARCstation–2.

2.3.4. IPI

In addition to SCSI, the ANSI committees have defined a mass storage organization known as the Intelligent Peripheral Interface (IPI). ANSI regards IPI and SCSI as complimentary standards. Both standards define physical standards for connectors, cabling and the like, as well as electrical protocols and both device–level and subsystem–level command sets. While

39. This is not strictly true. The overhead is dependent upon the physical distance the handshakes must cross. In fact, for very short distances (a couple of inches), asynchronous transfers can be very fast indeed (up to 6 MB/sec); unfortunately, for more realistic distances such as that involved in transferring data from a device located in another enclosure, realistic asynchronous transfer rates are about 1.2MB/sec. In any case, most asynchronous devices don't transfer any faster than 1.2MB/sec.

40. Interphase has announced a VME board implementing a fast SCSI–2 host adapter, with SunOS 4.1.2 drivers. Fujitsu, Seagate and Maxtor are all working on fast SCSI–2 disk targets.

There are a number of different forms of SCSI bus defined now. The proliferation of these forms and terminology can make it difficult to "speak SCSI" in a straightforward way[36]. The electrical mechanism by which signals are defined can differ; in one form, the host adapter transmits a "zero" signal and the "one" is measured relative to that signal. This is known as *single–ended* SCSI. It's not the opposite of double–ended! The alternative signal definition is called *differential*, in which both zero and one are defined by the host adapter. While single–ended is less expensive to implement, it physically cannot carry signals as far. Single–ended implementations limit the physical length of the SCSI bus to six meters, while differential implementations can carry signals up to twenty meters. These distances include all of the signal path inside of each peripheral unit (pbox, lunchbox, pedestal), most of the traces on the embedded SCSI controllers, all the way from the host adapter to the terminator, along with all of the cabling which connects the peripherals[37]. The impedance associated with most connector pairs also accounts for the equivalent of about half a meter. Most people are surprised at the length of the cabling alone inside of most peripheral boxes. For all of these reasons (and a variety of other arcane electrical trivia), six meters often doesn't go as far as one would expect. Exceeding these limitations results in a bus that performs unreliably or sporadically, especially when operated in synchronous mode[38].

Another difference between SCSI implementations shows up in the timing of data transfers. Sun presently offers two basic designs of single–ended SCSI host adapter: synchronous and asynchronous. These terms refer to the mechanism by which the SCSI host adapter and its targets coordinate data transfers. Under the asynchronous discipline, data can be transmitted at an arbitrary time; the sender need not wait for a specific clock cycle to begin its transfer; the sender and receiver need not be in clock synchronization. Synchronous transfers may occur only at specific clock phases. The clock is either transmitted as part of the bus or is agreed upon by the host adapter and target when the target is first initialized. (These generalized definitions for synchronous and asynchronous are applicable to serial communications as well.) In the SCSI world, asynchronous transfers require handshaking between the host adapter and

36. Terminology has run terribly amok in the Sun SCSI world, and nearly all of it is incorrect usage of the terms defined by the SCSI standards! The original 50–pin D–shell ("big SCSI") connector has been called "SCSI–1", although the connector is not defined in either the SCSI–1 or SCSI–2 standards. The newer micro–SCSI connector has also been called "SCSI–2", equally incorrectly. The SCSI 1/2" tape drive uses a *third* kind of connector, which is often referred to as a "Centronics" connector because it looks like a Centronics parallel interface – even though Centronics had nothing to do with the connector, nor does it resemble the parallel interface in either size or electrical characteristics! Finally, some of the older VME SCSI host adapter boards are called "SCSI–2" (this was the original 3/160 SCSI board with internal access only) and "SCSI–3" (this is the SCSI host adapter [known as the 598A or 501–1217] delivered with VME systems from the 3/160 to the 4/490, in both internal–access and external–access configurations). Both of these boards implement single–ended, asynchronous SCSI as defined by the original SCSI–1 standard. *None* of these terms, commonly used in the Sun world, represent correct usage of similar SCSI–defined terms!

37. In addition, each pair of connectors that the signal must cross amounts to another foot or so of cable length, due to impedance and other electrical considerations. The amount of cable consumed by various Sun offerings is listed in Table B4 of the *Sun–4/SPARCsystem Hardware Configuration Guide* (Jan. 1991, p. 95).

38. When the limitation is exceeded, the symptom is usually a stream of messages from the SCSI host adapter driver (esp0, normally). The messages usually refer to an incomplete SCSI command, a data overrun, or a parity error. Sometimes this manifests itself when probing the SCSI chain upon bootup or from the PROM (from the `probe-scsi` verb in a machine with Open Boot PROM). Usually when this happens, disk labels are listed as either corrupted or missing. Note that devices operating in synchronous mode are substantially more sensitive to bus length and impedance.

Sun peripherals nearly always are set up to use unit 0 on whatever target number they are configured to be. The single common exception to this arrangement was the dual–disk shoeboxes (not P–boxes) which were shipped with Sun–3/60's and Sun–4/110's. These boxes contained a single disk controller in the box along with two disk drives; the controller managed the I/O activity of both disks. In these boxes, the controller was assigned a target via jumpering, and the disks were units 0 and 1. All other Sun SCSI configurations have a controller for each device.

The kernel configuration recognizes and names SCSI peripherals by target and unit number. By convention, disks are always targets 0–3 on the bus, tapes are always target 4 or 5, and CD–ROM units are always target 6. This convention is implemented in the kernel configuration file; it can be easily changed if a device has an unusual target address. For example, if a disk device (perhaps an array or a jukebox) connected to the second (first Sbus) SCSI host adapter is hardwired to target 5, it can be addressed as "sd4" by changing the entry

```
sd4 disk at scsibus1 target 3 lun 0
```

to

```
sd4 disk at scsibus1 target 5 lun 0
```

in the kernel configuration file.

Some third–party peripherals use more elaborate addressing schemes. These usually are the sort of device which is logically a single entity with several subunits, such as an optical jukebox with multiple read/write stations. Such a product is normally configured as a single target, with each of its internal components configured as a unit[35]. One way to think about the entire addressing scheme is to think of targets as street addresses. In such a scheme, each building (target) has an address on the street (the SCSI bus); Sun peripherals are like single–family homes, while more complex devices are more like small apartment buildings in which each dwelling has both a (common) street address and its own apartment number.

As noted above, Sun systems normally configure the host adapter of each bus as target 7 unit 0, but this need not always be the case. If some third–party peripheral insists upon being configured as target 7, it is possible to reconfigure the host adapter to be another target (this takes kernel modifications). One particularly interesting variant of this scheme is to configure a single SCSI bus with *two* host adapters, each on a different system. This arrangement, known as a double–ended SCSI bus, might be used to provide fairly high–speed system–to–system communications over short distances. Although SCSI host adapters are cheaper than FDDI, this mechanism has some disadvantages: in particular, there is little isolation of system faults, so if one system panics or has some sort of abnormal electrical circumstance on the bus, the result on the other end is hard to define. If a customer is interested in this configuration, Sun Consulting has implemented the software at least once.

35. The SCSA implementation is much more administrator–friendly. The SCSI kernel configuration now is expressed in SCSI terms (targets and units), rather than in bus units and addresses. Compare the SCSI sections of a Sun–4 kernel with a Sun–4c or Sun–4m kernel!

data will transfer the data to the host adapter, which is then responsible for making arrangements to transfer the data across the backbone bus to the final destination[34].

The commands used by host systems to request specific actions of devices are defined in the SCSI standard. At present, commands can be divided into two groups, known as Group 0 and Group 1. The primary difference between G0 and G1 commands is the maximum addressability in the device (G0 commands are limited to about 1GB, G1 commands can address considerably more). The commands defined in the standard are quite diverse and surprisingly high–level (*e.g.,* “find key” and “format”). It is worth noting that no major system or peripheral vendor uses much more than the most basic subset of commands. Vendors can do much more to distribute I/O processing from the host cpu to the peripherals themselves. One of the factors precluding more extensive implementation of SCSI commands is the consequent necessity for even more powerful embedded SCSI controllers: present embedded disk controllers usually consist of fairly dense circuitry which is already approximately the size of the host disk drive itself. The addition of a more powerful processor would necessitate either larger form factors or the – expensive – fabrication of ASICs.

Each device resides on the SCSI bus has a unique address consisting of two parts: a *target id* and a *unit number.* The bus is actually arranged in a hierarchical fashion, as shown in Figure 11. The host adapter itself must also have an address. On Sun systems, the host adapter

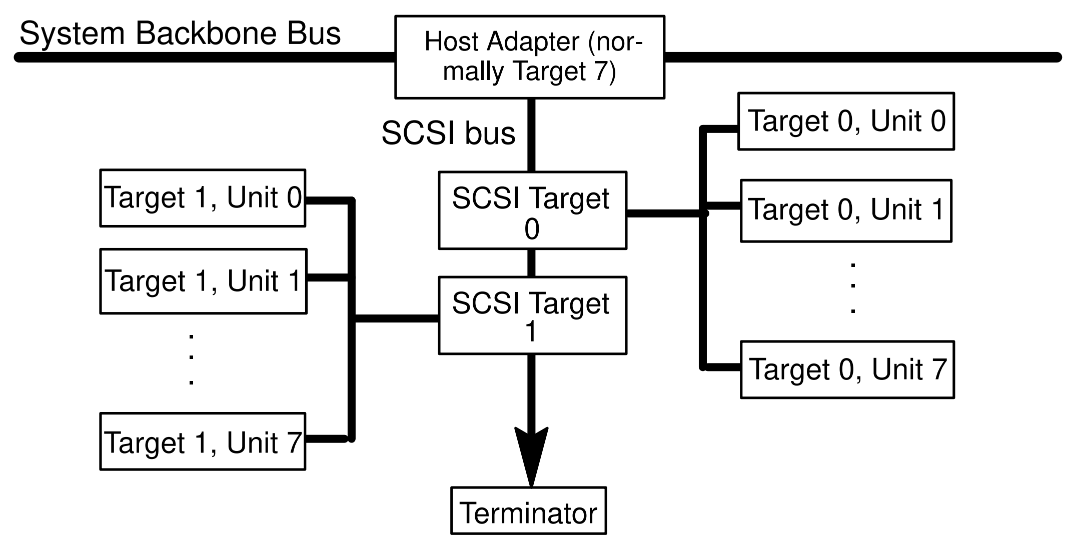


Figure 11. The idealized hierarchical architecture of a SCSI peripheral bus. The host adapter is nearly always configured as target 7 in Sun systems.

is nearly always target 7, unit 0. Every device on the bus has both a target number and a unit number. Both target numbers and unit numbers are 3–bit quantities. Since it is impractical to configure other devices with the host adapter, the maximum theoretical configuration of a SCSI bus is 56 devices (seven independent targets each with eight units).

34. The recent re–implementation of the SCSI device driver system, known as Sun Common SCSI Architecture (SCSA) is an attempt – mostly successful – to reorganize the device driver code to more closely follow this idealized organization. In particular, SCSA aims to make it possible to implement device drivers for various targets without writing code that is specific to a particular host adapter.

The Sbus clock speed is generally some integer multiplier or divisor of the system or cpu clock. Otherwise, complex synchronization would be necessary to transfer data between busses of differing clock phase. Generally speaking systems with clock speeds of 20– or 40–Mhz operate Sbus at 20Mhz, while 25– and 50Mhz systems use a 25Mhz Sbus.

2.3.3. SCSI

The Small Computer System Interface (SCSI, usually pronounced “scuzzy”) peripheral bus is probably the least–understood architecture in the Sun world, although every Sun system has had SCSI since at least 1985. The term SCSI refers to a particular set of standards defined by ANSI which defines a mechanism for implementing a “data highway” with well–known characteristics between a backbone memory bus and a wide variety of peripherals. Because it was originally intended to address the needs of small, inexpensive systems, SCSI is oriented toward achieving excellent results at low cost. Accordingly, it was designed to be simple, and especially to provide flexibility to configure peripherals without having to radically change arrangements in the host computer. SCSI defines a peripheral bus for all of its member devices to connect to. It is completely separate from the system’s backbone bus, except for an intersection point where the two join. This point is called the SCSI Host Adapter. It is best thought of as a portal through which data passes from one bus to the other and back. Most current SCSI host adapters additionally provide buffers in order to enable transfers to be made without requiring control of both the system backbone bus and the SCSI bus.

In order to achieve the desired high level of device independence for the host operating system, SCSI devices present a very simple architecture. For example, the geometry of a disk drive is presented as a linear sequence of uniform disk blocks, when in actuality a disk has a complex multi–dimensional geometry consisting of surfaces, cylinders, tracks, densities, bad block maps, and a multitude of other details. In such cases, the device itself or an attendant controller/interface is responsible for providing translations services from the simplified SCSI notions to the more complex reality. This permits the host system to communicate with a wide variety of peripherals in a very high level, idealized fashion. One consequence of this organization is that intimate knowledge about a device is available only to entities at the remote (peripheral) end of the SCSI bus. This information is often used for optimization; the practical result is that most optimizations must be effected in a device’s (normally embedded) controller. On the other hand, high degree of device abstraction successfully permits a wide variety of devices to be connected via SCSI, ranging from high–performance disks and tapes to floppies, hand–held image scanners, printers, and even projection displays.

System components which reside on the backbone bus (VME or Sbus in Sun systems) transfer data to peripherals residing on the SCSI bus by sending the data to the host adapter, along with a command and a destination address. The host adapter is then responsible for completing the transfer to the device and notifying the host system of the completion status of the actual device’s transfer. In the other direction, a peripheral responding to a host request for

GXplus	no	16MB		
ECL (hires) monochrome	no	1MB		
analog (low–res) mono	no	1MB		
GS	no			
GT	yes			
VX, MVX	yes		8 MB/sec	4–8 MB/sec
FDDI/S			12.5 MB/sec	4 MB/sec
HSI/S			0.768 MB/sec	192 KB/sec
Sbus Ethernet	no		1.2 MB/sec	0.9 MB/sec
TRI/S (4– or 16Mb)	no		2 MB/sec	600 KB/sec
SCSI/Buffered Ethernet	no		6.25 MB/sec	3 MB/sec
Sbus SCSI	no		4.8 MB/sec	2.5 MB/sec
Sbus Serial/Parallel	no		250 KB/sec	10 KB/sec
Sbus PrestoServe	yes		5 MB/sec	5 MB/sec
SPARCprinter interface	yes		4 MB/sec	2 MB/sec
3rd–party Sbus expanders	yes (but most don't *provide* masters in the expansion)	40K	N/A	N/A

Table 6. Characteristics of various Sbus expansion offerings.

There are presently three different flavors of Sbus: Rev A.0 (maximum clock speed 20Mhz, maximum physical transfer 32 bits), Rev A.1 (25Mhz, 32 bits), and Rev B.0 (25Mhz, 64 bits). All current Sun systems implement Rev A.1 Sbus at 20– or 25Mhz for the basic interconnection. However, the complete implementation of the Sbus varies from system to system. For example, the SPARCserver 600 systems have write buffers configured on both sides of the Sbus, which capture transfers and enable the transmitter to complete a transaction without having to acquire control of the bus. The write buffers accelerate write throughput and reduce bus residency. The effect of the buffers is dramatic: the 20Mhz Sbus in a SPARCstation–1+ sustains about 20MB/sec; the buffered Sbus on the SPARCserver 600 operates at the same 20Mhz clock but sustains over 32MB/sec. Peak rates are even higher; the 20Mhz/32–bit implementation transfers four bytes every clock, resulting in a maximum of 80MB/sec. The Rev B.0 version transfers 64 bits per clock; at 25Mhz peak transfer speed is 200MB/sec, and a fully buffered implementation should be able to sustain nearly 160MB/sec. This is particularly important for systems involved in multimedia and other data–intensive applications.

Addressing is also different from VME. Instead of each board claiming address space from a common pool, Sbus allocates 32MB of address space per slot. This is known as geographic addressing. The geographic addressing scheme drastically simplifies the design of the Sbus board itself, because it may confine its attention to a single address space which by definition is private to itself. This imposes some restrictions on Sbus expansion. Like a VME expander, an Sbus expander locates a “jumper” board in one of the available Sbus slots. This jumper board provides a cable to an expansion chassis with more (usually four more) slots. The configuration catch is that the entire expansion chassis is in effect “wedged into” the single slot occupied by the jumper board, *all of the boards in the expansion box must be able to fit within the total address space of the jumper slot*. The address space integrity of each board in the expansion chassis is maintained, because the expansion chassis itself divides its address space geographically. However, Sbus boards which expect to utilize a great deal of address space may not be able to coexist within the limited addressing of an expansion box. For example, the GX graphics accelerators require 16MB of address space. Since all of the jumper boards require some address space of their own (typically 40K or so), it isn't possible to configure two GXs into an expansion box (since the addressing requirement would be 16MB * 2 + 40K, exceeding the 32MB available). On the other hand, one GX and two ECL monochrome framebuffers *could* be configured into an expansion box, since the ECL framebuffers address less than 1MB each. The performance implications of this method of Sbus expansion have yet to be explored.

Rule VI. Calculate required Sbus board address space requirements on a per–slot basis before configuring an Sbus expansion box.

Some third–party Sbus expansion boxes permit *sparse addressing*, in which the 32MB address space is divided up on demand. This permits the use of boards which actually map relatively large address spaces. However, if the board actually uses the entire mapped space, this approach doesn't solve the addressing problem. The GX, for example, apparently does use most of its 16MB address space, but not *all* of it[33].

Like VME, Sbus boards can be masters and slaves. Unlike VME, where the cpu either supports foreign master operation on every slot or doesn't, Sbus master/slave operation can be implemented on some slots and not others. The original SPARCstation–1 and 1+ offered master/slave operation on Sbus slots 1 and 2, but offered only slave operation on slot 3. All other Sbus slots on all other Sun platforms permit both master and slave operation. However, many of the third–party Sbus expansion boxes provide *only* slave operation in the expansion slots, limiting the usefulness of these options. Fortunately, it seems that a much smaller proportion of Sbus boards require master/slave operation than VME boards, where probably 80% of all boards require master/slave. Sbus configuration characteristics are listed in Table 6.

Sbus Board	Requires Sbus master?	Required Address Space	Maximum Bandwidth Demand	Typical Bandwidth Demand
GX (1– or 2–slot versions)	no	16MB		

33. The Texas Microsystems expansion box implements sparse addressing, and users have reported that two GX boards and a 1055W SCSI board work in such an expansion.

whenever possible, although some VME peripherals have no DMA hardware and must operate in non–stream mode[32].

Because private P2 bus operations are not really VME transactions, the IOC has no effect on transfers across these busses. These buses are usually synchronous and always oriented toward block transfers, and an IOC would not appreciably accelerate these subsystems.

VME board	Maximum Bandwidth Demand	Typical Bandwidth Demand
ISP–80 IPI–2 String Controller	6 MB/sec	6 MB/sec
FDDI/DX Dual–Attach FDDI	12.5 MB/sec	4 MB/sec
Asynchronous SCSI host adapter (598A)	1.2 MB/sec	1.0 MB/sec
ALM–2 Asynchronous Line Multiplexor	350 KB/sec	10 KB/sec
Network CoProcessor	0.9 MB/sec	0.9 MB/sec
ie2 Ethernet	1.0 MB/sec	0.5 MB/sec
MCP	750 KB/sec	250 KB/sec
IBM Channel Attach	3 MB/sec	3 MB/sec
VX image processing accelerator	8 MB/sec	4–8 MB/sec
MVX image processing accelerator	0 MB/sec (uses private P2 bus)	

Table 5. Maximum and typical bandwidth consumption of various Sun VME boards. Typical demand estimates the bandwidth consumed in typical heavy load applications. In most cases, typical demand is *far* from maximum consumption.

2.3.2. Sbus

All current Sun systems have an Sbus, which is anticipated to be the expansion bus of choice on Sun systems for the foreseeable future — certainly through 1995, at least. Even systems which do not have Sbus expansion slots, such as the SLC and ELC actually have onboard Sbus circuitry connecting the built–in peripherals with the processor complex. Physical implementations of Sbus are called mezzanine busses because the boards typically are configured in such a way as to physically overhang part of the main processor complex, much like the mezzanine level in a stadium. Like VME, Sbus is a circuit–switched bus.

Sbus was designed specifically to overcome many of the shortcomings of VME. Transfers may be performed in 8–, 16–, 32–, 64– or 128–bit blocks, using duplexed data lines to pack the bus into a small (about 3”x5”) form factor. All transfers are done internally as block mode transfers (in effect, the Sbus has its own I/O cache designed in). The differences in data transfer size are defined to be transparent to Sbus boards, except for timing considerations. Thus a 64–bit board may be used with full functionality in a 32–bit bus (albeit at the slower speed of the 32–bit implementation).

32. In the 4/400 systems, the IOC was shared with the on–board Ethernet interface. This is not necessary on the 4/600 systems because the ethernet is provided with its own buffers.

(*i.e.,* the signal routed onto the board) for boards which are *bus masters*[27],[28]. Bus masters are boards which can assume control of the bus and initiate transfers to and from the board and main memory. Boards which are not masters are referred to as slaves. Slave transfers must be initiated and managed by a third party, generally the cpu. Generally speaking, high–performance DMA boards which transfer a great deal of data such as disk controllers and advanced frame buffers are bus masters. Other, less demanding boards, such as TAAC–1 visualization accelerator are VME slaves[29]. (All VME masters also permit VME slave operation.) All Sun VME–based systems except the Sun–4/100 support both VME masters and VME slaves (the 4/100 did not permit other VME masters).

2.3.1.1. I/O Cache interface to VME

As defined by Motorola, the VME devices may interact with memory in quantities of two to 32 bytes (larger transfers are broken down into 32–byte transactions). Each transfer requires setup time as well as transfer time. This consumes time, and more importantly it drives up bus residency. Obviously, the larger the transfer, the more efficiently the bus is utilized, since fewer setups are done. Older Sun servers (through the 4/300 series) permitted arbitrary sized–transfers to take place at the discretion of the device and its driver. However, in the 4/400 and 4/600 processors, Sun has implemented an I/O cache (IOC) between the VME bus and the memory bus. The cache collects VME transfers and bursts all transfers to and from memory in 32–byte blocks, greatly increasing the efficiency of the bus and bus/memory interface. Suitable arrangements are made for transfers which are not full increments of 32 bytes. The IOC is essentially a write–back cache with a 32–byte line size, placed between the VME and memory busses.

The use of the IOC enables SPARCservers to sustain DMA transfer rates (18–25 MB/sec) very close the the practical limit of 27.9MB/sec[30]. These transfer rates apply *only* to so–called stream I/O transfers performed under DMA or DVMA. Stream I/O supports one transfer mode: sequential, unidirectional, block–sized and block–aligned to and from memory. Non–stream transfers (such as those typically performed by programmed I/O) defeat the IOC mechanism by spreading transfers out across multiple IOC bursts: the transmitter appears to be a very slow device due to the large amount of overhead involved. In addition, the hardware ensures that non–stream operations are cycle–by–cycle coherent with the system caches, due to the fine–grained nature of the transfer[31]. This too is considerably less efficient than stream operation. For these reasons, programmed I/O transfers operate at about 4MB/sec on 4/300 systems and at about 5 MB/sec on 4/400 systems (in addition to consuming considerably more attention from the main processors). Non–stream I/O is to be avoided

27. Refer to the *Sun Field Engineer's Handbook*, *Slot Assignments*, for information on jumper settings for specific boards.

28. Misconfiguration of the jumpers, for example, when the jumpers are pulled in an empty slot, causes boards "downstream" (in higher numbered slots) to fail to respond. Sometimes, misconfigured jumpers can cause apparently random VME interrupts to "appear" in the system.

29. The TAAC–1 rarely transfers data to the host memory, and in general makes relatively slight demands for data transfer to and from VME.

30. Patterson and Hennesey, p. 532.

31. Narad, Chuck. *Sun–4M System Architecture (revision 2.0)*. SMCC Internal document, July 1991.

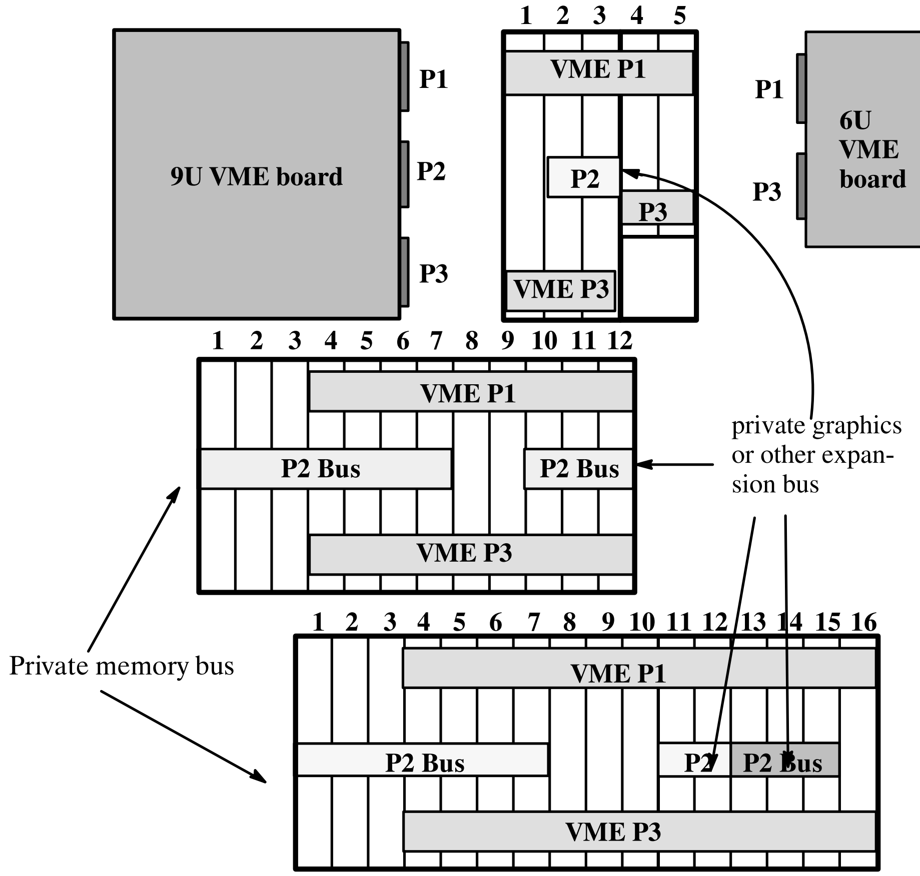


Figure 9. VME backplane configurations. The P2 backplane connections provide the ability to implement various kinds of private busses for board–to–board communications. Sun uses the P2 backplanes in slots 1–7 of 12– and 16–slot backplanes to implement high–speed cache–to–memory (3/200 to 4/400 systems) or Mbus–to–memory (4/600MP systems) busses. Historically, the P2 busses in slots 2–3 of the 5–slot, 10–12 of the 12–slot, and 11–12 and 13–15 of the 16–slot backplanes have been used for implementing graphics busses.

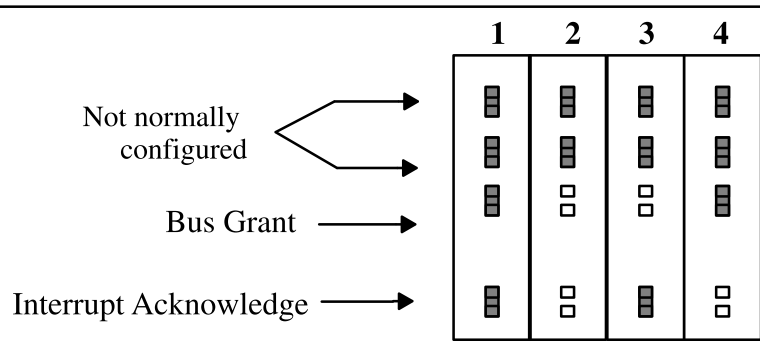


Figure 10. Typically configured VME front–plane jumpers. Normally, both bus grant and iack are pulled (as in slot 2 shown here) for bus masters, and both are left in, as shown in slot 1.

The VME card cage itself provides both physical mounting and signal connections. Timing restrictions built into the VME standard limit the number of slots in a cardcage to 21. Sun offers expansion up to sixteen slots; if more slots are required, third parties offer a multitude of expansion card cages. In these schemes, the last slot in the primary card cage is used by an expansion board which provides a “jumper” cable to a card cage with additional slots. Each slot has access to the entire VME address space[26]. Boards residing on the backplane are required to choose an address space, treating that space as exclusive property. Usually, this address space is selected by jumpers or DIP switches located on the board. Device drivers must be informed of the addressing mode, as well as the physical address space of each board, usually in the kernel configuration file.

The VME standard provides definitions for three different backplane connections, one of which is optional. These connections are referred to as P1, P2, and P3; the P2 connector is optional. P1 and P3 carry the standard VME signals, including power and ground. The P2 connector is specified by the standard as *implementation defined.* Sun often uses the P2 area for Sun–specific, high–performance data transfer across the backplane. Most Sun processors with VME use the P2 to connect the main cpu board to offboard memory (the 3/200, 4/200, 4/300, 4/400 and 4/600 models all use this backplane connection for – different – private memory bus implementations). Sun uses the implementation–defined nature to provide high–speed synchronous private memory busses: transfers across the P2 run at 66MB/sec on the 3/200, and up to 270MB/sec on the 4/600. Sun has also used a private memory bus in some of its VME–based accelerated graphic accelerators, such as the CXP and GXP. These multi–board framebuffers used a *different* bus protocol to communicate between boards. To avoid conflicts between the main memory bus and the graphics bus, the P2 part of the backplane is divided into two or three discrete areas (depending upon the number of slots in the backplane). The VME backplane configurations are shown in Figure 9.

In addition to addressing on each board, there are two board–level characteristics which are configured on the VME frontplane (that part of the VME backplane physically on the other side of the actual board plug receptacles; these are usually accessible from the *front* of the system). These are the *bus grant (BG)* and *interrupt acknowledge (IACK)* jumpers as in Figure 10. Presence of the jumper permits the circuit to pass through the slot without requiring the board to take any action. Consequently, slots that are not in use should be jumpered. When the jumper is removed, no circuit is present, and the signal is forced onto the board. This is done to permit the board to intercept the signal and act accordingly. The signals in question (BG and IACK) are both generated from the VME controller, which on Sun systems is always located on the cpu complex. Normally, both the IACK and BG jumpers are pulled

26. There are actually a variety of different kinds of (overlapping) VME addresses: a24d32 (24–bit address, 32–bit data), a32d32 (32/32), a16d32, a24d16, etc. Virtually all Sun boards use the a32d32 space.

2.3. I/O and Peripheral Busses

Sun offers two basic I/O busses: Sbus and VME. The processor/memory complexes in all Sun systems are connected to the external world via one of these two bus structures. Each I/O bus has one or more peripheral busses which effectively are branches of the parent I/O bus. Sun offers two basic designs of peripheral busses, namely SCSI and IPI. High–performance peripherals which conduct bulk transfers generally are configured on one of these peripheral busses, in order to benefit from intelligent processing at the host adapter level, and also to avoid contention for the higher–level system bus. Any of these busses can be a bottleneck, and some attention must be paid to each in configuring a system.

> **Rule V. Estimate total demanded throughput across each configured bus (SCSI, IPI, VME, Sbus), and compare to maximum ability. Typical bandwidth consumption is listed in Tables 5. and 6.**

2.3.1. VME

By 1991, the standard 32–bit VME bus is becoming obsolescent. Its definition limits it to about 40MB/sec peak throughput, and most actual implementations are limited to 25MB/sec or less in practice[24]. Because of this performance limitation, expensive backplane design (when compared to smaller mezzanine busses), relatively large physical size and power dissipation, VME has lost favor throughout the industry. However, two factors argue for the VME expansion: the tremendous number of existing peripherals and the design freedom made possible by the much larger form factor (especially the so–called 9U form factor used in Sun's VME servers). For the moment, it is not feasible to create designs such the VX and MVX imaging accelerators or the ISP–80 IPI string controller within the size and power constraints of Sbus and other mezzanine busses.

Motorola and the VME steering committee have defined several versions; the VME Rev C.1 definition has been officially accepted by the IEEE as IEEE–1014. Sun systems from the 3/160 to the 4/260 implement VME rev B.2. The 4/300, 4/400 and 4/600 cpus implement VME rev C.1. There is some current talk in the industry about implementing 64–bit versions of VME called VME64 and VME64+, but these are neither standard nor popular at this time. All versions of VME are circuit–switched busses.

There are three different common VME form factors, known as 3U, 6U, and 9U. Most VME boards offered by Sun are in the 9U form factor[25], which is approximately 14"x17". This form factor was chosen over the 6"x9" 6U form factor because of the much greater integration possible on the larger board (the 9U/6U decision was made in 1985 with the introduction of the Sun–2/160). The 3U form factor is even smaller, about 3"x6", and is not especially common. Most VME boards not built specifically for the Sun world are 6U. Sun and many third parties offer 9U–to–6U adapters which permit the use of a 6U board in the standard Sun 9U card cage.

24. Note this is a *defined* limitation, and no standard–compliant implementation can exceed this. Some vendors claim as much as 55MB/sec throughput on VME, but these vendors are either not complying with the standard, or they are transferring at this speed over the implementation–defined, *proprietary* P2.

25. Only Sun's Sun–3E and SPARCengine 1E processors, and their attendant peripheral boards are 6U form factor. Note that other SPARCengines, such as the SPARCengine 300 are 9U implementations.

cache[23], since data that is faulted into memory stays there subject to the normal (highly optimized) page–replacement algorithms used to manage virtual memory. Most systems derived from System–V Release 3 or earlier do not use this technique, using instead a reserved area of memory (normally 10% of physical memory) known as the *buffer cache* (see Buffer Cache, Section 2.10.3.), and are consequently often much slower in many situations where the large memory sizes common today result in much high cache hit rates on Sun–like virtual memory systems.

The single–level virtual memory greatly accelerates disk “I/O” when a server is called upon to repeatedly supply data that can be cached, for example in situations when a server is supplying shared binaries to a wide community of workstations (Open Windows, for example, which would be in use on a large number of workstations simultaneously, and would be frequently paged out of the client workstation, resulting in page retrieval traffic on the server). Generally speaking, this is most effective in random–access situations, as serial transfers are often often longer than even a large physical memory.

> **Rule III. Configure NFS servers with large memory (say, 2MB per client), especially when providing a relatively restricted set of data to a large number of clients.**

Note that it is sometimes possible to configure too much memory and end up with a slower system! In particular, off–board memory in the SPARCsystem600MP is about 15% slower than memory physically located on the main cpu complex board. The reason for this is the distance of the memory from the Mbus. Offboard memory is connected to the Mbus by traces which are several feet long, while the onboard memory is only a few inches from the Mbus. If the application does not require more than 128MB (512MB with 16Mb SIMMs), don’t configure any more in. The slower off–board memory is configured into the top part of the physical address space, and contrary to intuition, it gets used just as quickly as the on–board memory. This is due to the way that UNIX allocates the kernel’s memory from the top of memory, and grows it downward; user processes are allocated at the bottom of memory and grown upward. Fortunately, the slower speed of the memory bus does have an upside: the expansion memory boards can accept memory with cycle times as long as 100ns, while the onboard memory banks must be configured with 80ns parts.

> **Rule IV. When configuring SPARCsystem 600MP, try to fit all configured memory onto the cpu complex, rather than using expansion boards.**

Both the SPARCstation–2 and SPARCsystem 600MP use multi–way interleaved memory to accelerate memory access. The SPARCstation–2 uses a four–way interleave. When an 8–bit byte is accessed from memory, two bits are obtained from each of the four banks. Likewise, the SPARCsystem 600MP uses a 16–way interleave scheme. The SPARCsystem 600MP’s memory bus is 128–bits wide; each memory access is divided across all 16 banks, permitting faster access due to fewer memory recovery cycles. This is why SIMMs must be installed four or sixteen at a time in the respective systems.

23. This technique, pioneered by the original Honeywell Multics system from which UNIX was derived, is also known as “file mapping”.

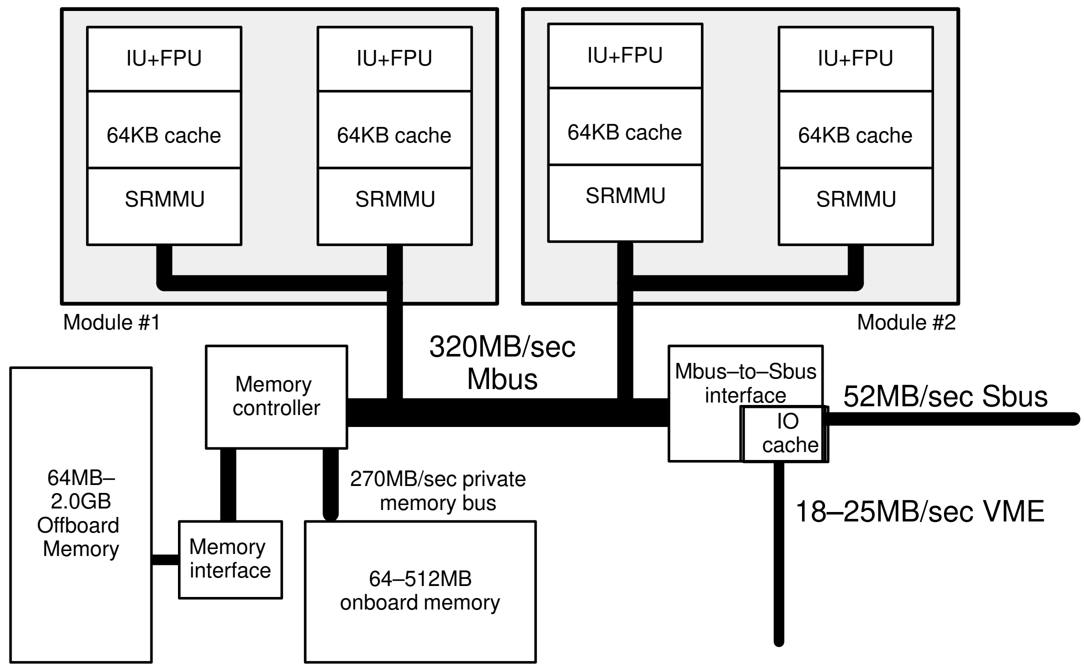


Figure 8. The organization of the SPARCserver 600MP. The Mbus is the nexus of all activity. Both main memory and the I/O busses are connected directly to the Mbus, as are the processors.

quired to keep the caches coherent increases more quickly, roughly in proportion to the number of cache misses. But as the number of processors and processes increases, the number of cache misses increases, and the Mbus can become saturated. In these cases (which are fortunately very rare – they typically occur with maximum processors when executing processes which individually are cache busters), the system *may* run faster with fewer processors. In such a circumstance, the caches get overwhelmed by the large address spaces being staged from memory, and cache coherency Larger caches help, since they involve fewer cache misses, and hence less Mbus traffic (less traffic with memory since it's already in the cache as well as less traffic for keeping new cache blocks up to date). Expect future Mbus modules to have (much) larger caches, drastically reducing memory bus and especially Mbus traffic.

2.2.5. Main Memory

There are two main purposes for memory in most servers. Most obviously, memory is used to accommodate the data storage needs of application programs, such as a DBMS back–end. Memory is also frequently used as a cache for accelerating disk access. The entire memory and disk address space is organized as a single–level virtual memory. This means that any I/O is treated as a page fault: the process merely refers to the desired data by its virtual address. If the data is in physical memory, nothing special happens; if the data not in memory, a page fault is generated, and the virtual memory subsystem brings the data in, restarting the faulted reference. The effect of this is to utilize virtually all of main memory as a disk I/O

System	MMU type	Contexts	PMEGs	Page size	mappable memory
Sun–4/280	SRAM	8	512	8K	128M
SPARCstation–1, 1+	SRAM	8	128	4K	32M
SPARCserver 390	SRAM	8	256	8K	64M
SPARCserver 490	SRAM	64	1024	8K	256M
SPARCstation–2	SRAM	16	256	4K	64M
Ross 6002 module	TLB	4096	N/A	4K	all

Table 4. Current SPARCsystems and their MMU characteristics. Note that the term "context" means different things under the two different MMU organizations. Mappable memory refers to the amount of memory that can be mapped without exchanging PMEG entries. The SRMMU in the Ross 6002 can map all of physical memory, although it must reload TLB entries to do so; the difference is that the TLB can be reloaded without an expensive kernel trap.

2.2.4. Mbus and other Memory Busses

Main memory is connected to the cpu complex by a memory bus. Each platform uses a different design optimized to the specific needs of the system. On all systems except the SPARCserver 600MP, the memory bus is matched to the capabilities of the cache and cpu. In these systems there is always sufficient memory bus capacity regardless of the task at hand. The memory busses in VME systems are routed across the implementation–defined P2 section of the VME bus (for further details see section 2.3.1.), using proprietary protocols. The capacity of these busses ranges from 66MB/sec in the SPARCstation–2 to 120MB/sec in the SPARCserver 490[22].

In a uniprocessor system, it is easy for the system architect to design the memory bus with sufficient capacity that it is virtually never a bottleneck. However, in a multiprocessor, the varying demands of many processors makes it economically impossible to design a bus fast enough to accommodate maximum throughput to all processors simultaneously. Accordingly, multiprocessor memory busses must take into account the speed of the cpus, the size and organization of the caches, the cache coherency protocol, and the speed and organization of the memory subsystem. To add complications, the SPARCserver 600MP has replaceable processors, meaning that the memory bus must be able handle the demands made by different generations of processor modules.

In the SPARCserver 600MP, there are *two* busses which constitute the cpu–to–memory connection, the Module Bus (Mbus) and a private memory bus. The processors reside directly on the Mbus, which serves as the system nexus, as illustrated in Figure 8. The Mbus carries user memory traffic as well as the cache coherency protocol messages, which keep the four independet caches consistent. As more processors are added onto the Mbus, the traffic required to supply data to the processor modules increases roughly linearly. But the traffic re-

22. The SPARCserver 490's bus is asymmetric, due to the design of its high capacity I/O subsystem. Consequently, it transfers at 90MB/sec from memory, but 150MB/sec to memory. To reduce confusion, Sun's literature usually cites this bus as being 120MB/sec, the average of the two rated speeds.

address mappings[21]. Under such an arrangement, the number of contexts available in the MMU directly affects the number of active processes the system can efficiently manage. Each context contains a table mapping the virtual–to–physical mappings for *page map entry groups.* These are the infamous PMEGs. A PMEG contains the mapping for a group of pages, necessarily belonging (mapped) to a single a process.

This organization has one great strength and two relatively minor weaknesses. The advantage is that because there is such a relatively large amount of memory devoted to address translation, the processor complex operates very efficiently most of the time – usually, the MMU has only to retrieve data from fast SRAMs to perform the requested address mapping. However, since each PMEG has a limitation on the number of mappings it can retain (256KB per PMEG), attempts by a single process to map more than the sum of the PMEGs' maps causes a trap into the kernel for the purpose of re–arranging the PMEGs to accommodate the latest mapped segments. The normal address translation process is handled in hardware using the PMEG SRAMs, and this represents a drastic reduction in efficiency. Likewise, attempts to run more *active* processes than the number of physically provided contexts in the MMU result in a similar kernel trap to swap contexts. This too represents a drastic reduction in efficiency. This design works *very* well in a workstation or a small server, in which relatively few (say, 2–8) processes are frequently active, and in which the address space requested on a per–process basis is fairly small (generally less than 64–128MB).

For large servers, or for workstations intended for power users, a different design of MMU is required, in which the consequences of exceeding the MMUs limitations are less severe. In order to address these requirements, SPARC International has provided a design for the SPARC Reference MMU (SRMMU) which is based on a completely different organization: the table lookaside buffer (TLB). Instead of storing complete virtual–to–physical mappings in private SRAMs, the SRMMU retains the high–order bits of the most recent few (4096, in the Ross 6002 modules) discrete mappings in SRAM; these are obtained from the system page tables, which are stored in normal main memory. While the SRMMU normally doesn't contain completed translations for as much memory as a typical SRAM–based MMU, the cost of encountering a virtual address without a translation is orders of magnitude less: the TLB merely accesses a few location in memory, while the SRAM MMU must trap to the kernel to perform a fairly expensive operation, often involving tens of thousands of instructions. The cpu modules shipped with the SPARCsystem 600MP are the first Sun offerings to use the TLB–based SRMMU, and it is safe to presume that future high–end servers will also use the TLB design. The various MMU–related characteristics of past and present Sun systems is provided in Table 4.

Rule II. Configure SPARCserver–2's with 16 SRAM MMU contexts only in server which have fewer than 16 active processes. NFS servers commonly start 8–16 nfsd processes; most database backends also start fewer than 10 processes.

21. The notable exception is when the System V shared memory features is used extensively. Some of the popular database management systems make very heavy use of these mechanisms, and SunDBE provides some special mechanisms, known as *intimate shared memory* which optimize the virtual–to–physical translations for shared memory segments. See section 2.13.on SunDBE later.

patterns. Under this scheme, the instruction stream is protected from undue cache misses due to wild data access patterns (instruction access is nearly always serial, while data access is subject to the whim of application code). Generally speaking, Harvard caches don't provide any substantial benefit until the cache size grows to about 512KB[20]. Sun's caches to date are all 128KB per cpu or less, leading to the unified design.

As with main memory, the size of the cache, combined with the cache working set of the application, can have a significant effect on overall performance. When the cache working set exceeds the size of the cache by a large amount, the hit rate on the cache drops from around 97% to 65% or even lower, resulting in many delays waiting for memory recycles and (on multiprocessors) for memory bus or Mbus residency. This is known as *cache busting,* and this can result in very low performance. Some applications can bust caches because of their access pattern, rather than sheer size. These applications are typically matrix– or array–oriented mathematical applications which step sparsely through large data spaces in intervals which are larger than the cache line size. For example, consider an application which access data in arrays of integers. Let's assume that the application is to be run on a machine whose cache line size is 32 bytes. This application will run *much* faster if it access each element in its array sequentially than if it steps through the array in steps of 8: in the former case, every eighth reference causes a cache miss, while in the latter case *every* reference causes a miss. On many current Sun machines, the cache miss cost is twenty to *eighty* cycles (these miss costs are typical for machines in this class).

There is little that a user can do to monitor the cache hit rate. The tight coupling of the cpu and the cache hardware means that software cannot be inserted to monitor the hit rates. When the system runs an application that causes frequent cache misses, the cpu seems to be busy doing useful work. About the only way to avoid this situation is for the application developer to try experimenting with alternate data organizations.

2.2.3. Memory Management Unit (MMU)

So far we have ignored the translation of virtual addresses to physical memory addresses, but this too can have a significant impact on performance. The hardware used to perform the virtual–to–physical address translation is called the memory management unit, or MMU. Most Sun systems employ a *virtual address cache*, in which the cache deals solely in virtual addresses. In such systems, the MMU is placed between the cache and main memory, as shown in Figure 7. An alternate design places the MMU between the cpu and the cache, meaning that the addresses seen by the cache are physical, rather than virtual addresses. Physical address caches make the operating system less complex at the expense of more complex hardware in the MMU. Physical addresses are unique, while virtual addresses are not. Thus a virtual address cache must be at least partially flushed at intervals when a *cache alias* is discovered. The new TLB–based MMUs (see below) are better adapted to physical caches than Sun's traditional SRAM–based MMUs.

The traditional Sun MMU design uses a wide block of SRAMs to retain virtual–to–physical mappings. The SRAM memory is divided into equal–size blocks called *contexts*, which correspond roughly to processes – each process normally has its own set of virtual–to–physical

20. Patterson and Hennesey, *Computer Architecture*.

A write into the cache is even more complex, since writes to memory (whether to cache memory or to main memory) are even slower than reads from memory. When the processor issues a store instruction, the data is stored into the cache. What happens next depends upon the design of the cache. Under the write–through discipline, a store to the cache is immediately "written through" to the underlying memory. In very simple systems (*e.g.,* the Sun–3/50 or most PCs), the underlying memory is main memory. Since main memory is so much slower than the cache, the main effect of a cache in this design is to (greatly) accelerate memory reads while having virtually no affect on memory writes. To avoid this undesirable situation, a *write buffer* can be placed between the cache–write port and main memory. Usually this write buffer is very small – often just four bytes – but by permitting a cache write to proceed at cpu cycle speeds, it can have a significant effect on performance. The effectiveness of the write buffer was demonstrated by the SPARCstation–330. It had a write buffer capable of accepting two double–word stores, capable of accelerating back–to–back double–precision stores to memory. Although it used essentially the same chips as the SPARCstation–1[19], it turned in nearly twice (2.7 Mflops vs 1.5 Mflops) the floating–point performance, despite having only a 25% advantage in clock speed. Virtually all of the performance increase not attributable to the clock is due to the write buffer.

An alternative to the write–through cache with a write buffer is a different cache organization, known as write–back. Under the write–back cache discipline (sometimes called "copy-back"), a write to cache memory is treated as just that – a write to the cache. No write to main memory is immediately scheduled; the cache mechanism is responsible for maintaining the integrity of main memory by writing the data back "when appropriate". The write–back cache is most effective under circumstances when many writes to memory occur; these situations arise most frequently in mathematical computations, especially array– and matrix operations. The organization of the actual cache memory also plays a part here, especially under the write–back discipline. In order to avoid making many small references to main memory with its attendant set–up and take–down overhead, a cache is normally divided into sections of consecutive memory locations, known as *cache lines*. Cache lines in current Sun systems are 32 bytes. An entire cache line is read from or written to memory in a single burst, under either cache discipline. Most calculations proceed serially, so a sequence of writes through a write–back cache normally causes many fewer writes to main memory because the cache normally arranges for a sufficient write delay for the cpu to accomplish writes into a large part of a cache line before flushing the entire line back to memory. This means that a write–back cache also lowers memory bus utilization. This is especially important in multiprocessor systems in which many processors share a common memory bus.

Of course, the write–through cache is much easier and less costly to implement, especially in the multiprocessor configurations where a write–back cache is most useful (see *Mbus and other memory busses*, section 2.2.4.).

All current Sun caches are *unified* caches, meaning that a single pool of cache memory is used to cache all memory references out of the cpu, whether the reference is for instructions or data. The other popular design, known as the Harvard architecture, provides separate caches for instructions and data, on the theory that instructions and data have different access

19. For a short time, the SPARCstation–1 was shipped with a floating point unit based on the same TI8847 used in the SPARCstation–330.

However, to permit a single binary to operate on *any* SPARC machine[15], the compilers have a switch (–cg87) that permits generation of fully–software compatible code for integer multiply, square root and quad–precision floating point. Because code generated under this option is executable on any SPARC machine, –cg87 is the *default*. However, on a true V8 processor, compiling with –cg89[16] can result in much faster code.

> **Rule I. When configuring compute servers, be sure to configure with V8 cpus. Be sure that applications are compiled correctly for maximum V8 performance.**

For NFS and DBMS servers, there is basically only one variation between SPARC processors (other than the obvious clock speed issues): the presence or absence of special *bcopy* (block copy) support unit. The most expensive routine in the kernel is the kernel bcopy, used for copying tables and I/O from place to place. Providing hardware optimization for this simple but frequently–used routine can have a significant effect on I/O throughput. Actually, this support is built onto the processor complex and is not a part of the microprocessor at all. It permits a block copy to be performed without dedicating the integer unit to the copy, and at higher speeds. At present, only the SPARCsystem 400 processors have this support.

2.2.2. CPU Cache and Write Buffer

Like all RISC machines, SPARC processors execute – at least – one machine instruction[17] per clock cycle, usually operating on data already loaded into the cpu's registers. This means that the access time of a register must be 25ns with essentially no recovery time. The cache's memory must be similarly fast in order to avoid starving the cpu, although the cache may still require some recovery time after an access: not every instruction will require access to memory. In fact, one of the goals of the optimizing phase of the compilers is to rearrange code so that back–to–back references to memory are avoided. Typically, the cache is built out of static RAMs[18] (SRAMs) with access time equal to that of the cpu's cycle, but with a one– or two–read–cycle recovery. Thus a read from the cache happens within one clock cycle; if the next instruction also must read from the cache, the cpu will stall waiting for the cache to recover.

14. Specifically, the 4/200 and 4/100 processors did not have hardware square root, because their underlying Weitek 1164/65 floating point processors did not have square root. (This is also true of the Solbourne Series 4 and Series 5 machines.)

15. Solbourne systems up to the S4000 (*i.e.*, all desksides and racks up to but not including the S700 series) used V7 processors. Subsequent S700 and S4000 systems have V8 cpus. Most other Sun–compatible offerings introduced before mid–1990 also used V7 processors.

16. Do not depend upon the –fast option to correctly select the code generation option if you are compiling on one machine for execution on another.

17. This is a true native machine instruction, not to be confused with the "VAX MIPs" normally cited in the context of attainable performance.

18. Static RAMs are so called because data stored into them is retained without further intervention. Dynamic RAMs (DRAMs) require a periodic refresh to make make information "stick" in them. Although DRAMs are generally slower than SRAMs, the external refresh circuitry generally simplifies the design of each individual memory cell to the point that it is much denser and less costly (generally DRAMS are about a tenth of the price of otherwise equivalent SRAMs). Of course, this comes at a cost: the refresh itself complicates the design of the memory subsystem as a whole.

are not expected to commonly (and cost–effectively) exceed 70ns until approximately 1993, meaning that the processor–memory performance gap is increasing.

2.2.1. SPARC Processors

Due to the public nature of the SPARC architecture specification, there are a wide variety of SPARC processor implementations, from a number of different vendors. There are number of salient characteristics which govern various aspects of the performance of each chip: clock speed, number of internal functional units, internal bus width, load and store speed, number of register windows, and number of chips in the implementation.

Virtually all of the currently–available SPARC processors derive architecturally from the original 1987 Sun/Fujitsu SPARC design. Even features specifically called out in the specification as implementation–defined, such as the number of register windows, are essentially the same. Nearly all current implementations have a single integer unit and a single floating point unit, perform loads in two cycles and stores in three, have seven register windows, and use 32–bit internal busses. The primary variant is the Bipolar Integrated Technologies emitter–coupled–logic (ECL) implementation used in the FPS 5000 supercomputer, which uses 64–bit internal busses and a very high speed clock (67Mhz)[13]. Future CMOS implementations will see superscalar execution (more than one integer and more than one floating point units), single–cycle loads and stores, deeper register windows, and most important, 64–bit internal busses.

All current SPARC processors used by Sun except the SPARCstation IPC (*i.e.,* SPARCstation ELC, SPARCstation IPX, SPARCstation–2 and SPARCsystem–600MP) conform Version 8 of the SPARC specification, as defined by SPARC International. Earlier processors – all Sun processors from the original 4/200 to the 4/400 and the 4/40 (mid–1990) – conformed to SPARC Version 7. The differences between SPARC processors are primarily of interest when configuring the so–called “compute servers” typically used in floating–point intensive environments. The primary differences between Version 7 and Version 8 are in the provision of hardware support for integer multiply and divide, extended support for quad–precision (128–bit) floating point numbers and especially implementation of cache coherency for multiprocessing support.

Version 7 SPARC processors provided no hardware or instruction–set support for integer multiplication or division. This design decision forced the compilers to generate long (slow) sequences of basic instructions to implement integer multiplies. Version 8 processors provide a multiply–step instruction to accelerate integer multiply; thus V8 processors perform integer multiplies about ten times as fast as otherwise equivalent V7 processors.

In addition, a few early V7 processors[14] were offered without hardware support for double–precision square–root. Most V7 and all V8 implementations include hardware square–root.

13. ECL components are normally able to run faster than otherwise comparable CMOS parts. However, ECL typically consumes as much as ten times as much power, and is far less dense. The FPS cpu complex consists of about twenty chips and consumes well over 150W – excluding memory. The SPARCstation ELC, which uses CMOS parts (like all existing Sun processors), consumes under 20W for the entire cpu *including* memory. Of course, its performance is 65% less in integer and about 95% less in floating point, at 60% of the clock speed.

2. A Brief Survey of Sun System Architecture

2.1. Components

To understand the limitations of a system as a whole, we must consider the architecture and capabilities of some of the major subsystems that go into a server. Naturally, if any of these subsystems are saturated, they can become the limiting factor in the performance of a system. Configuration estimation, capacity planning and system tuning area all largely a matter of identifying and strengthening weak components. The biggest difference between capacity planning and tuning is *when* the analysis is done; otherwise, they amount to the same thing.

2.2. Processor Complex and Main Memory

The most obvious component of a system is its central processing complex, either uniprocessor or multiprocessor. The cpu chip(s) and main memory are the most prominent, but there are several other important parts: the memory management unit (MMU), the cpu's cache(s) and the cpu–to–memory bus. All are easily overlooked parts of the system, yet can have a significant impact upon performance. Unfortunately, there is no way to monitor either the cache miss rate or the cpu–to–memory bus residency without resorting to logic analyzers and other (very) specialized hardware. Various utility programs exist to monitor the effectiveness of the MMU[12]. The design of a typical Sun processor complex is shown in Figure 7.

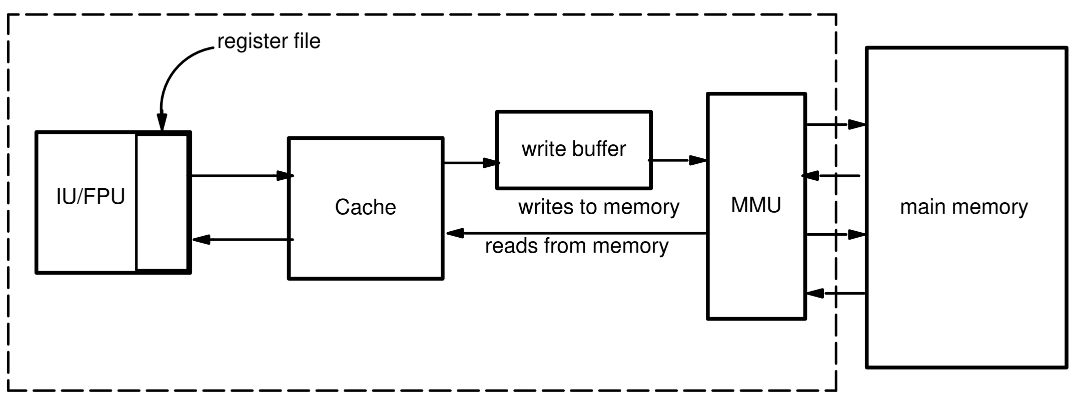


Figure 7. The organization of the typical Sun processor complex, joined to main memory. This organization employs a write–through, virtually addressed cache with a write buffer. This is the architecture of the SPARCserver–2 and SPARCsystem 300.

The design of the processor complex is oriented toward solving the most fundamental problem in modern microprocessor design: the increasingly large gap between the performance of cpus and main memory. Processor cycle times continue to decrease (speed up) rapidly, while memory has gotten generally denser rather than faster. In 1991, clock speeds are 40 Mhz and expected to increase to at least 80Mhz over the next two years. These clock speeds correspond to 25ns and 12.5ns cpu cycle times, respectively. This is far faster than the typical memory access time, which is currently approximately 80ns for reasonably fast systems such as workstations and 100–150ns for slower systems such as PCs. Workstation–class DRAMs

12. Utilities such as *pmegstat* are normally available from newstop.ebay.

devices can alternatively be configured with either ethernet or token ring interfaces. This board consumes one of the remaining Sbus slots.

The printing control system could be a SPARCserver 630MP, but this is probably overkill. There are only two devices which make use of the rasterizing facilities of NeWSprint, and a SPARCstation–2 can be configured with sufficient memory to handle two NeWSprint jobs. In this case, since the rasterizations will typically be 24–bit, 32MB of memory is appropriate. The system must have spool space sufficient to handle twenty printers. A 424MB disk probably is marginally sufficient, but a 1.3GB disk is probably better, and can be shared with network monitoring logs as well.

Finally, with a system as complex and ambitious as this one (meaning the group of three systems), especially when first implemented, it is wise to collect substantial usage data. Experience with SunNet Manager indicates that data collected at 1– to 5–minute intervals is the most useful for capacity planning. At these frequencies, the SNM logs can consume 40–60MB per week. This sort of disk consumption makes the 1.3GB disk look much more attractive. The printing control system seems to be a good system to run network console because it has spare disk space, and because SNM generally does not require very many resources in its monitoring mode. In addition, SNM requires a bitmapped display, and the printer system has one free Sbus slot in which to configure a GX framebuffer.

ruptible power supply must provide about fifteen minutes of power. SPOC indicates that each SPARCserver 690MP, including all of the Sun components consumes XXX watts of power. The third party vendors indicate that the total power consumption of the disks, tape and other equipment amounts to TTT watts. Multiplying (XXX+TTT) by the constant factor 1.43, each system requires a ZZZ VA power supply. This amounts to (XXX+TTT) * 3.412 = BBB BTUs per hour of heat dissipation.

This configuration will handle all of the requirements set down except two: printing services and serial or network connection for the Macs. Given the heavy load anticipated on the large systems, it seems more productive to configure the printing services onto another system. This system would have to support the two parallel line printers and ten printers presently connected to the VAX, as well as two high speed page printers obtained from Xerox. These printers also connect via a parallel interface. Finally, both a Matrix color film recorder and a RasterOps 24–bit dye–transfer color printer are contemplated.

To handle output to the Matrix and RasterOps machines, NeWSprint will be required, so a fast cpu is clearly required. Both are 24–bit output devices which consume substantial cpu and memory resources, in addition to larger than normal spool space on disk. Both devices connect through SCSI, in order to get sufficient bandwidth to efficiently transfer the large amounts of data associated with 24–bit output.

There are currently ten serial printers and two parallel printers, which will neatly fit onto two Sbus serial/parallel interfaces, which offer 2 parallel ports and 16 serial ports. The additional six serial ports may well be consumed by printers which are presently connected to standalone or networked PCs, or even Macs: with network printing becoming more accessible, existing printers can be relocated to make them more accessible.

This configuration also requires two parallel ports for the Xerox page printers. Several third parties offer Sbus cards with parallel interfaces, and these could be used; however, these generally will not sustain the anticipated nearly 1MB/sec transfer rates associated with 60+ page–per–minute printers[11]. The best fit is the Sbus SPARCprinter interface, even though there is no SPARCprinter to configure. This board includes a bidirectional parallel port capable of sustaining 2MB/sec. This causes a problem with Sbus slots: we now need four, and a SPARCstation–2 has only three. In addition, there are another two slots that could be productively used (see below). The best solution is an Sbus expansion box, which consumes one of the three slots in the box and provides four more in an expansion. If we configure such a box, the boards with the greatest bandwidth demand should be placed in the main box. In this case, we will put the two SPARCprinter interfaces in the system box and put the two serial/parallel interfaces in the expansion box.

Finally, the Macs now need a network connection, at least to the printer control system. This can be accomplished by configuring an Sbus board from IPT, known as the µShare. This board provides AppleTalk connections on a SPARCstation. This permits the Macs to be connected very inexpensively. Those stations which will make heavy use of the color printing

11. Although parallel interfaces can theoretically transfer at very high speeds, most parallel printers in the PC world cannot even accept data faster than about 80KB/sec, and many of the most popular are even slower, as low as 40 kilo*bits* per second. As a consequence, most parallel interfaces don’t operate much faster than about 100KB/sec.

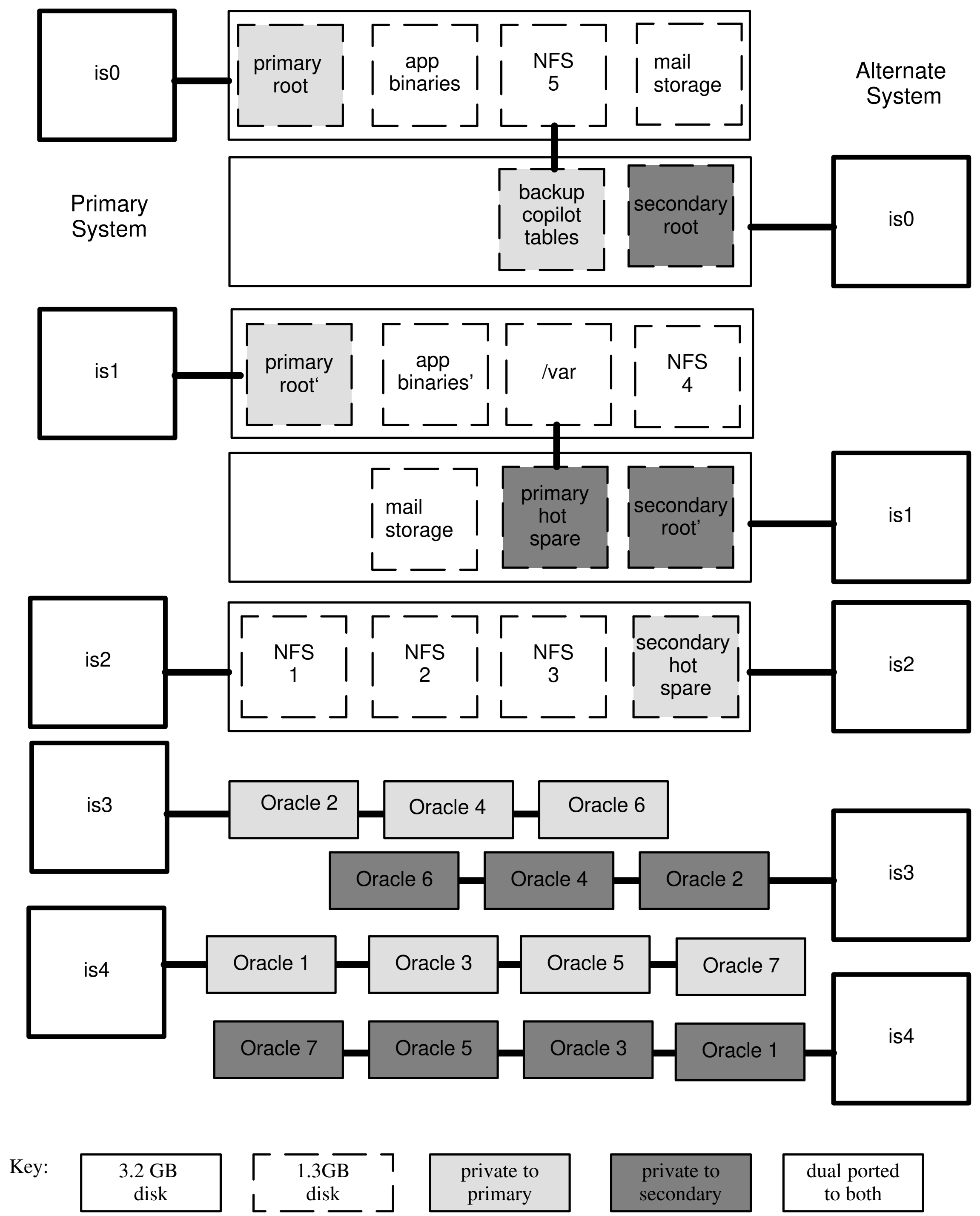


Figure 6. A good first–try configuration for the disks in Case Study 7. NFS 1–3 are all on the same string because all five NFS drives will be striped with an interlace of 12KB. With this interlace, any 56KB clustered operation will be spread across all five disks, rendering the ordering of the drives irrelevant. Note also that the PrestoServe is only to be applied to the root, /var and backup copilot drives. The 1.3GB disks are shown in their disk mounting trays, since each tray connects to exactly one controller on each system.

8mm helical scan tape drives – the customer has indicated that a 3rd–party tape stacker such as the one offered by VVV will provide the necessary functionality. The stacker provides access four identical 8mm helical scan tapes, providing the ability to dump the entire system without operator intervention. The tape stacker is a SCSI device, and it will be configured on the second SCSI bus along with the Fujitsu 2480.

Collecting all of these various facts together, we have the following hardware configuration for the primary system:

- four cpus (which means only 3 Sbus slots)
- 512MB of memory (next increment up from total demand of 470MB)
- three Sbus slots are used:
  - a SCSI/Buffered Ethernet for the SCSI tapes,
  - an HSI/S for T1 SNA to the corporate data center,
    - 256KB SNA to the 4381, and
    - 56KB external X.25 link
  - a TRI/S token ring interface, configured for 4Mb operation
- VME slots:
  - 1 PrestoServe
  - 5 ISP–80 IPI–2 string controllers
  - 2 Network CoProcessors
- 13 1.3GB IPI disks, mounted in four trays
- 7 3.2GB Seagate IPI disks, individually mounted
- Fujitsu M2480 SCSI IBM–3480–compatible tape drive
- VVV 4–way 8mm tape stacker

Because of the complexity of this configuration, we check to be sure there are enough slots:

- slot 1–2: 256MB memory
- slot 4–5: main cpu complex with 128MB memory
- slot 6–7: 128MB memory (with room for another 128MB)
- slot 8: PrestoServe
- slot 9–13: (5) ISP–80
- slot 14–15: (2) Network CoProcessors

Thus we have only VME slot 16 free for expansion (slot 3 is free, but no presently offered boards can be configured into it).

The configuration of the disks and disk controllers is important because we have configured the maximum number of string controllers. Additionally, the mirroring and striping options require careful attention to spread single logical accesses across controllers. A reasonable arrangement is shown in Figure 6. Care has been taken to spread access across controllers as frequently as possible, as well as to limit the number of active disks per controller to a maximum of four (the first two strings have six and seven drives on the string, but many of them are not running). Note that the primary and secondary systems share the NFS disks, but the alternate system actually has a completely separate set of database disks. Also, the secondary system has its own root/swap/usr disks (they are mirrored just like the primary's). In addition, the alternate has its own hot spare disk.

Both systems are to be protected against power failure by an uninterruptible power supply. The data center has a diesel backup generator sufficient to run the entire computer room in the event of a massive power outage, but it takes five to ten minutes to start. Thus the uninter-

widespread loss of service, so no other disks will be mirrored. The other disks are to be striped together by logical function, using the Online:DiskSuite striping/concatenation option.

Because we have configured Online:DiskSuite's mirroring and striping options, we must adjust our estimate of kernel cpu time. We are expecting to mirror the five drives which store the NFS data. This access will be dominantly file system random reads, so we expect to add about 30% overhead to the cpu time involved in performing disk access for the NFS clients. How much is this? If we assume that about half of the kernel time associated with the NFS service is spent processing disk traffic, this amounts to about 150ms per second. After overhead, this is about 180ms per second. Thus the total amount of time anticipated in the NFS service routines is about 330 ms per second.

Online:DiskSuite's mirroring option is also configured, but we have no information on how many I/O operations to expect against the mirrored drives. Since this includes the server's swap area, the actual figures are bound to vary widely. But a reasonable guess is to sustain 20–40 page faults per second, moderate paging for a system of this size and capability. At 30 page faults per second, we will consume about 3 ms per second of system time before mirroring. Adding in the 37% overhead (average of raw read and writes, weighted 2–to–1 in favor of reads – this is merely an educated guess), this is about 4 ms per second invested in the basic virtual memory subsystem.

The result is that during each second, NFS processing will consume about 300 ms of kernel time, MetaDisk overhead on the NFS disks consumes about 31 ms, the Oracle processing about 325 ms, directly connected ArcInfo users about 163 ms, the mail systems about 155 ms, and the SNA and X.25 interfaces will consume about 111 ms. This is a total of 1098 ms per second. Since the single–threaded Solaris 1.0.1 kernel permits only 1000 ms per second, the kernel lock will be the limiting factor in this system. (There is likely to be other kernel processing involved; however, we have covered the majority here.)

In this particular configuration, because we have two complete systems, we can configure around the single–lock kernel. We can logically divide the system into functional components and have some processed on the primary, the rest on the secondary. A reasonable solution would be to locate the SNA gateways on the secondary system, along the X.400 MTA processing. This would reduce the primary's load from 1098 ms per second to about 835 ms per second. The alternate would then be carrying 585 ms per second (it is *also* carrying most of the Oracle processing and some associated mirroring and striping). This is the most productive way of using a failover system – after all, the ideal situation would be for the alternate never to be used! Both systems have access to all of the data except the BackupCopilot data; if one system should fail, the other can be used to run all of the processing. In such a circumstance the single system clearly has the ability to shoulder all of the load, but the processing speed of the overall system would almost certainly be degraded.

By today's standards, the non–database data being managed on this system is far from exceptional. Nonetheless, with data resources of this magnitude, Online:BackupCopilot is an essential part of maintaining data integrity. It will be used to organize backups on approximately 6.5 GB of data, which means it requires about 1 GB of scratch space to store its indexes and other data. Because of the large amount of data – greatly exceeding the capacity of single

thing is certain, though: with an average of three cpu–bound batch jobs running at all times, there will be no problem utilizing the maximum of four cpus.

A few other peripherals must be configured: a 1/2” tape drive, to permit data exchange with other VAX and Prime systems within the company; the default CD–ROM of course is configured into the system. Data interchange with the corporate mainframe can be accomplished via the 1/2” tape, but current practice is to use 3480 tape cartridges. The Fujitsu M2480 tape drive is a SCSI tape drive compatible with the IBM 3480. The 1/2” tape and the CDROM are configured on the internal SCSI host adapter. However, because the Fujitsu must be configured in a separate cabinet (there is no space in the main system cabinet, even if it might fit physically), the 3480 drive must be configured on an second SCSI host adapter, due to cable length.

The customer has specified that the Oracle applications, especially the Oracle Financials, are critical to organizational functionality. In the event of a system failure, a ten–minute interruption in Oracle service is acceptable. Other functions can tolerate longer interruptions in service – up to an hour or so. To attain this high functionality, we will configure an alternate server system, capable of providing functionality in the absence of the primary. The most reliable way to continue database service is to use Oracle’s *Replicated Table Option*, which option actually builds two copies of each replicated table, and in effect runs two parallel database back end servers which listen to and act upon the same client request stream. Failover from a primary to a secondary server is automatic, as is database recovery. Of course, all tables must be replicated. Use of the Replicated Table Option requires no special hardware other than cpu, memory and disk resources sufficient to run the alternate server. In particular, this scheme does not require the use of dual–ported disks.

Of some importance in the actual setup of the systems is the incompatibility between dual–ported disks and a PrestoServe option. The system administrator must take care to configure the PrestoServe to operate *only* on drives which are not dual–ported.

In order to provide alternate hardware for other classes of service, the secondary system must have access to the same data as the primary, which will be accomplished via the IPI Dual–Port option. All data is to be stored on dual–ported disks, except the basic system disks (see below) and the 22GB used by the database back end: a total of 11 drives (plus a hot spare) and three controllers.

Both systems are to be resistant to disk media failure, by mirroring their root, swap and /usr file systems (total of four copies on two systems). Online:DiskSuite’s mirroring option will be used to provide this functionality. To avoid clogging up access the system swap space or access to /usr, each system must configure its system drive on two separate controllers. This configuration also provides marginally higher reliability reasons by removing the system disk controller as a single point of failure. Loss of the disk containing the myriad application binaries would result in a large number of users without service, so the binaries will be mirrored as well. With at least five partitions being mirrored (root, primary swap, /usr, the application binaries and a secondary swap), Online:DiskSuite’s hot sparing option can be used to further reduce the likelihood of a disk failure. Only one spare disk is necessary (per system). This provides about the same effect as three–way mirroring, but with slightly fewer disks and less cpu, disk and controller overhead. If any other disk fails, it will not result in

The message handling systems do not require significant memory. Their files are very small, and relatively few of them are open simultaneously. In addition, the agents are single–threaded and consequently will have only a few processes running simultaneously. With the PrestoServe to lower disk utilization and negligible memory requirements, the only significant resource consumed by the message transfer system is cpu. Unfortunately, with no detailed measurements or any other hard information to rely upon, there is little we can do to estimate how much cpu is required.

In addition to the electronic mail proper, we must configure a variety of other networking and communications products. SunLink DNI is required to support DECnet as practiced by the VAXstation community while the latter are moved to TCP/IP. No additional system resources need be configured for DNI.

The 4/260 provided 3270 terminal emulation via an SNA gateway. As the 4/260 will be traded in toward the new system, its SunLink SNA3270 license will be transferred. The old system utilized a 9600–baud synchronous modem connection, connected via one of the built–in serial ports. In the new system, we will want to keep as many of the serial interrupts off of the cpu as possible, and a higher speed line (at least 256Kb/sec) is desirable. In order to do this, the SNA must be connected to the synchronous line via a high speed serial interface, the SunLink HSI/S. Although it consumes an Sbus slot, the HSI/S permits very high speed serial transfers with very little cpu overhead. In addition, because it has four ports, a second port can be used to replace the existing T1 connection from the corporate data center to the 4381. With the older VME HSI board, managing a fully–loaded T1 (1.5Mb/sec) line consumed about 3 MIPs, and the newer HSI/S is slightly (about 15%) more efficient. This configuration will consume about 20% more bandwidth due to support for the 256Kb line to the 4381 and a 56Kb X.25 line (see below), so the end cpu consumption will be about 3 MIPs. All of this time is consumed in kernel processing, so this translates to about $3 \div 28.5 = 111$ ms per second of kernel time.

The X.25 connection to the outside world must be retained; this is the primary gateway for exchange of electronic mail with external sites. Managing the 56Kb line itself consumes no significant resources: the 56Kb is about 3% of another T1 line (this small amount of overhead is factored into the demand by the HSI/S above).

An Sbus token ring interface must be configured to accommodate those PCs requiring NFS service which are connected to their existing server by a 4Mb token ring. Because the NFS demand was treated as a single entity, disregarding the network media which carries the traffic, the cpu consumption is factored into the original NFS calculations. This is a valid simplifying assumption, as tests have shown that token ring and ethernet interfaces are of comparable cost to operate.

In order to manage batch processing queues, the customer suggests a productized version of the Network Queueing System (NQS), to meet the requirement for batch processing at a more sophisticated level than provided by SunOS/UNIX. This software originated at NASA and is widely available. Several companies offer productized and supported versions. All SAS jobs are to be submitted via NQS, which will schedule them at sufficiently low priority that they will have little impact on the interactive response of the large number of users. One

station. We need to configure 4GB of disk space for storing ArcInfo data. (This is in addition to the 2GB of data required by the workstations. In the existing facility, the workstations' ArcInfo data is stored on the Sun file server, completely separate from the timesharing ArcInfo data which is stored on the 9955.) There is no particular reason not to combine the two groups of ArcInfo data, leading to a total configuration of 6.5 GB of data. As with the Oracle situation, it is difficult to impossible to estimate how much cpu will be consume by ArcInfo, since the existing system is completely saturated. However, if the timesharing users load data as frequently as the workstation users, the system will need to service about 50 file system reads of 8KB I/Os per second to support the timeshare users (there are as many timesharing users as remote workstation users). This amount of I/O is associated with approximately 50 * 3.25 ms = 163 ms per second of kernel time. (The 3.25 ms is the cost of a file system random read against an IPI drive, normalized to the SS690's cpu speed.) Measurements of the ArcInfo processes running on the workstations indicate a working set of 8–10MB. Since twenty users will be sharing a single system, we can reasonably reduce this by 1–2MB (because they will be sharing executables and probably some other resources), so we should probably configure 20 users * 8MB = 160MB.

The next function is the electronic mail system. To accommodate X.400 we must configure SunLink OSI and SunLink MHS, which provides sufficient software to run an X.400 Message Transfer Agent (MTA). The heavy nature of the traffic, combined with the design of the SunLink MTA and the SMTP user agent (mail/mailtool), dictate the configuration of a PrestoServe board. Both the X.400 MTA and the sendmail/mail/mailtool SMTP mail software create and destroy large numbers of small temporary files in the process of delivering mail. Given the current load of 70,000 messages per day, mostly during working hours, this could well represent 50,000 messages in the eight hours during prime time, a rate of 2 messages per second. Incoming messages from X.400 being delivered to load mailboxes result in a temporary file being created and destroyed by the MTA and by sendmail, as well as opening and closing the destination mailbox. These are expensive operations because they include a substantial number of synchronous writes, either to create or destroy directory entries or to update the last–modified–time entry of a file. This is precisely the kind of operation that benefits most from a PrestoServe, even though the operations are all local to the delivery system. Use of the PrestoServe both accelerates the delivery operations and reduces the disk utilization of the associated disks (see section 2.8.). Because we will be short on Sbus slots, and because the system uses IPI disks, the PrestoServe should be a VME version.

Because the current mail service expires and removes messages older than three days unless the owner takes explicit action to save it, the disk space associated with message storage remains remarkably low in proportion to the massive traffic volume, slightly over 2.5GB. Although the message volume is expected to decrease slightly over the next 18 months, the average message size is expected to increase, by 5–10%. As a result 2.6GB of storage must be allocated.

The two drives configured for mail management achieve about 130 I/O ops/sec at normal rates. Since these are serial file system operations, this represents 130 * 1.5 = about 195 ms per second. After normalization to the 690MP, this is about 155ms per second of system time.

For the 10 PCs and 60 Macs, the primarily NFS activity is to load applications from a central repository (user data files will normally be copied to the server only when they need to be shared). A fast (33mhz) 386 PC demands about 10 ops/sec for 4–5 seconds. A Mac II is of comparable speed and demand. Macs and PCs, because they only load applications, end up demanding much less of an NFS server. Usually they load the application and then keep running it for quite some time – nearly always measured in minutes, and often hours. If we presume that each PC/Mac loads an application about once every five minutes, we arrive at a load of about 12 ops/sec for the 70 systems.

The combined load of 447 ops means two Network CoProcessors, and will consume about 300 ms per second of system time (because 435 Brown ops/sec is about 30% of a SS690MP's capacity, limited by kernel processing time). Between the workstations and the PCs and Macs, a total of about 2.5 GB of disk space must be allocated. Main memory required to support this function is probably minimal, about 32MB.

Next we must configure support for approximately an Oracle back end capable of supporting 90 Oracle clients – and the 90 clients. This represents both the existing Financials clients, as well as the new client survey clients. We know we must configure 22 GB of disk space to support this, and Oracle suggests (4MB + 90 clients * 1.5 MB per client) * 30% = 139 * 1.3 = 180 MB of memory for the back end. The front ends average about 1MB working set per client, so the total Oracle consumption will be about 270MB.

It is impossible to estimate the amount of cpu time necessary to support this application. It is clear that 90 clients and their back end can consume as much cpu as is available, but in the face of conflicting demands (NFS service, timesharing, *etc.*), it is not clear how much cpu will be available to support the database systems. We know that the existing applications consume nearly all of the 4381's cpu, but as with the NFS demand on the 4/260, we don't know how much demand is getting "dammed up" behind the 4381's cpu.

With this many clients, it is safe to presume that access to the database disks will be random. If we use the 1.3GB disk, we will need 17 drives to store the targeted 22GB. At the optimal 3 drives per controller, this would consume all five potential disk controllers. Because this system is clearly going to be *very* busy, we do not want to accept the compromise of configuring many additional disks on the IPI strings. The customer is willing to add third–party disk drives onto his maintenance contract, so a browse across the disk drive industry turns up an 8" Seagate IPI disk drive (the ST83200K) with dual ports and a 3.2GB formatted capacity; it transfers at 3.0 MB/sec, but otherwise is comparable to the Sun 1.3GB disk. With this drive, we need but 7 drives, a much more comfortable quantity.

Seven IPI drives should be configured on two string controllers. At maximum I/O rate, we can run six drives on the two controllers at full rates, or about 6 * 60 = 360 ops per second (at 65% utilization – see Table 8.). This translates to a cpu cost of 360 * 0.9 = 325 ms per second of I/O time. This presumes that most I/Os are 2KB raw operations.

Because the database is Oracle, this system will likely benefit from SunDBE, especially since it must handle the very large back end server.

The next function to configure is ArcInfo. Typically, about twenty users are logged into the Prime 9955–II, running ArcInfo in a Tektronics emulation window from a PC, Mac or VAX-

with four 28.5 MIP processors probably has enough cpu power to replace all of these machines, but it is hard to gauge the intensity of each class of work. Of particular importance is the fact that all of the major systems are running at capacity now, and that additional services are *already* being demanded. Another concern is that we have at least four different platforms, and measurements taken on one are exceedingly hard to translate from one platform to another. This is aggravated by the multiplicity of operating systems: the VAX runs VMS, the Prime runs PrimOS, the 4381 is running MVS *and* TSO (and the 4/260 runs SunOS, of course)!

The system will be a *very* complex one, in which a wide variety of options will be operating together. The proposed system will, for example, provide some level of service in all of the following communications areas: ethernet, TCP/IP, NFS, DECnet, token ring, SNA, X.25, X.400 and SMTP. Future expansion into faster token rings and/or FDDI are not out of the question. The possibility of interactions between subsystems which are presently unrelated is clearly present.

To configure this system, it is probably easiest to consider the various functions separately. NFS processing is the simplest, so we will start there. We need to provide for NFS service to 63 workstations, about 10 PCs (most of the PCs will continue to run 3Com software for now), and about 60 Macs. Most of the workstations are the slow 3–MIP VAXstation 3100's. The nearest entry in the NFS Client Demand Appendix is a diskless Sun–3/60, which produces an average of about 1.5 NFSops/sec. The VAXstations are equipped with more memory (16 or 24 MB), but they also typically run larger financial models and schedule planning software. Since they have local disks, they will probably generate somewhat less traffic. An average demanded load of 1.0–1.25 NFSops/sec is a reasonable guess. There are 43 VAXstations, so this means 53 ops/sec to the server.

Most of the SPARCstations were purchased to run ArcInfo, as such they normally obtain data from the 4/260, which is equipped with a 327MB SCSI, 2x280MB SMD disks and 2x688MB SMD disks. The SPARCstations, which are much faster than the VAXstations, can and do consume all of the remaining NFS performance available from the 4/260 (a vanilla 4/260 using the built–in Intel ethernet interface is capable of 75–100 NFSops/sec). Each of the SPARCstation IPX's can demand nearly 35 ops/sec while loading or saving a dataset from the server. For this reason, measuring the existing 4/260 isn't particularly productive: it is very clear that the system is completely saturated, and how much the new system will have to supply is still unknown. Increasing the performance of the server will virtually guarantee a drastic increase in the demand imposed by high performance clients, since the clients are typically waiting for the completion of one NFS request before submitting another.

Investigation of the files stored on the server indicate that the typical ArcInfo files are about 6 MB in size. If we presume that the SPARcstations perform either a load on the average of once every five minutes, we get 20 SPARCstations * 6 MB = 120 MB or 15,360 read ops per five minutes, or 51 ops/sec. But this is just reads, so if we presume a mix similar to the Brown (dataless) mix[10], we have 51 $\div$ 0.13 = 392 ops/sec. Added to the 53 ops/sec that the VAXstations are demanding, this is 435 ops/sec.

10. In the Brown mix, reads are 13% of the operations.

VAX and the Prime, most of those session are usually active. The system normally runs with the cpu 90–100% busy; the disk channels are also near saturation. The primary applications are customer information retrieval, some statistical analysis, and interaction with Oracle Financials. The financial data is updated daily via 3480 tape cartridges from the company's main data center, located four miles away. Although there is an SNA connection to the data center, it is a single T1 line and is not sufficient to transfer the approximately 500MB of data per day. The SNA line provides remote terminal access to the corporate data center. The corporate data center does *not* permit submission of batch jobs. For political reasons it is impossible to increase the bandwidth of this channel. There are presently about 15GB of storage on the 4381, but these are completely full and a disk purchase for this machine is being deferred solely because the system is about to be replaced. The system managers estimate that this should grow to about 22 GB within the next year as additional applications come into production (mostly the client survey system). There are two high–speed line printers connected to the 4381. Like the system itself, they are leased, so while their function must be replaced, the printers themselves will not be retained. The terminal session onto this system come from the PCs, which are connected to the IBM via a 4Mb token ring network, gatewayed by a 3Com 3Share server.

Five of the six of the 3Com servers are Intel 80286 machines, and like the large systems, they are quite saturated. The exception is a new Compaq SystemPro, which works well and is to be retained as the main 3Com server. In addition, the SystemPro is the mainframe gateway system. Most of the anemic DP budget in the past three years has gone to buy or upgrade PCs and Macs, and now most of the client PCs are at least 386SX systems, so it is no surprise that the ancient 286 servers are overwhelmed. Most of the Macs are Mac II's of one type or another – which means they can accept ethernet or token ring interface cards. Some of the PCs (about ten) will be converted from 3Com to PC–NFS. The others will be retained on 3Com for now, although over time they will be transitioned to other networking solutions.

If the new system can provide the new printing and output services, the Macs are to be connected to the network in order to take advantage of them. Otherwise, they can continue to be connected to the hosts by serial line.

The elderly Sun provides file service to the SPARCstations and VAXstations – and is now clearly being overrun by the demands. Although the actual load being demanded by the workstation community is not especially high (about 40 ops per second, with very few writes), the delays associated with the NFS service are becoming unacceptable – the NFS latency of the 4/260 is simply too high to sustain high performance on the newer workstations. Besides file service, 4/260 also provides SNA gateway service to the Sun workstation community. The VAXstations which require SNA sessions log into the 4/260, although only one or two sessions are typically found at any given time.

Given this much information, it should be easy to configure a new system, right? Actually, it isn't at all easy to formulate a reliably accurate estimate of what computing power is required in this situation. The volumes of data being managed – about 35 GB projected 18 months away – combined with the requirement for dual–ported disks strongly suggest that a SPARCserver 690MP is the platform of choice. The VAX is about a 6 MIP machine, the Prime is about a 7.5 MIP machine, and the 4381 is about 6 MIPS, so a SPARCSserver 690MP

maining Macs and workstations are used for electronic publishing and advertising layout. FrameMaker is the publishing software of choice; a wide variety of Mac packages are run in the advertising community, most of which produce color slide output which is transmitted by modem to an (expensive) service bureau. Some of the Macs are connected via AppleTalk, but for the most part, the only connections between the average Mac and the rest of the computing facilities is a serial line running to the VAX. Finally, about 40 X–terminals are provided to users with light demands. Normally, the VAX runs three to four simultaneous SAS jobs. The data center also supports a hodgepodge variety of printing services, ranging from impact line printers to color printers. The customer would additionally like to support color slide output and possibly a deep color dye–transfer printer.

The VAX is used primarily for statistical analysis with SAS (in batch mode), as well as to provide general timesharing service, including word processing. It presently has about 5.5GB of storage. This is expected to grow very moderately over the next year, possibly to as much as 6.5GB. (Because most of the files are old and infrequently accessed, management feels that the disks are underutilized and does not feel the need to purchase many more drives.) It normally supports about 50 timesharing sessions and three or four low–priority batch jobs. Most of the timesharing sessions are idle, as the typical load average on the VAX is about 5 (this would imply that about 20 sessions were actively working). With the batch jobs, the load average is about 8. All of the X–terminals are connected to the VAX via an Ethernet. There are two parallel–interface line printers and ten serial–interface laser printers connected to the VAX.

The Prime was originally purchased to be the hub for an externally–managed X.400 electronic mail service, as well as some limited use of a geographic information system, ArcInfo. Inevitably, the despite the original intent to run only three or four ArcInfo sessions, usage has grown dramatically over the years and now nearly 20 ArcInfo sessions are common. The system normally supports a total of about 100 timesharing sessions. Like the VAX, the vast majority are idle. The externally–managed mail service can be dispensed with if the new system can support an X.400 private domain. Essentially all of the terminal sessions are idle, and their only purpose is to check for electronic mail. The Prime presently has 4.2 GB of disk space, divided about equally between mail storage space and ArcInfo databases. Mail traffic is unusually heavy: an estimated 70,000 messages are handled by the MTA per day, mostly during normal working hours. (A number of the custom applications distribute reports automatically and otherwise communicate electronically via electronic mail.) The message traffic is not expected to increase by any great amount; in fact, it is likely to decrease slightly. On the other hand, the use of the ArcInfo seems to be growing rapidly, and the projected demand for ArcInfo database space appears to be almost 4GB within 18 months. The system administrator for the Prime indicates that the existing system is presently running at capacity. All reasonable attempts have been made to tune the system, and this process has been fairly successful: ArcInfo consumes virtually all of the available cpu power, and the mail system seems to be bottlenecked on disk access. The 9955 is connected to the VAX and to the outside world via 56KB X.25 connections.

The 4381 runs MVS and TSO, and is used primarily for Oracle processing, both custom applications and Oracle Financials, although large SAS jobs may be submitted at low priority. Normally, the system supports about 40 timesharing sessions on 20 initiators, but unlike the

considered viable for some purposes (GIS in particular), distributed processing is not a primary goal of this installation. Rather, substantial gains in throughput are desired, using the current computing model. Only after the rationalization effort has been completed will management consider wholesale movement to distributed computing. The process of migrating from old proprietary environments to more powerful, open systems is – correctly – viewed as a very complex one, estimated to take a year and a half. Although the provision of significantly more powerful computing resources is expected to spur additional consumption of computing power, the system to be installed now is to be able the existing load in addition to projected usage eighteen months from installation.

The user community and their primary usage looks like this:

Marketing		
Organization	Hardware	Timesharing Application sessions (platform)
Analysis	14 SPARCstation IPC 6 SPARCstation IPX 5 VAXstation 3100 10 386SX PCs 30 Macs 8 X–terminals	15 ArcInfo (Prime) 10 Oracle Financials (IBM) 25–35 misc. timesharing (VAX) 15 Client–Survey (new) 25 email (Prime) 3 SAS jobs (VAX, batch)
Products	38 VAXstation 3100 41 Macs 30 PCs 10 X–terminals	10 Oracle (IBM) 30 Client–Survey (new) 5 ArcInfo (Prime) 15–20 misc. timesharing (VAX) 20 email (Prime)
Finance	10 PCs 8 Macs 16 X–terminals	20 Oracle Financials (IBM) 12 email (Prime)
Advertising	74 Macs 9 PCs 9 X–terminals 1 SPARCstation–2	15 misc. timesharing (VAX) 3–5 Oracle (IBM)

The new system is to be provided in duplicate, so that critical applications can continue to function even during a system failure. A period of parallel operation must be provided, while the Oracle Financials are operated on both the 4381 and the new system.

The requirements include provision of NFS service to approximately 70 workstations (20 SPARCstation–IPCs and IPXs, and 43 VAXstation 3100's), as well as part of a community of 59 PCs and 153 Macs. Virtually all of the workstations have their own disk, but current policy dictates that users not store data on their disks. In addition to the existing applications, a new client survey application will be brought up on the new server. To reduce licensing cost, the new database will be built on Oracle. Most of the PCs run typical DOS applications, and must have access to the various Oracle databases via 3270 terminal emulation. The re-

dom I/Os per second at an average cost of about 2ms per operation (some will be serial writes costing about 5 ms, but most will be random reads at about 1 ms), we can consume at most 400 ms per second of kernel time.

Combined with the 125ms for data transfer, it's clear that we will consume most of a cpu in the kernel, but that there is plenty of kernel time remaining (even considering that the line printers will consume some kernel time). Since we have the two SPARCprinters, which will be running much of the time, a four processor system seems in order.

This system must support up to 75 dial–in asynchronous lines. While this could be handled with five 16–port ALM–2's – there is certainly plenty of space in the rack – the large number of interrupts associated with this many active liens suggests that we use an Ethernet terminal server to provide this connectivity. The terminal server will provide as many ports but will require significantly less context switch overhead in the system. The fly in this ointment is the parallel interfaces on the old line printers: the ALM–2 is the most logical way to connect these. The best solution is to connect 32 modems via the ALM–2's, along with the two printers. The remaining modems (and any future additions) are connected via the terminal server.

The system's built–in Ethernet port can be used for connecting the terminal server. There is no need to provide a simultaneous backbone connection since the back end server already has such a gateway and traffic from the front–end machine to the backbone is expected to be very light; the path from the front–end to the gateway is via the very wide FDDI ring, so media contention is not an issue. Also, the relatively low volume of traffic (75 terminals * 2400 baud on a dedicated ethernet) means that there is no requirement for a Network CoProcessor interface.

The front–end system has no need for the specialized server software: there will be one user file system containing all of the binaries and providing temporary space; another drive will be used for swap space and line printer spooling space. There is no critical data to be mirrored, and in fact very few user–created files will reside on this system. Neither DiskSuite nor BackupCopilot is necessary. The Sybase clients do not make use of SunDBE, so this is not required either.

1.3.1.2. Case Study 7. An Enterprise–wide Server Complex

Finally, consider a very large server complex for the marketing organization of a Fortune 100 company. The vice president wants to rationalize and collect his diverse and largely incompatible computing resources, which today consist of a VAX 8650, a Prime 9955–II, an IBM 4381, a Sun–4/260, and a half–dozen disconnected PC and Mac networks. The DP budget has been virtually non–existent for the past three years. As a consequence, all four of the primary systems are either heavily loaded or outright overloaded. Now, with a reasonable budget finally available, wholesale replacement can be contemplated. All four of the primary organization in Marketing are affected: Analysis, Finance, Product Development and Advertising. Unlike many installations, the various departments share resources in the marketing data center. Although the corporation has a large data center, very little of marketing's work is transferred there.

The VAX in particular does not provide the desired response time, especially for batch jobs, and faster cpus are viewed as the most appropriate alternative. Although workstations are

Configure in a fudge factor, since this kind of estimation is fraught with error. This is the basic approach that we followed in Case Study 4.

1.3.1. Examples.

There are basically no rules specific to configuring this class of system, because of the diversity of application, so most of this discussion is is case studies. Note that it is *much* easier to configure an essentially homogenous system.

1.3.1.1. Case Study 6. Front–end Server for the Ad–Hoc Query System

In our first multiuser case study, we return to the Freedom–of–Information application in Case Study 3. (Refer to that case study for additional details.) Now we will consider the configuration of the front–end server which we deferred earlier. Basically, this machine is expected to run the Sybase client front end for 40–75 dial–in terminals. The only other responsibilities this machine will have is driving four printers: two old line printers (2500 lines per line per minute) and two SPARCprinters. The customer has a 4/390 which will be upgraded to a SPARCserver 690MP of the appropriate configuration.

Measurements earlier showed that each client process consumed the rather surprising amount of 3MB of memory. This implies that handling 75 users will require a 256MB configuration. The systems analysts should double–check this situation (this isn't an unheard–of size, but it is large for DataWorkbench clients). This is probably attributable to the extremely varied nature of this type of DBMS user and is unlikely to be reduced by a large margin. If the clients cannot be reduced to less than 2MB each – probably through some sort of query optimization[9] – it is much safer to configure 256MB (2MB per client would permit a 192MB configuration).

Since this system will be receiving about 625 packets per second from the back end system, we will be consuming about 0.2 ms/packet * 625 packets/sec = 125ms per second of kernel time for data transfer. An FDDI interface must be configured into the system to connect to the ring; either the VME or Sbus versions will do. Since we are upgrading from a 4/390 to a 690MP, either is an option. This system has plenty of VME slots available, and also has no contention for Sbus slots. The Sbus board is somewhat more efficient and costs substantially less. In this case, we already have a Class–A–to–Class–B concentrator, so this is the preferable option.

Relatively speaking, we don't have much disk activity – it's all virtual memory traffic, primarily page faults against the Sybase front–end binaries or swap space and files being written onto the printer's spooling area. This system's disk space requirements are low: the largest consumer is the printer spool queues, which sometimes reach 750MB in size. The three 1GB disks remaining with the system (it had eleven, but eight were transferred to the back–end system in Case Study 3.), since the client processes should not page or swap much since we have configured so much main memory. With so few drives, a single controller (already in the system) is quite sufficient. Since the three drives can only support about 180–200 ran-

9. This would likely involve some sort of normalization of the data tables, or possibly better indexing schemes. There are some third–party products on the market which perform or suggest normalization and/or optimization of relational databases.

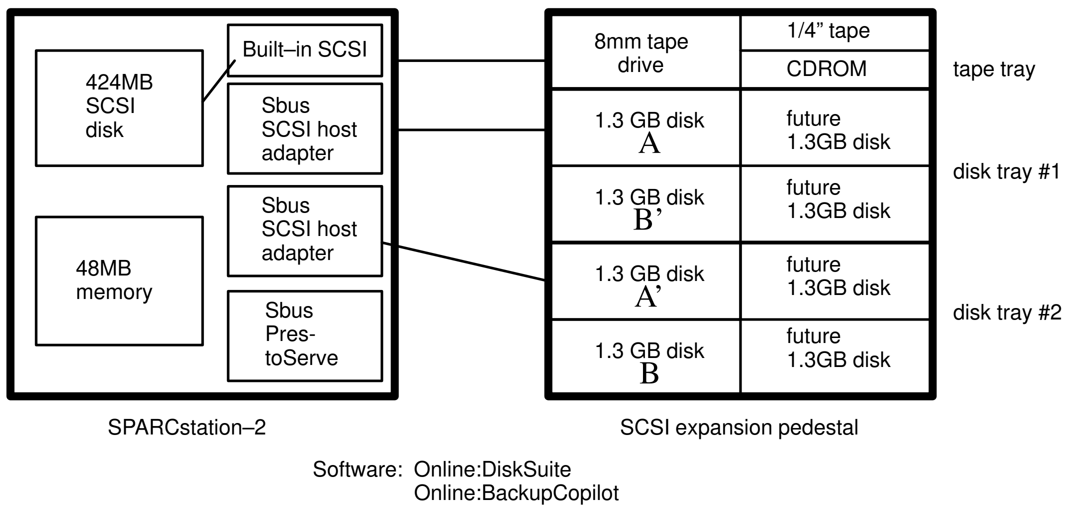


Figure 5. The organization of the MCAD community server.

the disk accesses will lower the bus utilization to the critical point. If the database doesn't grow more than the doubling in size forecast by the customer, this won't be a problem. But if the database grows past the 2GB forecast *and* the number of engineers grows substantially, SCSI utilization could become a problem.

1.3. Configuration of Multi–User and Other Mixed–Use Configurations

Multi–user servers are even more diverse than database servers. Following the minicomputer tradition, virtually anything and everything gets done on this class of system. Whereas single–purpose servers typically make large demands of a specific chain of resources, multiuser systems often stress diverse hardware and software resources. Even today, when powerful computers are very common, it is rare that a server is used for only a single purpose. Because servers are shared resources, many users have access to them, and inevitably their demands become more and more diverse. This is both a blessing and a curse to the configuration planner. It means providing more kinds of resources, and ensuring that each of those resources does not become the limiting bottleneck in the system, but the nature of the workload often provides opportunities to overlap usage of different resources, with fewer requirements for maximum capabilities in many categories.

To a degree almost unknown in NFS and DBMS servers, configuring a multiuser system is largely a matter of educated guessing. Usable data on how a system will be used is rare, and the diverse and changing nature of the production workload means that even when data is available, it often applies only at fleeting instants. The best way to attack one of these problems is to consider various classes of work separately, and then try to combine resources to accommodate all classes simultaneously. Each major class of service must be considered and broken down into the basic architectural building blocks (cpu, memory, bus, disk, *etc.*). Each of the basic services must be configured to handle the sum of the simultaneous demands.

Memory requirements are hard to measure in the absence of real data. The MCAD vendor recommends 16MB plus 2MB per active user. We have ten users, thus calling for 36MB. We can probably get away with 32MB – the ten users are unlikely to be simultaneously active – but since we will be supplying OpenWindows, MCAD and Interbase binaries for the entire community, it's probably safer to configure 48MB.

Networks are easy. Because there are few users and a low duty–cycle, Ethernet is an excellent choice. A 4Mb token ring would be too slow, since the transfers are all relatively large and the low media speed would quickly become a bottleneck. A 16Mb token ring would work out well, but requires an Sbus slot that we don't have, as we will see shortly. And the volume of data is insufficient to warrant the expense of installing FDDI (and again, we don't have the Sbus slot). There is plenty of cpu power to push the packets around, so the relatively small MTU won't be an issue.

Finally, we need to configure the peripheral busses. We need to disk space as follows: 2GB of user storage, mirrored; about 450MB for the operating system, swap space, and application binaries; and about 15% of the 2GB, or 300MB for Online:BackupCopilot's directory space. This is a total of about 5GB, easily managed on four 1.3GB disks. We will also need a CDROM, a 1/4" tape drive, and an 8mm tape drive, for a total of seven SCSI devices. The most convenient packaging is the SPARCserver–2 (there is clearly no necessity for a SPARCserver 600MP) with all of the peripherals except the 424MB disk mounted in an expansion pedestal. We will configure the 424MB disk and the tape devices on the built–in SCSI host adapter. Because we expect to run online backups during working hours, the built–in SCSI host adapter essentially isn't available for disk drives – the SCSI bus utilization will be too high to support high–performance disks – and since the 424MB disk can't be configured on any other SCSI host adapter, it will be used to hold the Backup:Copilot directory tables. The other drives will be located in pairs in separate trays in the expansion pedestal; they will be used as mirrored pairs, connected to separate SCSI host adapters. This leaves us with two Sbus slots devoted to SCSI host adapters, leaving the last slot for the PrestoServe board.

If another Sbus slot were required, for example to connect to a token ring, the PrestoServe would be sacrificed. In this application, Interstream's eNFS cannot be used to replace the PrestoServe, since there is no NFS traffic: the PrestoServe is being used solely to accelerate local synchronous writes.

This configuration will suit the specifications quite well; it is diagramed in Figure 5. It has considerable room for disk expansion because we can add two disks each to the Sbus SCSI host adapters: since we our disk access is dominantly serial, we can safely configure four disks onto each host adapter without fear of driving the bus utilization into the critical range. If the design group is reasonably constant in size, this is fine. The limiting dimension of this configuration is Sbus slot availability, which will not become an issue unless the size of the group increases dramatically. If the group were to double in size, accesses to the database could well come much more frequently, changing the typical disk access from serial to random. Such a circumstance poses two potential problems. First, the cpu could become a bottleneck, although this isn't especially likely: if the disk access becomes random, the cpu will probably spend a lot of time waiting for disk seeks. More likely is that the random nature of

memory than the upper limits is in order. This leads us to configure (4 MB + 22 clients * 3 MB per client) * 1.75 = 122.5 MB for Oracle. Thus a 128MB configuration should suffice.

1.2.9.3. Case Study 5. Database Service for a community of MCAD users

Finally, let's configure a server for a design group at a small industrial firm. The group designs aftermarket automobile parts, and is switching its design efforts from standalone PCs to to a commercial MCAD system running on workstation clients. The user's goal is to increase productivity by using better software, while simultaneously providing a more robust operating environment than provided by his PC's. Automated backup and redundant disks are essential to achieving these goals. Support of the MCAD system is the customer's sole and primary goal. Anything else that comes for free is fine, but management isn't willing to pay for anything else.

The MCAD system is built on Interbase, a commercial object–oriented database management system. There are eight designers, in addition to a product manager and a librarian. Although the customer has a large amount of existing data stored on his PCs – about 200MB per PC (totalling about 1.5GB), he is unable to provide a reliable estimate of what size the data will be after conversion to the new format. Investigation of some existing drawing indicates that most of the drawings are between 5 and 10 MB each. The MCAD vendor suggests that the converted database will likely be somewhat smaller, totalling about 1 GB. The customer requests that we plan for a database of about twice what the current data will consume.

The first thing to do is to get an idea of what kind of load this represents. Since there isn't an existing system, we're at a big handicap. Interbase runs in the UNIX file system (Oracle, Sybase and Informix customarily do not); consequently a PrestoServe is a good idea if we will be doing many updates to the database[8]. This also tells us that most I/O's will be done in 8KB or 56KB units, rather than the 2KB ops usually seen when raw partitions are involved. We know that the typical operation will be to load or store a drawing, which we think average about 7.5MB. Unlike many server configurations, this small–scale server will likely experience primarily serial access to its disks. Because there are few clients and the primary operation will happen relatively infrequently, it is unlikely that multiple operations will occur at the same time, as normally happens on large–scale servers. Thus we want to tailor the system for optimization of serial access. Reading or writing one of these drawings necessarily involves moving 8–9MB of data serially, by the time the Interbase overhead is figured in. This is about 1000 operations. The 1.3GB disks can achieve about 450 sequential ops/sec, and can thus perform the disk–related portion of this in 2–3 seconds: see Table 7. The disk activity will thus consume nearly 90% of the cpu during the disk–intensive portion of the operation, and Interbase consumes cpu resources on both the server and the MCAD client. From watching perfmeter during demos, it seems that the back–end gets soaked for about three seconds for each load/save, and additionally yields 100% utilization for short intervals (0.5–1 seconds) at periodic intervals. Otherwise, the cpu seems to be idle most of the time. Clearly, the 28.5 MIP cpu of the SPARCserver–2 is sufficient for this application.

8. Any synchronous write to the UFS file system will be slow unless a PrestoServe is used to accelerate it. Since databases typically must use synchronous writes to ensure data integrity (for example, during a two–phase commit), PrestoServe is a good idea. See Section 2.8., *PrestoServe*.

This isn't a significant load (14 ops per second, spread across 14 drives) unless it happens to occur at the same time as a Query 1 and a Query 2. The updater has very low priority compared with the queries, so this process can safely be ignored.

The last major activity on the system is the data collection back end. The load imposed by this process is infrequent but heavy. It is clear from the preceding analysis that there is no remaining kernel time with which to handle this task if it occurs while both Query 1 and Query 2 are active in the server. For now, this process must be run at the lowest possible priority, hoping to stay out of the way of the normal user processes. When Solaris 2.0 arrives, this processes can be returned to normal priority. The final configuration is diagrammed in Figure 4.

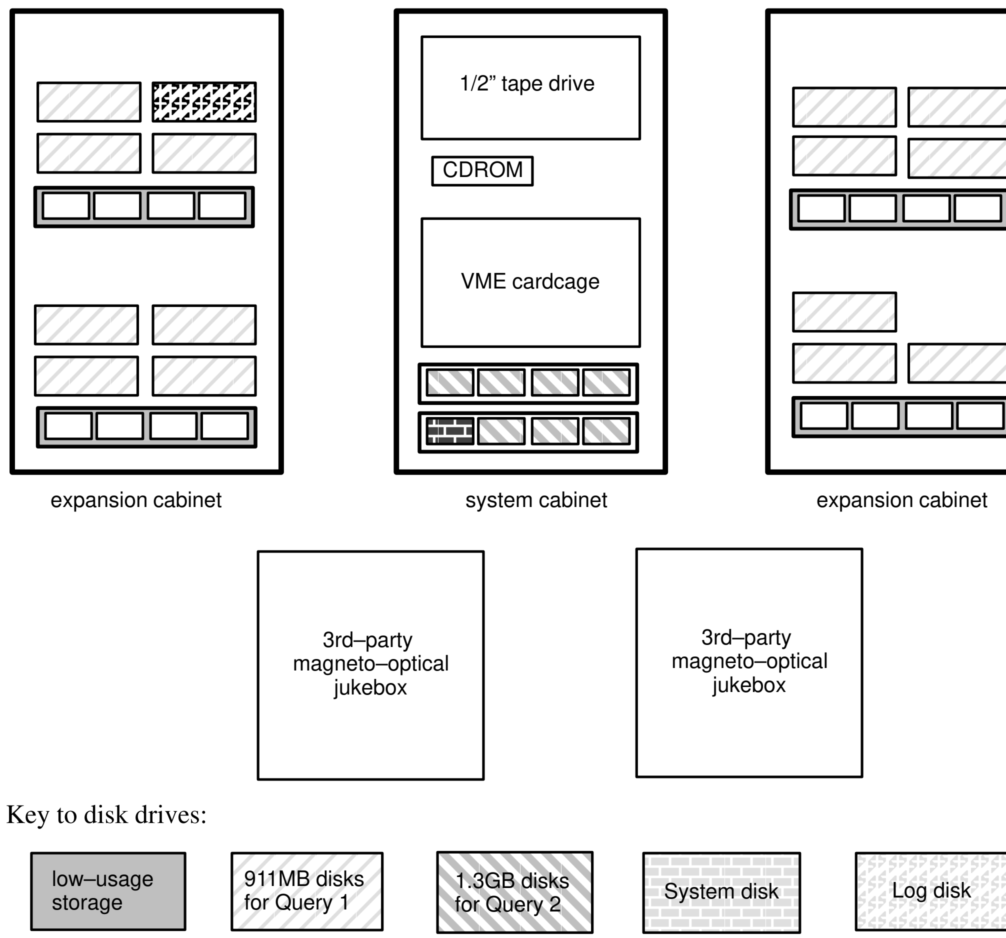


Figure 4. Disk layout and general organization of the GIS database server. The physical organization of the disks in the expansion cabinets is due to the requirement to mix heavily– and lightly–used (relatively) disks together on the same controllers.

For memory, Oracle recommends 4MB + 1–2MB per client + (30–50% more). The pieces of data being manipulated in this system are much larger than typical, so using a bit more

of processor. The network is clearly the bottleneck here. The 1MB/sec rate is 233 packets per second, which represents 150ms of kernel time per second. Total kernel time consumed processing Query 2 is 330ms + 150ms = 480ms per second. Total Measurements showed about 30% of a single cpu in user time during this query.

Between the two queries, we are consuming 600ms + 480ms = 1080 ms per second of kernel time. Unfortunately, due to the single–threaded nature of the Solaris 1.0.1 kernel, we have access to only 1000ms of kernel time. We cannot achieve the maximum search rate on Query 1 *and* the 3–5 second retrieval rate noted above, since there will not be sufficient processor power below the kernel lock. Fortunately, we are missing by only 8%, and only if the network is not bogged down (if the network bogs down, the system will spend time waiting for the network instead of continuing to process; this permits other – ready – processes to continue in possession of the kernel lock). There is plenty of user space computation to be done – we have already completely consumed a full 28.5 MIP cpu in user processing, but we can configure up to three cpus for user (plus the one devoted to kernel processing[6]).

In the process of accommodating the two important queries, we have already configured as many disk controllers as are permitted. Yet we will need to connect to an additional 20 GB of data – 16 more 1.3GB drives. Because these are low–usage items, they can be connected to the end of the performance–critical strings without fear that they will significantly detract from performance. However, the system and database administrators will need to take care not to store data on these drives that will used frequently. Should additional data access be necessary, it is probably best to obtain rack–mountable SCSI disks from third parties for this purpose. Higher capacity disks aren't the answer, since in this case the capacity of the drives is well balanced against the controller's ability to provide optimal access to the drives[7]. A denser disk would permit additional storage to be connected (2.0–3.2 GB drive mechanisms are currently available in the industry, with a choice of SCSI or IPI interfaces). Using denser disks would lower the disk space–to–disk–arm ratio, resulting in lower performance overall.

A log disk is required; we are out of controllers to conveniently hang this drive onto, so we will pick the least of the evils and connect it to the end of one of the strings with 911MB disks. Query 1 is considerably more tolerant of performance anolomies.

The scanner/updater process brings new complexity. The magneto–optical jukeboxes must be configured on a SCSI bus. A 1/2" tape for data interchange must be configured on the built–in SCSI host adapter, along with the mandatory CD–ROM drive. Because of the physical distances involved, a separate SCSI host adapter must be configured for each jukebox (on the SCSI/Buffered Ethernet card). Contention for the SCSI bus is therefore not an issue. This consumes two of the three available Sbus slots.

The updater process must consume about 20MB serially from the disk, and writes it to optical storage. This means that in three minutes the updater generates about 2600 raw serial I/Os.

6. Obviously, the "one" kernel cpu is actually all four cpus executing in the kernel. This is the definition of symmetric multiprocessing on a shared memory multiprocessor.

7. In fact, higher–density disks are *usually* not the right answer; to determine if higher–density disks are suitable in a specific situation, consider both the amount of storage required and the rate of I/O operations against that same storage. Often it turns out that the correct disk size is *half* of the available disk drive! Some vendors are developing disk assemblies which have more than one disk arm per platter stack. This would be the correct solution if the application requires frequent access to data which is relatively small compared to the frequency of access.

- The data transfer system obtains blocks of 120MB of data from the IBM mainframe where it is preprocessed, then broken down into approximately 12,000 blocks of 10KB, which are written serially into a temporary table. This serial table is then processed into a variety of smaller tables, in groups of about 1000 blocks (about 1MB each). Including cpu time on the transfer system and network transfer time, the process takes about an hour. The process is repeated about once every other hour.

From the large capacities, large volumes of data, and relatively large number of users, it is clear that this server will be a SPARCserver 690MP. An analysis of the above data yields substantial configuration information.

Query 1 must perform 25,600 reads to scan the 200MB; because they are issued serially by Oracle against raw disk partitions, they will be 8KB raw reads. The best disk to configure for this task is the 911MB drive, because of its 6MB/sec serial transfer rate. One such disk can deliver about 530 8KB serial raw reads: a single drive would take 49 seconds to read the 400MB. To perform the reads in less than about 20 seconds (the requests are issued to the server every 50 seconds or so), we should configure three drives to be active simultaneously. The configuration aims to complete the reads in 20 seconds or less to minimize the possibility of multiple copies of the query arriving in the server's work queue simultaneously, causing the serial access pattern to become substantially random. The 12GB table requires 14 drives, and if we need three active drives we must configure the 14 drives on three controllers. Since the data is being read serially, the maximum is one active disk per controller. Oracle's internals will be responsible for spreading the data evenly across the 14 drives. Three controllers can deliver the data in about 30 seconds. This means that about 850 disk reads are being performed every second; at a cost of 0.53 ms per operation (0.67 ms normalized to a 28.5 MIP cpu), this represents 450 ms per second of cpu time in the kernel.

Each copy of Query 1 returns 20–50 MB – typically about 30MB – of data to the client for further processing. The volume of data being transferred mean that FDDI is the only realistic network media. We will configure a Class–A FDDI board in the VME card cage. FDDI's maximum transmission unit is 4500 bytes, meaning that the typical 30MB transfer will arrive in about 6990 packets in a bit over 30 seconds (233 packets/second). At a cost of 0.2 ms per packet, this rate represents 150 ms per second of kernel time. Note that this time, combined with the disk service time, we've consumed 600 ms of kernel time per second servicing Query 1. Earlier measurements indicate that Query 1 also consumes about 70% of one cpu in user space.

Query 2 has a much different performance characteristic. It must retrieve 20MB in 16KB blocks, chosen randomly from a large table. This is 1280 random reads. Because they are purely random accesses, the best disk to configure is the 1.3GB drive. Seven of these disks are required to store the 8GB table; if we configure them on two controllers, we can expect about 450 random access I/Os per second (74 ops per drive times six active drives). At this rate, the disk read portion of Query 2 can be satisfied in a little under three seconds. The cpu cost for the disk operations is about 450 * 0.73 = 330 ms per second of system time.

To get the data back to the client requires 1639 FDDI packets or 7.2 MB/sec – far in excess of the realistic speed of the network, which is probably about 1 MB/sec with this generation

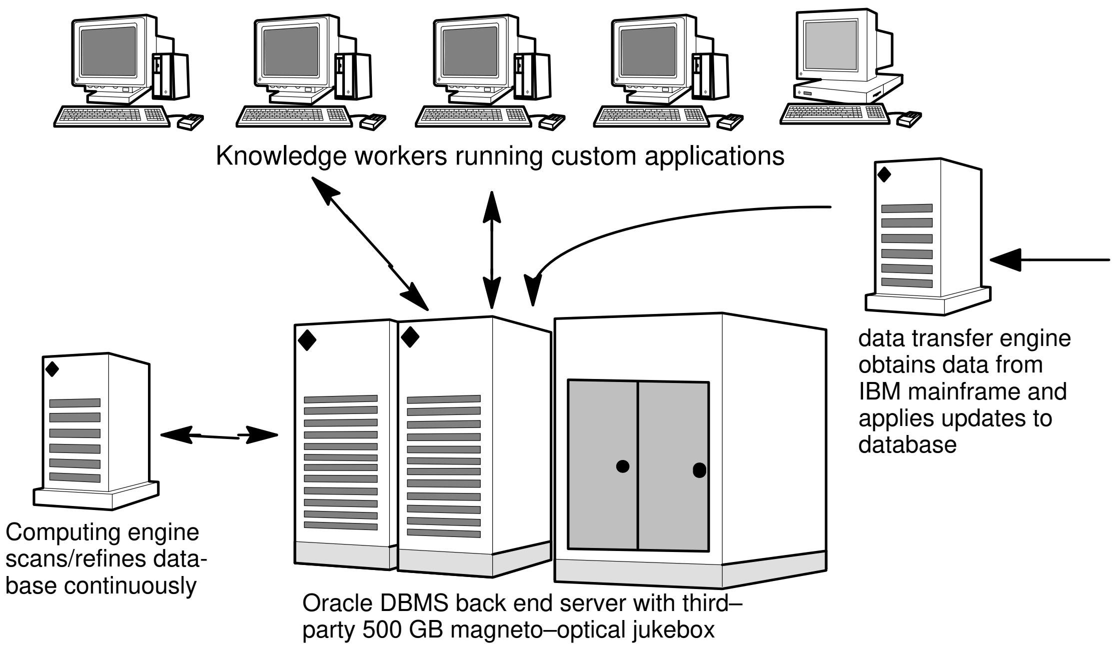


Figure 3. The logical flow of the GIS application in Case Study 4.

After monitoring the various types of queries and talking with the development staff, the following things are known:

- The most frequently used query (Query 1) involves a serial search down a part of a table. Each instance of the query is expected to scan approximately 200MB of data, returning 20–50MB for further processing. The active part of the table is approximately 12GB (the remainder is staged separately from optical disk upon demand). This query is executed about four times per hour per client workstation. Because the query is executed in parallel with other user application tasks, there is no especially tight performance specification. The other tasks generally consume 1–2 minutes, so the query should normally be completed in two minutes or less.
- The most performance–sensitive query (Query 2) involves selecting a variety of data points by key from another large (8GB) table. Each instance of the query only accesses 20MB, but it is randomly spread throughout the 8GB table. The data is retrieved in 16 KB blocks. This query is executed about once every five minutes on each client workstation. The user community has specified that this query must be completed within an average of 30 seconds.
- The process which runs the scanner/updater retrieves 20 MB of data serially, generally from optical disk, then typically performs about 300 updates of about 1KB each – to disk. Additionally, it consumes about 20 MB of data from one of the main tables; this data is obtained randomly. The entire cycle is repeated about once every 10 minutes, although it is expected to increase in frequency to about once every three minutes.

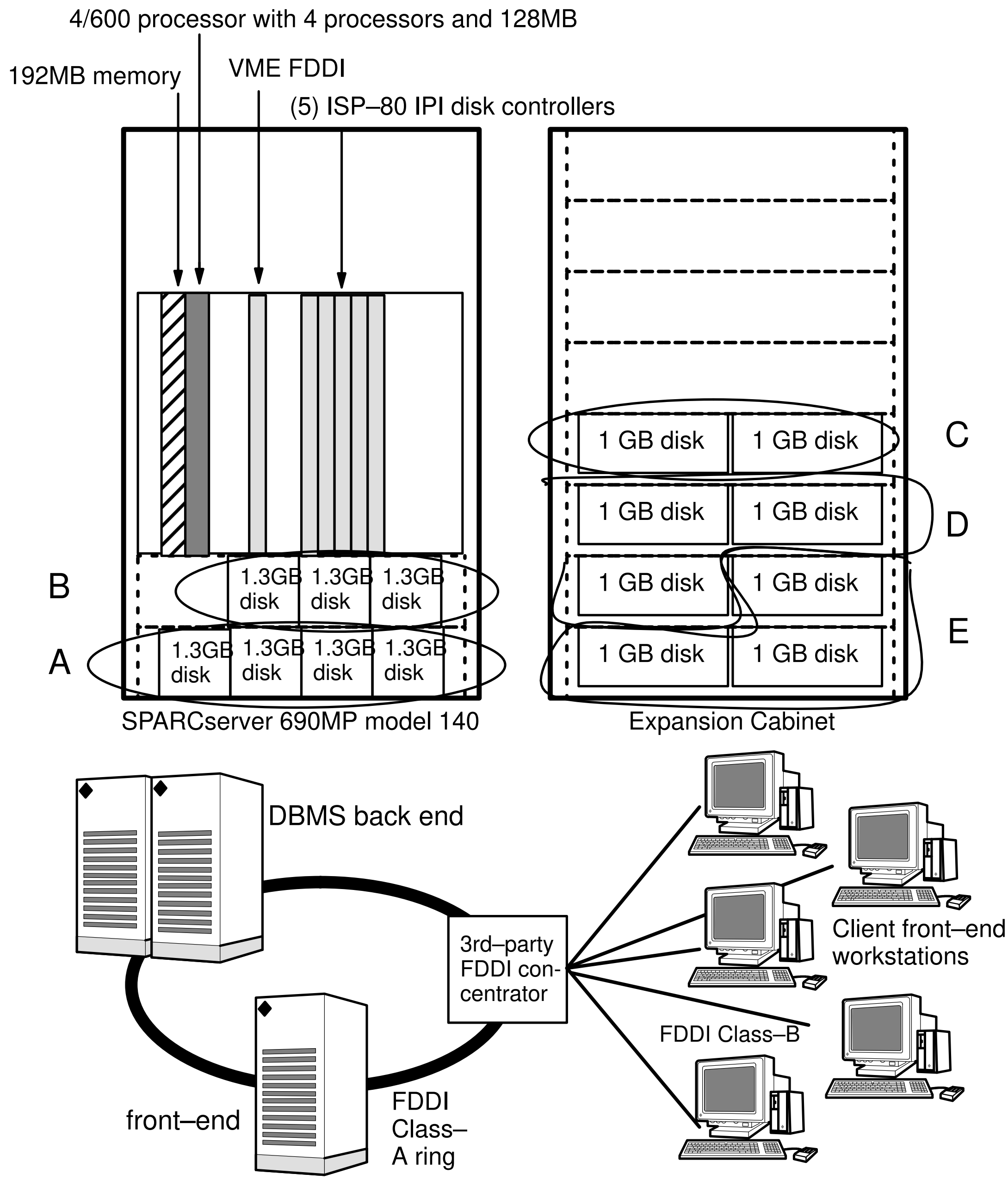


Figure 2. Final configuration for example 3., a server intended for servicing Freedom–of–Information–Act database requests. The large letters indicate the grouping of disks on the IPI–2 controllers.

it is reasonably easy to predict what the query mix will be (at least, it can be predicted as well as the use of the application itself). Unlike the previous case study, all of the queries to the database are program–generated, so the software engineers can provide a complete list of which queries must be executed. If the customer doesn’t already have SunNet Manager, this is a good time to install it. (See Appendix C., *Measuring an Existing System.*)

possible. This means four processors, and a maximum of three Sbus slots. Fortunately, we don't need many Sbus slots.

For software, we want to have SunDBE since we're using Sybase. We know we will make use of it since at a minimum we already know that we're working with very large virtual memory spaces. The server has no use for eNFS (there is no NFS service), PrestoServe (there is little write activity), Network CoProcessor (we will be using FDDI for the high–volume networks), DiskSuite (Sybase provides this internally), or BackupCopilot (Sybase also provides backup facilities), so we're done. The front–end machine needs some attention, but we'll come back to that in the example section under multiuser systems. The final configuration is shown in Figure 2.

Note that we were missing a lot of information in this example, even though we had an existing system running the actual target application, and hence we had to guess – a lot. In this case, we are making such a quantum change in both the size of the client community and in the processing power being made available that we have little opportunity to accurately estimate what will happen after the dust settles from the installation. In addition, we were handicapped by the widely varying nature of the queries being processed. This is a characteristic of many archive–style applications, as well as a class of applications known as executive decision support systems. This latter category is particularly notorious for generating queries which consume vast amounts of system resources, yet are singularly difficult to predict in advance.

1.2.9.2. Case Study 4. Database Service for a Large GIS application

Next let's configure a server for a group in the exploration department of a Fortune 500–sized company. This group will support field teams by cataloging geological, mineral, usage, legal and pollution information in a very large GIS database. The underlying database is to be Oracle. The users will never interact directly with Oracle, but rather with a group of custom applications developed in–house. The database server will be responsible for a database of some 470 GB, of which some 40GB or so is expected to reside on disk; the remainder will be stored on a pair of optical jukeboxes, which are SCSI devices. The group has eighteen knowledge workers who are constantly querying and updating the database from a mix of Sun and SGI workstations. In addition, some of the data in the database is under continuous refinement: an expert system running on another Sun continuously scans small (10–20MB) segments of the data looking for exploitable areas. In addition to obtaining data from the database, the expert system also removes instrument noise from the data, so it typically makes updates. Finally, raw geologic data is continuously added and updated. The source is yet another Sun server, which receives the data from an IBM mainframe via a dedicated 16Mb token ring network. Most of the applications are now complete, but this will be the first group to go to live data. The logical flow of the application network is shown in Figure 3.

In this case, we have the opportunity to measure canned editions of each query. The development group is necessarily running much of the application suite now, so it should be easy to quarantine one of the development servers for a short period of time for the purpose of measuring the various loads imposed on the servers by the individual application programs. Also, because the system is being designed with a specific purpose and user community in mind,

any of the 6MB/sec drives into the new system; however, if we were configuring from scratch, probably two of the 911MB disks would have been a good idea.

Memory configuration is easy; we know we need 300MB to accommodate those huge reports, and we know that we don't want to spend on much more than that, so 128MB on the cpu complex and one memory expansion board with three banks of 64MB memory will yield a total of 320MB.

Network connections are next. It's probably not a very wise idea to attempt to connect the database server system to the DBMS client system via ethernet, even the network is dedicated to the purpose. The existing ethernet is full with 15 clients now, and even if the dial–in users consume only half as much network bandwidth, 75 of them will cause even a dedicated ethernet to melt. A 16Mb token ring might be an option, except that it can't be configured into the 4/390, and it doesn't offer the advantages of an FDDI network. In particular, it uses a maximum transfer unit of 2KB, meaning that FDDI will save nearly 70% of the protocol processing overhead: those long strings of 1500–byte packets will come in a third as many 4500–byte protocol bundles. The token ring, while offering somewhat higher media throughput, doesn't substantially reduce the protocol overhead. Adding the Class–A FDDI/DX board to the 4/390 and to the 4/690MP is the most straightforward solution, and provides plenty of network bandwidth for those 75 client connections. In addition, because there's plenty of bandwidth, we can consider rewiring the workstation clients onto a Class–B network, although we will need an FDDI concentrator to connect the Class–B physical net with the Class–A network we have already set up. This is some valuable insurance against the day when the existing ethernet can't handle any more data – which is likely coming soon, if the collision rate is any indication (it usually is, especially with as few as 15 clients). The 690's built–in ethernet port can be used to gateway the entire operation to the existing ethernet backbone.

For cpus, we don't really have a good idea of what to expect – both the cpu and the disk subsystems on the 4/390 are saturated, and there is no easy way of telling which is the bottleneck. We know that we are already handling 8*40 IOPs to the eight 1GB disks, which is translates to about 288 ms/sec (see table 7.), and we expect to be able to saturate the seven new disks, too, at 7*65 ops, consuming another 410 ms/second. This totals around 698 ms/second, normalized to about 550 ms/second (the costs in table 7. are given in terms of a SPARCserver 490). The 90 clients seem to be able to generate 800 ÷ 3 (workstation clients, presuming FDDI instead of ethernet) + 75*25 ÷ 3 (estimating the dial–in clients to be half the load of the workstations) = 267+625 = 892 packets per second over the FDDI network, which translates to 178 ms per second (at 0.2 ms/packet). We can see that we will be consuming about 75% of a cpu in the kernel for the I/O operations we know in advance we will be handling. We can see from this quick calculation that there will be enough cpu time under the kernel lock to handle the new load, and that there is a 25% margin for handling other strictly kernel functions. Clearly we will have much more cpu available than we did before, but we will be supporting 75 dial–ins instead of 30. Since we know that one cpu will be mostly consumed with processing in the kernel, and the vast majority of the processing time appears to be *above* (outside) the kernel lock, it's probably reasonable to configure as many processors as

2%) even though there are only fifteen remote workstation users and the other thirty or so users are all running their front–ends on the existing server. If alarm bells are going off in your head over the volume of data we might be trying to transmit in the new configuration, you're getting the idea. Some investigation with `etherfind(1)` and a Network General Sniffer shows that the vast majority of the packets are long strings of 1500 byte data packets being sent from the Sybase server to the clients. We have no particularly easy way to measure the clients running locally on the server, but the customer indicates that the dial–in users generally are less demanding that the phone–in requestors – if the volume of printing is any indication.

As much as we've learned from asking questions and poking around on the current system, we don't have a lot of the information we'd like to have.

Situation	Configuration impact
CPU required for typical query	N/A – number of cpus unknown
Disk accesses for typical query	N/A – number of disks is unknown
Sybase server working approaches 300MB during certain queries	Must have at least 300MB main memory on new server
Sybase client/server interaction has existing ethernet very full	Should configure FDDI between front–end and back–end machines
Disk volume about 15 GB	SCSI would need lots of host adapters – probably too many
Customer has existing 8x1GB disks	IPI disk configuration
Very little write/update activity	PrestoServe won't help much
Significant serial access in task that limits its performance today	Should try to configure 6MB/sec IPI disks for fast serial access

So what does this all add up to? For starters, the initial disk volume of 15GB essentially eliminates SS630 and SS670 configurations due to their limitation on Sbus slots – we'll need at least 12 disks, and since most of them are accessed randomly, we would want six SCSI host adapters. The clincher is that we have eight 1GB IPI drives and a mass storage rack to be transferred from the 4/390 (which will retain three drives). This forces us into an SS690 configuration. We still need at least 8GB of disk, so we will want seven of the 1.3GB IPI disks. Because they're randomly accessed, we would like to assign the disks three to a controller – we have fifteen drives, so that means five controllers (the maximum). It also means that we will specify the seven 1.3GB drives in two trays, with three and four disks, respectively (other configurations might have warranted specifying just two or three disks in each tray, since each tray can be connected to at most one controller). Since the vast majority of the disk space is consumed in the database, and the only other data on the system will be the Sybase and UNIX binaries, we won't bother trying to spread the swap space, as we may well end up extending the seek times by doing so (swap would have to be a separate partition from the database accesses). We will use one of the old 1GB drives as the log drive. Because we have so many of the original drives to carry forward, we won't be able to justify configuring

workstations of various kinds (Suns and VAXstation 3100's) which provide reporting services for users without dial–in capabilities. Users interact directly with the Sybase DataWorkbench application, since queries are typically proposed on the fly. The Data WorkBench and the APT–Report writer are the only applications. One of the main problems with the existing system is that while it handles the typical load reasonably well compared to the VAX it replaced, the system is severely overloaded at peak demand periods, and whenever certain large reports are run.

Since we have an existing system, the first thing to do is to measure it to get an idea of what problems really need to be solved. In particular, we want to see why those large reports are such a problem. After some investigation, we determine that the large reports use an SQL query which performs a 3–way join on very large tables, resulting in a 1500MB temporary table, which then must then be sorted (this is the reason for the 2GB of temporary free space!). The impact on the existing system is catastrophic: the Sybase server's working set swells to nearly 300MB, and with only 56MB of main memory, running the report consumes all of the available memory *and* all of the I/O bandwidth of the two IPI controllers. Now we know that we'll need lots of memory and swap space. One thing to do here is to investigate the seek behavior of the disks while doing this sort of operation. In this case we can't look at the disk activity while it's actually running one of the real queries because the page thrashing invalidates any information we might get, but running query against a (much) smaller dataset that better fits in memory reveals (via `iostat(8)`) that the sort operation uses relatively few, very long serial disk operations. This tells us that if we put the that temporary table onto 911MB disks, we'll get better sorting performance than if we use the 1.3GB disks we'll be using otherwise for the new server.

Returning to measuring the more normal queries, we find that the system does very few writes and many, many reads – as expected for this kind of application. Use of `vmstat(8)` shows that the cpu is nearly always saturated, and that the vast majority (around 90%) of the time is spent executing user code, outside the kernel. Furthermore, the disks all seem to be close to saturation (averaging 40 I/Os per second on the older 1GB IPI drives) – the customer's system administrator managed to balance the whole system fairly well.

A brief look at the client processes on the server indicates that their typical working set varies between 300KB and 6MB – mostly clustered around 3MB, apparently due to the large volumes of data being handled. The existing system is swapping fairly frequently, since it's out of memory, and from this, it's easy to see that even in its new role as a front–end machine, it's going to be out of memory with 75 users if they're going to average 3MB each.

We also find that there doesn't seem to be a "hot" area of the database – things seem to be randomly distributed if those troublesome queries aren't being run. There also doesn't seem to be a "typical" query, so it is virtually impossible to accurately gauge when either the cpu or the disk subsystems will become the bottlenecks. Unfortunately, this is characteristic of DBMS applications which are primarily ad–hoc queries, making them some of the hardest to accurately configure.

Another fact that SunNet Manager reveals is that the network is sustaining about 400 packets per second, mostly 1500 bytes in size – which is a remarkable amount of of network traffic for this number of clients – and collisions seem to be hovering near the danger zone (about

network I/O it can prevent. If in doubt, guess high on the memory. Thirty–two megabytes is probably the smallest reasonable configuration today except for prototype systems; even prototypes should have at least 16MB.

1.2.8. Main Processors

All those milliseconds per operation have been adding up. It's time to multiply them together to arrive at an estimate of what kind of and how many cpus you'll need. Include about a fifth of a millisecond for each network packet, and the cpu expense of a disk I/O from Table 7. If MetaDisk is being configured, take into account the additional overhead shown in Table 11. Remember to normalize from the speed of the cited processor to the speed of your target processor: not all cpus are equal. The 5ms you measured on a 4/280 isn't the same as the 5ms you'll get from a SS690. Hopefully, it has been possible to configure sufficient peripheral resources that the cpu is the bottleneck (the peripherals are generally more configurable than cpus).

If you come up with more than 1000ms per second, you have exceeded the capacity of your target processor. The only way around this is to go to a multiprocessor with a multithreaded kernel – but don't forget that Solaris 2.0 will likely have different expenses for each operation. The reason we need a multithreaded kernel is that all of the cpu time we have accounted for up to this point is system time. Even on the SPARCserver 600MP we only have one cpu worth of system time.

After system time is calculated, add up the estimates for user time. This is generally much fuzzier unless you have an existing system to measure. Add the number of cpus required to support the user time. Take into account the fact that the current compilers generate single–threaded code, necessitating multiple processes if more than 1000ms per second of user cpu time are required.

1.2.9. Some Examples

Let's work through a couple of examples, since all of this architectural stuff tends to be very dry and remote.

1.2.9.1. Case Study 3. Database Service for an Ad–hoc Query System

First, let's configure a database server for a large department of a governmental agency, whose function is to provide on–demand information in response to Freedom–of–Information–Act requests.

The customer provides the following parameters: the database presently contains eight gigabytes of data, with an additional two gigabytes of indexing information and another two gigabytes of free space required within the database for various temporary tables and indexes. The data is expected to grow at about a gigabyte a year for the foreseeable future. The database is Sybase, and the customer has an existing 4/390 which currently runs the database – slowly, as it turns out. After the arrival of the new server, this existing machine is expected to run the front–end processes for 40–75 dial–in users. Additionally, the customer has fifteen

5. Source: Sybase, Oracle and Informix, respectively.

`select(2)` system call. Some of the other enhancements, particularly those such as enhanced handling of very large virtual memory spaces, do not require programmatic changes, and any application can transparently derive benefit from them. If you are using another DBMS, SunDBE may well provide substantial performance benefits.

On the other hand, the use of Online:DiskSuite's MetaDisk driver is *not* recommended if your database manager has the facilities to provide disk mirroring and/or striping. The database managers normally assume responsibility for optimizing disk access (they usually use the raw disk interface), and adding another software layer may well confuse the optimization routines in the DBMS's disk management layer. Of course, if the server has responsibilities other than DBMS service, Online:DiskSuite may well be appropriate.

The Backup Copilot is not relevant to backing up the database (at least not without additional use of the DBMS's own backup/dump facility). The exception to this would be a database which is stored in a file system rather than in a raw partition. Of course, most DBMS servers will also provide some (possibly a lot of) NFS service as well, so the Copilot may well be applicable outside the database itself. In this case, *be sure to pay attention to SCSI host adapter loading if disks are configured on the SCSI bus* (see section 2.4.2.).

1.2.7. Main Memory Configuration

The goal in configuring memory is to provide enough physical memory that the system doesn't swap, and preferably doesn't page except when the server has to retrieve new data from tables[4]. If the system must swap (different from "must use swap space"), performance will be terrible, regardless of which DBMS you are running. Even non–I/O related paging will result in significant degradation. If you can't measure a live system, the best thing to do is to reserve 8MB for the operating system, 4MB for the DBMS, and then a given amount per client connection as in the following table:

DBMS product	Basic memory requirement	Per–client memory requirement	Additional memory requirement
Sybase	data cache: < 10MB + procedure cache: 5–15K per client	60 KB per client	N/A
Oracle	4MB	1–2 MB per client	30–50% of (basic + total per–client)
Informix Online	570 KB + 1–80 MB shared memory	70 KB per client	N/A
Ingres			
Interbase			

Table 3. Per–client memory requirements of popular database management systems[5]. These figures were quoted by the respective vendors. They represent typical applications and are not meant to be inflexible laws.

Memory is the cheapest single resource that's reasonably configurable. It also usually provides the biggest bang–for–the–buck because it is so much faster than the physical disk or

4. Most disk I/O activity is now done via the paging mechanism.

loading of a file system.) At the end, add another disk to be used for the log file. This drive might be shared with other applications, but not with the database proper. The log drive is insurance against failures in the main database, so it is unwise to configure it on the same physical spindles. Size this disk according to the size of the log file generated by the application.

Next configure the host adapters or string controllers. For SCSI, try to configure two randomly–accessed disks or four sequentially–accessed disks per host adapter. If this is not possible, distribute as evenly as possible within the constraints. If the disks are IPI, try to configure at most three "hot" disks per controller.

If you're in doubt, or if there is uncertainty in the operation–cost calculations, it's generally safer to overestimate than to underestimate. Insufficient disk or channel I/O bandwidth can result in very poor response time.

1.2.5. Networks and Network Interfaces

The process of determining number and type of networks and interfaces is similar to that of determining disks and controllers. A reasonable understanding of how the underlying DBMS breaks down SQL queries for processing, as well as the number, frequency and data transmission characteristics of each transaction is *mandatory!* Perfect forecasting isn't mandatory, since network traffic is even more subject to randomization effects than the disks, but a ballpark estimate is. Unfortunately, there just isn't any other way to accurately forecast the load on a network.

Multiply out the number of clients, the average number of transactions, and data volumes to arrive at an estimate of network bandwidth requirements. With this information we can configure the right number of networks by consulting Table 10. Like configuring the disks, don't presume that the load will precisely balance between configured networks. If in doubt, try to configure an extra network.

Experiments have shown that the traversing the protocol stacks costs about 0.2 ms of cpu time per packet on a SPARCstation–2 or SPARCserver 600MP. Multiplying by the network load gives an indication of how much cpu time will be required to sustain the network load (remember that protocol stack processing constitutes the vast majority of the cpu time involved in network communications). If the network interface includes protocol processing (such as the Network CoProcessor, or to a lesser degree, the TRI/S token ring interface) this factor can be essentially ignored.

1.2.6. SunDBE, Online:DiskSuite and Other Software

If your database management system is Sybase, Informix, Oracle, or Ingres, we strongly recommend using SunDBE; it is inexpensive and provides a means of providing DBMS–specific tuning inside the Solaris kernel. Other DBMS often do not benefit from DBE since some of the optimizations make (slight) changes to the programming interface which must be accommodated by the DBMS software. One example of an optimization that requires application modification is the use of large numbers of file descriptors (more than 255); extending this requires violation of some UNIX interface definitions, such as calling parameters to the

Sometimes it is possible to configure sufficient main memory to basically avoid any paging and swapping activity. This is always desirable, but often impractical. There is no practical way to forecast the amount of VM–related paging in a database management system without some specific knowledge about the actual behavior of the target application system. If you have some knowledge about the paging activity expected, it is obviously important to factor this into the I/O capacity calculations. As with all other aspects of the planning effort, knowledge about existing systems is the best guide.

As with data disks, it is worthwhile to spread the swap partitions across several drives. However, sharing a disk between a swap partition and a frequently–accessed database partition is likely to result in many long seeks as the arm tracks between the partitions. A good solution to this problem is to configure just the basic raw swap partition on the boot drive and then spread additional swap space onto additional drives *in the file systems which supply the application binaries* (use `swapon`(8) and `mkfile`(8)). Swapping through the file system has no significant penalty as opposed to swapping on a dedicated swap partition, and locating it in the same file system as the application binaries tends to keep the disk arm in the same vicinity[3], holding down seek times. Refer to Section 2.4.4., *Disk Drives* for a more complete discussion.

When configuring multiple swap areas, try to divide the swap space across the disks in approximate proportion to the random I/O capability of the drives; the virtual memory system seems to access swap space proportional to the size of the partitions. If you have two disks, one with 10MB and the other with 30MB, the second disk will get about 75% of the swap accesses.

1.2.4. Configuring Disks and Controllers

Once a cost can be associated with each query and application, you can compute the number of I/Os you'll probably need to support. Divide this number by the number of random I/Os available from the appropriate disk drives, as listed in Table 8. Most I/Os in a database management system will be random access, unless the applications are primarily batch–oriented. As with NFS servers, it is often worthwhile considering more small disks, as opposed to fewer large disks. The 424MB disk delivers 50 random I/Os per second (at the recommended 65% loading, see 2.4., Disk Drives), or 8.5MB per random I/O, while the 1.3GB drives offer 21.6MB per I/O. For small–to–medium size configurations – say, with database sizes of 800MB to about 2.5GB – multiple 424MB drives can offer substantial additional performance compared to the 1.3GB drives if sufficient SCSI host adapters can be configured (see 2.4.2., *SCSI Host Adapter Loading*).

Don't assume that data access will spread completely evenly across drives. If the average number of I/Os per disk is close to the 65% margin, it's probably wise to add at least one more disk. That 65% margin listed in Table 8. is not a margin for error; rather it is headroom to enable the drive to sustain maximum performance. (This is similar to the 90% maximum

3. If you configure swap space in this way, try to allocate the swap space in the file system *first*, so the file system can take advantage of an empty disk to allocate the blocks of the swap file in contiguous data blocks. If a swap file is added into a well–used, nearly–full file system, its disk blocks could be scattered all over the file system. This is still better than putting it in a separate partition. If you are tuning an existing server, dump the file system, newfs it, build the swap file, and then restore the user files.

1.2.2. Estimate the Operation Mix

In addition to knowing which database manager is to be used, a reasonable understanding of the typical operation mix is absolutely mandatory. For example, a database which implements an executive decision support system is likely to produce mostly select operations, combined with sorts and probably joins. At the other extreme, a database used to implement a satellite download link is more likely to do many inserts and relatively few reads and essentially no updates. The organization of the database tables is also important. From these pieces of information it is usually possible to arrive a rough estimate of the number of I/O's that will be required. From this we can get estimate the number and type of disks and controllers, and we can get a vague idea of how much memory we will need to configure.

Usually it is best to break down the application mix into its components, and then break each application down into typical or "popular" SQL queries (query here actually means any database access). For each query, calculate or estimate the number of I/O's that would be required, presuming no cacheing. We presume no cacheing in this calculation because there are quite a few disk accesses that are virtually impossible to predict without knowing exactly how the DBMS lays out its internal data. This disk activity includes index table references, linked–list updates and various other non–data related control information. Experience has been that, generally speaking, ignoring cacheing effects about covers the internal DBMS I/O costs, at least for Sybase and Informix.

Don't overlook an existing system if one is available to observe. It is often very feasible to observe a limited run of an existing system and obtain sufficient information to accurately forecast a configuration. Finally, if the target system's performance is mission–critical, it may be worthwhile to construct a model database to measure various system load characteristics. This may seem like a lot of work – it is – but remember that botching a server's configuration can have severe effect on a very large community of users.

If an existing system is available, the amount of cpu time expended on each type of transaction can be quantified. There really isn't much point in trying to guess these quantities. If it isn't available (and it usually isn't), you'll end up having simply to guess. Fortunately, with multiprocessing, it is easier to recover from the effects of a missed guess.

1.2.3. Virtual Memory Considerations

An NFS server has essentially no swap space requirement, because all of the data it manipulates is data that resides on its disk space. All of the active data is simply mapped into the server's virtual address space and transferred directly from VM to the network interface. Very little data is copied for manipulation, and the NFS daemons retain essentially no overhead data (less than a couple of megabytes). Even large NFS servers rarely require even 64MB of swap space. Database management systems, on the other hand, generally have quite substantial VM requirements. Many database operations, such as a join or sort, require the generation of large amounts of temporary data which do not reside in the database at all. These must be placed in virtual memory. It is common for large DBMS servers to consume 100–300MB of swap, and this trend is increasing with the complexity of the applications beginning to come online.

the front ends to run on the back–end machine, such a server is really a multiuser– or mixed–use system, which we will take up last.

The rules for configuring a database server are less precise than configuring an NFS server, primarily because there is one common implementation of NFS, but at least half a dozen common database management systems, each with unique performance characteristics. Although they perform the same functions, and often can be treated by users as functionally identical (*e.g.,* by SunSimplify users), the database management systems impose radically different demands upon their supporting systems.

1.2.1. Which DBMS?

There are many more variables involved in configuring a server to support databases than to support NFS. Consequently, understanding the demanded workload is even more important when configuring a database server. The first variable to be faced is the DBMS software itself: while there is essentially only one flavor of NFS server, there are at least a half–dozen popular database packages. Each has strengths and weaknesses, some preferring or requiring more or less memory, disk bandwidth, or cpu power; they also differ in how and where they break an application into client and server tasks, and the effectiveness of their associated application generation and optimization tools. It is basically an impossible task to accurately configure a database server without knowing which database management system is to be used.

The database managers often split atomic SQL operations differently across the client/server boundary. Typically, the so–called object–oriented DBMSs (such as Interbase) simply use the server as a large retrieval/storage engine. In these systems, a high–level query from the client application is broken into storage/retrieval/search operations performed directly on the server system, and into application–specific dataset modifications, such as joins and sorts, which are usually transferred across the network and performed in the client. Most of the more traditional “client/server” databases – including Sybase, Oracle, Informix and Ingres – perform most of these functions primarily on the server, sending only the essentially completed query results across the network to the client. Both of these approaches have their merits and associated performance implications. Basically, systems which divide the work more equally often require a smaller server and more network bandwidth than systems which are server–intensive. For example, the query

```
select * from personnel
      sort by descending salary, ascending last-name
```

always causes retrievals from the personnel table. Under Sybase, the sort is done in the server, and the sorted data is passed down the network to the client. With Interbase, the raw data is retrieved on the server and immediately transferred to the client – which then performs the sort. If there are 10,000 records in the personnel table, the sorter must have sufficient virtual memory resources (swap space) to accomplish the sort. Most servers have this, but often clients don’t. This is particularly relevant if you have diskless clients! In this example, the amount of data transferred across the network is constant, but other operations, notably joins, can result in drastic differences in network load.

overall performance is critically governed by the server's performance. Because the Network CoProcessor can process a maximum of 290 NFSops/sec, it is clear that we need at least two networks, which rules out the SS630MP configuration (it doesn't have enough VME slots in which to configure the NCPs). If we configure four NCPs, each is responsible for 985 $\div 4$ = 246 ops. This slightly exceeds 232 ops/sec, which is 80% of the 290 maximum throughput per NCP. This is a logical arrangement from a physical standpoint anyway, since there are four classrooms.

As it turns out, the courseware is actually quite small. The typical module is about 15 minutes long, or about 7.5MB. There are six or seven modules per day, and the classes run two days, so the entire class fits in about 100MB. Only half of that would be required on any given day, and the data files for the simulated case studies amount to about 30MB. Even if future video is added, a single 1.3GB disk is clearly sufficient to store all of the data.

Since we are seeking the fastest possible response time, a PrestoServe should be configured to minimize the write latency of any conceivable virtual memory pageout. Since the SS670MP uses SCSI disks, we will configure an Sbus PrestoServe. Client paging should be reduced to the minimum by configuring additional memory in the client SPARCengines.

Probably the biggest impact on the overall network in this situation is to configure sufficient memory to avoid disk activity in the server. This is possible in this particular application because of the minimal size of the data. A full day's data is about 50MB. Combined with the client swap space (not all of which will be used extensively) and the server's own minimal requirements, a 128MB configuration is nearly certain to permit virtually the entire application to be cached in memory[2]. Minimizing the disk service activity also minimizes the amount of work the main cpu must do to service the NFS requests. This provides the necessary margin of safety until Solaris 2.0 arrives.

There is no point in configuring additional cpus at this time, since the server is dedicated purely to NFS service. With Solaris 1.0.1 and only one cpu in the kernel, this system will be very close to cpu saturation. When Solaris 2.0 becomes available, it will provide a healthy margin for expansion, since it will permit both cpus to operate in the NFS system call code.

With only one disk and very static data on that disk, Online:DiskSuite and Online:Backup-Copilot are unlikely to provide much benefit (virtually the only data on the disks that will change is that created by the analyzer during the labs, and this removed after each class).

1.2. Configuration of Database Servers

In this section we will be discuss a "pure" database back–end server. In particular, we will consider the specific configuration where a server is being used solely as a repository for a large amount of data managed by one of the client/server database management systems. Under these conditions we expect to have the database front ends running on remote systems, either on a single separate machine which runs many front ends, or on a cluster of workstations, each of which runs a front end. While it is certainly possible and useful to configure

2. The most conservative engineer might even go so far as to write a small program that mapped the courseware pages into virtual memory, then wire down the physical pages. This is quite extreme and unlikely to provide much benefit in this particular case.

data being "collected" is actually stored in read–only data files! A "chalk–board" mathematical analysis seems to be the most appropriate means of estimating the load.

The largest load on a server is likely to be the loading of the images in the self–paced reading. Each frame is on the screen for an average of 30 seconds; its associated text and questions take that long to read and answer. Each image is 512x480, and 8–bit color, a total of 245KB per frame. The associated textual information averages about 10KB per frame, so we can allow for 1/4MB per frame. If we allow for 30 seconds of audio at 8000 16–bit samples per second (the current SPARCstation audio rate), this represent an additional 16KB per frame, for a total of 275KB per frame. If we have 80 clients engaged in the self–paced classes, we will see 275 KB per frame * 2 frames per minute * one minute per 60 seconds * 80 clients = 735 KB/second being transferred. Each NFS read operation will obtain 8KB, so this is 92 read ops per second.

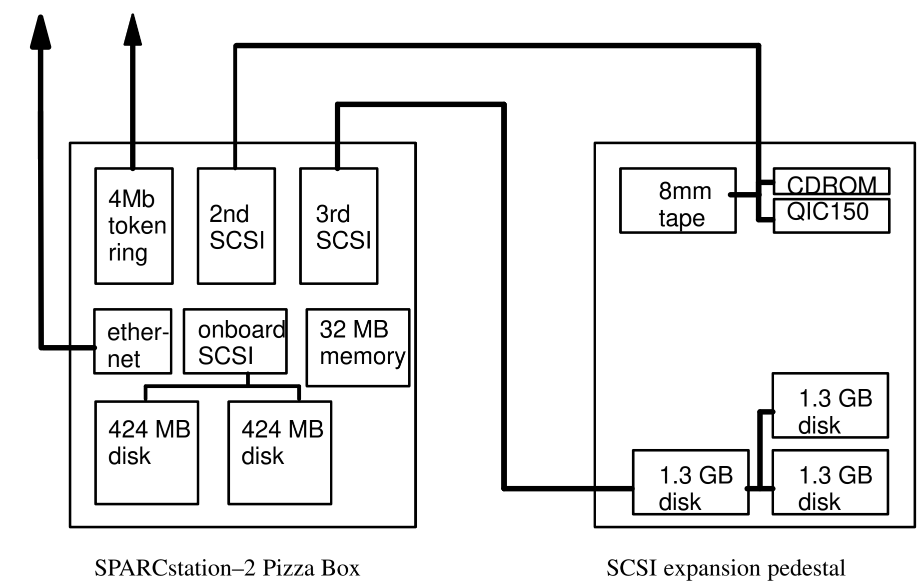
Assume for the moment that the NFS operation mix is the same as the commonly–cited Brown Mix. Reads are 13% of operations of the Brown mix, so this read load then implies an overall load of 92 $\div$ 0.13 = 707 NFSops/sec, which is quite a substantial load. Note that achieving 707 ops/sec on the Brown mix is much easier to achieve than the same load on the Legato mix, due to the large proportion of writes on in the latter mix.

The next load component is the diskless virtual memory traffic represented by the 80 client engines. From the Appendix on NFS Client Demand, we see that a diskless SPARCstation–SLC normally produces about 1 NFSop/sec when running typical office automation applications. The SPARCengine in the analyzers is of comparable speed, and because the analyzer is a captive, dedicated application, it seems reasonable to assume that the virtual memory demands will be about the same. Eighty stations thus means 80 ops/sec on the server. This traffic can be demanded at the same time as the operation of the courseware.

In the lab, the data acquisition process imposes no writes (the data collection is faked), followed by a data load of 8KB–1.25MB, depending upon case study. The analysis then proceeds at the rate of approximately 1 MB per minute. Typical case studies take about 5 minutes to perform and comprehend. The average case study loads about 300KB of data. Given this, the acquisition process imposes a load of 80 clients * 300 KB per case / 8 KB per NFSop = 3000 NFSops per five minutes, or about 10 NFSops per second. Measurements of the AnswerBook activity indicate that AnswerBook traffic is about the same intensity. Even doubling the traffic to 20 NFSops/sec, this is far less traffic than the self–paced instruction, and because it cannot occur at the same time as the courseware (the class is set up to provide each student with *either* courseware or lab time at any given moment), we will configure to meet the needs of the courseware.

Between the courseware and basic virtual memory traffic, we get 707 + 80 = 787 ops/sec. A fully–configured SPARCserver–2 might actually sustain this amount of traffic, but only at rates which are dangerously close to complete saturation. A much more comfortable fit would be a SPARCserver 600MP. Allowing for the full load to be achieved at 80% of system capacity, we need to configure for 985 ops/sec. (Remember that this is 985 ops/sec on the Brown mix, not the normally quoted Legato mix.)

The most responsive system configures Ethernets onto Network CoProcessor interfaces; this is especially desirable in this case because we have a large number of diskless clients whose



Software:

Solaris 1.0
Online:DiskSuite to mirror two of the 424MB disks
Online:BackupCopilot
Interstream eNFS to lower write latency and disk utilization

Figure 1. Sample mid–range NFS server configuration from Case Study 1. Note the distribution of the SCSI devices.

dents most of the time. In the lab, the analyzers use case–study data instead of actual samples; these data are to be obtained from a central NFS server. Additionally, the application's help facility is embedded in Sun AnswerBook–style technology. The self–paced reading material consists primarily of small (512x480) images created on a Mac with some associated textual information. For now, the presentation is silent, but the course instructor would like to add audio and limited–motion video in the near future. Finally, the embedded SPARCengines normally have their own disks, but in this environment it will prove easier to completely reinitialize each station after each class if the engines are booted diskless from a central server. The customer emphasizes that the simulated system must be at least as fast as the actual product, for obvious reasons.

The current classroom facilities permit eighty students to be in class at once (in four classrooms). Between current enrollment and projected sales of the analyzers, the classroom facilities are expected to be full within the next six months!

How much load is this? Good question. It's certainly a lot of traffic, and lots of customers tend to get spooked and head for some of the specialty vendors. But closer inspection is required. In this case, we have practically no existing information to work from – the present course consists of lecture material. The actual product can be measured, but this does not provide accurate data for the classroom since the input data is being synthesized – and the

software. Since some of the software vendors are still distributing on 1/4” tape, we will configure this as well (although it and the CD–ROM could just as well be put somewhere else in the network). All told, we have two 424MB disks, a CD, a 1/4” tape, an 8mm tape, and three 1.3GB disks. This is a lot of devices, and to keep the packaging simple and reasonably neat, it’s worthwhile considering the use of the SCSI expansion pedestal. Since we do not expect to be doing continuous backups during periods of heavy usage, it is reasonable to configure the CD and both tapes onto a single host adapter, along with the 1.3GB disks, all in the disk farm pedestal. The disk farm must be connected to the Sbus SCSI host adapter, since the two internal 424MB disks will be connected to the onboard adapter. The 424MB disks will be mirror components configured on the same host adapter, which is not an ideal configuration; however, since we are out of slots for host adapters, and we don’t want to drive up the utilization of the second adapter by stringing many disks off of it. In this case, we can use MetaDisk to solidify the configuration. We are providing primarily read–only service with the mirrored 424MB disks; by using the round–robin read option, we can balance the read load on the disks. (See Section 2.11.1., *Disk Mirroring* for details.) Disk writes have the potential to introduce a bottleneck at the host adapter, but writes should be rare in this application.

We need to configure enough memory for the system to have a reasonable chance of cacheing the important parts of the application binaries, which means allowing for them as well as a fairly substantial buffer reserve for use when servicing the serial ECAD requests. In general, since most requests will be short, memory size will not need to be excessive, and the 32MB base memory is likely to be sufficient. However, since the ECAD files are so large, it might be wise to configure a 32MB system and measure it, adding additional memory if warranted. On the other hand, another 32MB of memory is a very small fraction of the cost of such a server, so it might be well to configure it in to start.

This particular server has the potential to receive quite a few write requests, from both the software group and especially as the ECAD users save their large files. A PrestoServe might help lower write latency by a considerable margin; however, there are no Sbus slots with which to configure it. An alternative in this case might be the use of eNFS by Interstream. Although eNFS consumes more cpu in order to lower latency, and does not accelerate local synchronous writes, we have plenty of cpu power left over (the approximately 220 NFSops/sec will leave the cpu about 40% idle), and we do not expect to have much local traffic. The final configuration is shown in Figure 1.

1.1.9. Case Study 2. NFS service for a Corporate Training Center

This case study considers a very powerful NFS server for a corporate training center at a biotechnology company which sells a medical analysis machines; these contain an embedded SPARCengine–1. The training department gives offers classes on the tools to its resellers and end users. The attendance is currently stretching the capabilities of the limited education staff, so automated assistance is desired. Additionally, surveys of past students indicate that the highly technical nature of the material dictates much more one–on–one time with each student. To solve these problems, the curriculum designer has formulated a completely new course of study. The new course is a combination of self–paced reading, lab work, and some classroom–style lecture. This scheme leaves the instructor free to work with individual stu-

How much activity does this all represent? It looks like the twelve ECAD users will be demanding 120–150 ops/sec (4–5 users actually active at 30 ops/sec), and the programming staff is another 15–20 ops/sec, after allowing for some fudge factors and room for growth. Thus the maximum would be approximately 170 ops/sec for these users. Because they are workstation users, they are already wired for Ethernet. The 170 ops/sec load almost half of the capacity of a raw Lance interface, and over 60% of a high–performance Network CoProcessor interface, even before accounting for expansion room and the fact that normal 100% utilization means that peak periods of demand will suffer serious performance degradation. The PCs, on the other hand, aren't as demanding clients. Consulting Appendix B., we see that 80286 PCs loading email over a network seem to generate about 10 ops/sec, in short, relatively infrequent bursts. If we guess that the PC users will generate those bursts on the average about once every five minutes for about five seconds at a time, we get a sustained load of about 12–15 ops/sec for the email. Assuming (very conservatively) that the PC applications will generate the same load, we estimate a sustained load of 30 ops/sec or less for the PC clients, well within the capacity of even a 4Mb token ring (200 ops/sec).

It looks like we need to accommodate 170 ops/sec on one ethernet for the technical staff, 30 ops/sec on a token ring for the PCs, and 8 ops/sec a second ethernet for the administrative group – a total of 208 ops. Since this is well below the capacity of a SPARCstation–2 server (which can sustain 360 ops/sec on two networks, as noted in Table 1.), it looks like this will be the base platform, equipped with two ethernets and a token ring. We can safely configure more networks than shown in Table 1. because the SPARCstation–2's limitation is not network bandwidth, but rather cpu power.

Selecting disk configuration is next. This server will support two very different kinds of user groups. The mail spooling area, the software development group's files, and the application binaries will be frequently and randomly accessed. On the other hand, the ECAD files are large, infrequently accessed, and dominantly serial. This looks like a configuration which would benefit from two different kinds of disk configuration. The customer estimates that the mail directories will consume about 150MB, and the application binaries are another 250MB. Since the customer has specified that this data is to be mirrored, this means a pair of 424MB disks; we can use MetaDisk's round–robin read option to get a slight performance increase as well as providing for data reliability (geometric reads help sequential performance, round–robin helps random performance). Because both disks will be accessed randomly and simultaneously, these disks should be configured on two different host adapters. The second host adapter consumes the last Sbus slot (this particular situation will likely change in the near future when the SBE/S is supported on desktops).

The ECAD group estimates that they require slightly more than 3 GB of file storage. Because of the quantity of CAD data, this would have to be configured with 1.3GB disks. Because they are accessed primarily in serial fashion, the disks holding the ECAD files can be configured on a single host adapter if necessary. Finally, we need space for root, /usr and swap. In this configuration we can put root, /usr and swap on one of the 1.3GB drives.

The number of SCSI host adapters is driven by the large number of SCSI devices as well as the need distribute disks across multiple adapters for performance reasons. For backup, we obviously need an 8mm helical scan tape drive, and a CD–ROM is of course required to load

server 690MP's IPI disks, again at a slight (5–15%, depending on which disk operation) cost in cpu power.

The Metadriver's mirroring option provides data reliability and availability; in addition, it improves disk performance marginally *in certain circumstances*. Check table 12. and section 2.11.1. for information about your specific situation. Again, there is a slight cpu cost to this: about 20% of the cpu associated with managing the disk, or 2–8% depending on operation.

Servers which store large volumes of data must have a mechanism for providing backups. Systems with relatively small amounts of data (less than 2GB) can often make do by simply configuring a tape drive and setting up a cron(8) job to wake up in the dead of night and dump the whole system onto tape. Systems which must be used at night, or which have more than about 2GB of data need to make more comprehensive arrangements, such as those provided by Online:BackupCopilot. The configuration issue with BackupCopilot is SCSI host adapter loading. If there are plans to do backups during normal usage periods *and* if the disk devices are attached via SCSI, it is nearly imperative to configure the backup devices on a separate SCSI host adapter, if peak performance is to be maintained.

1.1.8. Case Study 1. A Mid–Sized NFS server for several dissimilar groups

Let's work through an example. Consider a server for a medium–sized department in a high technology firm. The server is expected to provide a mail spooling space and application binaries for about 75 PCs and 8 administrative workstations, as well as providing data storage for 10–12 engineering workstations running large ECAD applications; the data files for the ECAD applications are typically 40–100MB in size. In addition, a small programming support staff requires file service for their CASE tool binaries and source code; this group has three workstations. If it can be provided inexpensively, the customer would like to make the mail and binaries very reliable by mirroring them. Presently, the customer has six of the ECAD stations running the current ECAD software; they and the programming group are using a 3/280 server which is out of both disk and cpu power. All of the workstations have local swap disks.

The first thing to do is to decide what level of activity is being requested. Since the 3/280 is already installed and running, we can measure what kind of load existing users are demanding. In this case, nfsstat and SunNet Manager show that the six ECAD users typically demand about 30 ops/sec per client in long bursts as they load and save their files, but otherwise they generate very little activity. It seems to be an infrequent occurrence for more than two of the ECAD clients to be loading or saving at once, although the group manager warns that this might increase over the next six months as critical deadlines approach. The programming group, on the other hand, seems to demand about 5 ops/sec per client on a reasonably continuous basis. Most of the PCs do not yet have access to the network, and those that do have network cards do not have their email software installed yet, so they can't be measured. The PC users would like to continue to use their token ring wiring and boards, if possible. A bit of investigation on the installed server indicates that users tend to let their mail accumulate, though, such that mailboxes tend to pile up to a megabyte or more. The 8 other workstations have a separate network already (they really belong to another group), and they appear to be demanding about 1 op/sec per client.

as they're not all trying to get the IPI bus at the same time. See section 2.4.3. on the IPI string controller.) However, for disks which are primarily random – most medium–to–large scale servers are dominantly random access – one SCSI host adapter is appropriate for every *two* disks; one ISP–80 controller is appropriate for every three randomly accessed IPI drives. See Section 2.4.1., *Host Adapters and Controllers* for additional details.

1.1.5. Main Memory

Main memory should be configured around the access pattern. For optimal response time, there should be enough memory to cache most data which is frequently accessed. For example, a server which provides OpenWindows V2 to a number of clients doesn't need to have enough memory to cache all of OpenWindows, just enough to cache the frequently–accessed parts, which are the typical working set of the xnews server itself, most of `libX11.so`, most of `libxview.so`, and a few miscellaneous initialization files – all told, about 7MB. This can be hard to determine! If in doubt, try the minimum configuration and plan to add. In addition, we need to allow 8–12MB of working space for Solaris. NFS servers rarely require more than 64MB unless the individual files being handled are *very* large (> 100MB).

1.1.6. Processors

Configuring the number of processors for a pure NFS server is always easy: take the minimum, at least with Solaris 1.0. NFS runs *only* in the kernel, almost always under the kernel lock, and as a consequence, adding additional cpus just doesn't provide much performance benefit. There are situations where additional processors can make a difference: heavy use of a PrestoServe accelerator, or when the server is expected to provide more than just NFS service.

1.1.7. Other Considerations

These are the basic components of an NFS server, but there are few other things to consider. One of these is the use of a PrestoServe accelerator. In this context, the PrestoServe accelerates NFS writes. If writes constitute a significant proportion of your NFS mix – more than about 6–7% – a PrestoServe can substantially improve a server's performance. A PrestoServe can be used to "serialize" some of those operation (see the PrestoServe discussion in section 2.8.), potentially permitting fewer disks or controllers to be configured. Unfortunately, the cost of using a PrestoServe is increased cpu utilization (5–10%), and for heavily loaded uniprocessor systems, this can push the server "over the edge" and result in *lower* performance. Effectively, the PrestoServe cannot be used on uniprocessors if the cpu utilization is more than about 85%. Because of the nature of the Solaris 1.0 kernel on the SS600MP systems, PrestoServe can almost always be accommodated (see section 2.10.1., *Kernel Lock* for details). In situations where the application is characterized by many NFS write operations, but in which a PrestoServe is not an option, Interstream's eNFS software may be used to lower disk utilization and accelerate the write operations. This happens most frequently in SPARCstation–2 based servers with many disks. The eNFS product is discussed in Section 2.12.

Another consideration is the use of the Online:DiskSuite Metadisk driver. The Metadriver's striping option improves random disk performance by a substantial 15–30% on the SPARC-

server. A substantial fraction of NFS operations are virtual memory page faults, especially in diskless environments. For this reason, we recommend the use of the Network CoProcessor in a diskless environment whenever possible. Virtual memory faults across a network are common even in diskfull and dataless environments, because Solaris simply maps application binaries into virtual memory space, without regard to their actual network location. This can put a premium on NFS response time.

1.1.3. Disk Configuration

The next thing to do is to determine the type and number of disks to use. If there are stringent *continuous availability* requirements, this means using IPI on the SS690 in order to take advantage of the dual–ported nature of the IPI disks. However, data *reliability* can be accomplished on either SCSI or IPI through the use of the Online:DiskSuite Metadisk driver. Since Sun offers only one size of IPI disk, you don't have a lot of flexibility configuring disks.

Obviously, the volume of data to be accessed plays a big part in this; however, especially for smaller configurations, one big disk is often not as useful as several smaller drives, even though the largest disks are usually the fastest. The smaller drives offer more read/write arms and the option of reducing disk utilization. While three 424MB disks cost 15% more than a single 1.3GB disk, they offer the potential of 150 random I/Os per second, as opposed to the 60 available with the larger disk. The pattern of access to the disks is the crucial factor. If the data is spread out among a large number of files and is accessed frequently, try to spread it across a large number of small drives; infrequent access permits the use of fewer large drives.

It is also helpful to understand the nature of usage of the proposed client base as a whole. If each client typically generates bursts of intense NFS activity, followed by long periods of inactivity, it may turn out that the access pattern to the disks on the server is primarily serial. Clients that exhibit this behavior are frequently initializing a large application by serially reading a large file. If the server is not disturbed by many conflicting requests, it may have the opportunity to read data serially (naturally, the client's access pattern would have to be serial too)[1]. Typically, however, client requests arrive steadily and the server ends up with a random access pattern on the disks.

Often a server will provide both kinds of service, and configuring separate disk subsystems for each service class can yield a big performance boost.

1.1.4. Disk Controllers and Host Adapters

Once the number of disks is determined, the number of disk controllers or host adapters can be determined. Again, the server's access pattern is the governing factor. For disks which are accessed primarily serially, up to four disks can be reasonably configured onto a SCSI host adapter. Serially accessed disks configured on an IPI–2 string will saturate the string if they transfer large quantities of data; try to arrange for at most one "hot" disk per string controller (connectivity is fine; there is little penalty for having many drives on a string, as long

1. In cases such as these, it may be worthwhile to configure the older 911MB (CDC/Imprimis 9722) drive, if it is available. These drives perform serial transfers at 6MB/sec, instead of the newer 1.3GB which is dramatically faster for random I/O but has only one 4.5MB/sec head. See Section 2.4., *Disk Subsystems* for more details.

This first information provides an initial estimate of what kind of system is required, as shown in Table 1. Do *not* configure to the maximum load; use the sustained load instead, which is 80% of maximum. Configuring to the maximum means that any increase in demand over the sustained maximum will result in *very* poor response time (possibly five to ten times worse than response at the 80% of maximum level).

	Sustained Load	Maximum Load	Required Nets
SPARCstation–2	320	400	1
SPARCstation–2	360	450	2
SS630MP	440	540	2
SS670MP	650	800	4
SS690MP	650	800	4

Table 1. Nhfsstone loads sustainable by various platforms. These numbers presume the Legato mix, although many current installations report substantially different actual mixes. Required networks lists the number of networks and network interfaces across which the sustainable load must be spread.

1.1.2. Networks and Network Interfaces

Once an estimate of the client load is available, some quick multiplication yields the number of network interfaces that are required to support the proposed client load. The number of required networks often dictates the configuration of the server. The various network interfaces offered by Sun can support loads as described in Table 2.

	NFSops/sec (Legato Mix)	Latency, ms
Lance Ethernet	400	12–42
Intel Ethernet built–in	220	42–69
Intel Ethernet, VME	190	45–68
Network CoProcessor, VME	290	9–37
4Mb token ring	200	16–42
16Mb token ring	390	16–36
FDDI/DX		
FDDI/S		

Table 2. NFS capacity of various Sun networking media as measured by nhfsstone.

Note that the latency associated with the Network CoProcessor is significantly lower than that of any of the raw Ethernet interfaces, although the Network CoProcessor's per–network throughput ceiling is lower due to the limits of the 68020 protocol processor. Even if the performance of the server as a whole is sufficient to meet the anticipated demand, use of the Network CoProcessor can dramatically increase user's perception of the capability of the

1. Server Configuration Estimation

In this section we will make use of architectural and experimental information to provide some guidelines for estimating a correct server configuration. These rules are of necessity very general, in the sense that they will rarely apply precisely to a real–life situation, because the process of accurately estimating a configuration *always* depends on specific data about a system's intended use. Nevertheless, the methods outlined here provide a good starting point and a framework for using such information when it is available. Later chapters will expand upon the details of the architectural information on a component–by–component basis, and provide an introductory guide to tuning and measuring installed systems.

1.1. Configuration of Pure NFS Servers

A pure NFS server is probably the easiest server to configure. By "pure NFS" we are referring to a system whose primary, if not only, task is to provide NFS file service to a group of clients. The reason that an NFS server is relatively easy is that NFS makes a very simple set of demands upon a server: there is only one implementation of the NFS server code (on Suns, anyway); that code always executes in Solaris' kernel space; and only a relatively small subset of Solaris itself is exercised.

The most important information needed for configuring this class of server are the number and application type of clients per network, and the number of networks they will be configured onto. From this information one can arrive at an estimate of the overall and per–network load expected to be demanded of the server.

1.1.1. Client Demand Estimation

The first thing to do is to arrive at some estimate of the number of NFS operations expected to be generated by the clients. This can be hard, primarily because this data is generally unknown. *It is always wise to measure existing similar clients if possible*. Solaris provides the basic tools necessary to measure this demand, and SunNet Manager provides a reasonable framework around which to measure an existing network. Some additional tools are available within Sun, such as `statit, leinfo`, *etc.* Another useful tool is a Network General Sniffer (be sure to get it equipped with the right network interface type). Appendix C. provides a refresher course on measuring and monitoring an operational system. In the absence of real data, consult Appendix B. for some estimates of various client loads.

For a first approximation, use the `nfsstat` command on a client (it may be helpful to zero out nfsstat's counters with the –z option first) and compute the average number of operations demanded. It is of critical importance to take note of the *mix* of operations as well as the volume and rate. Read and especially write operations are by far the most expensive, both because they consume network bandwidth – they are much longer packets – and because they involve many more resources on the server. Fortunately, read and write operations are also the most susceptible to configuration and tuning.

If model clients and SunNet Manager are available, it is often useful to research the behavior of the clients over a longer period of time, in order to discover the variability of the imposed load (of course, this is a terrific opportunity to measure – and tune – the existing server). SunNet Manager provides agents watch network utilization, etc.

Preface

This paper is about making configuration recommendations. In the sense that the customer will probably buy only one system to address a particular need, making a configuration recommendation implies placing limits on the end–results computation that she can achieve. In the past, only the barest configuration guidelines have been published. As a result, the process of specifying a solution–oriented configuration has been something of a black art, requiring knowledge the architecture, abilities and limitations of Sun's processors and peripherals, Solaris (1.0.x), standard networking media, as well as a myriad of other details, such as the customer's applications, personnel, future plans and inclinations. This work attempts to address some of these questions, with the Sun systems engineering force in mind. We are interested in what performance is, and what limits it, how to extend those limits, and how to fix problems with performance when they arise.

In order to limit the scope of this treatise to a manageable size, we have restricted our efforts to the systems which often have the most impact on a customer's day–to–day operations: servers, and specifically server systems in use as database servers, NFS file servers, and to a lesser degree, multiuser timesharing systems. Other topics, such as X–terminals, must wait for more time and opportunity. While we can't read the customer's mind, we will try to propose some rules of thumb to use when laying out a configuration to solve a particular problem. Of course, no such rules of thumb and preliminary estimates can be as accurate as actual measurement of an operational system. Accordingly, we will discuss some of the ways we measured our benchmark systems, how these can be applied to real–life situations, and (hopefully) point out some of the pitfalls we encountered along the way.

In the first section, we will cover general topics of configuration estimation. We will investigate some general rules–of–thumb for configuring various kinds of servers, and present detailed case studies, designed to emphasize many of the principles and rules presented in the text. *The case studies, while designed to be realistic, are completely synthetic, and do not represent actual installations!* In the next section, we will take up the performance aspects of subsystems from which systems are constructed. In this section we will presume an understanding of system and component architecture.Because performance is dependent upon most or all of the components of a system, it is important to investigate the performance of a system both at a microscopic component level and at a macroscopic global level. This provides a basis from which to think about configurations. It is intended to be reference material for the experienced SE. And finally, since even the most expert configuration planner can only estimate a configuration in advance, we will consider system tuning in the third section. This is not intended to be an exhaustive treatise on tuning, only a few pointers on the most salient "knobs and dials" available to measure and improve the performance of already–installed systems. Common benchmarks are taken up in the first Appendix, specifically what they measure and what they don't.

This work is intended to be a guide to the Sun server world, at best a introduction or refresher on conifiguration, capacity estimation, architecture, benchmarking and tuning. There is a wide variety of published work for those interested in these topics in greater depth; references are provided in the biliography.

This work is the result of an ongoing study. If you find, generate or otherwise accumulate data which is relevant, please help to advance this study!

Like any other large undertaking, this project owes much to the work of many people whose names are not on the cover. The authors would like to thank Doug Kaewert, Carl Stolle, Mike Schafir and especially Cheena Srinivasan of Server Systems Marketing for making the entire project possible. Nhan Chu provided guidance and lab facilities, which Sadegh Cameron kindly (and tolerantly!) supported. Maneesh Dhir, Rajiv Khemani, Varun Mehta, and Prasad Wagle of Performance Engineering and Chuck Kollars of the Boston Development Center provided an education and many, many technical insights and useful ideas. Dave Edstrom, Jean Edwards and others read (and reread, and reread and...) the draft copies and provided invaluable advice. And finally we have to thank our wives and significant others for putting up with us while we went off to war to learn about all of these things. And lastly, we have to think Andy, Chuck, Greg, and all of the rest of the engineers for building us such nice toys!

Brian Wong
Pat Shuff
Hal Stern
January 1992

Measure thrice, check twice, cut once; finish by sanding.

watchword for the woodworker

Measure thrice, configure twice, install once; finish by tuning.

watchword for the Systems Engineer

Configuration and Capacity Planning for Sun Servers

A Brief Survey of
Architecture, Configuration and Tuning
in the Sun World

Brian L.Wong

Pat Shuff
Hal L. Stern

Sun USA Systems Engineering
SMCC Technical Marketing
SMCC Performance Engineering

January 1992