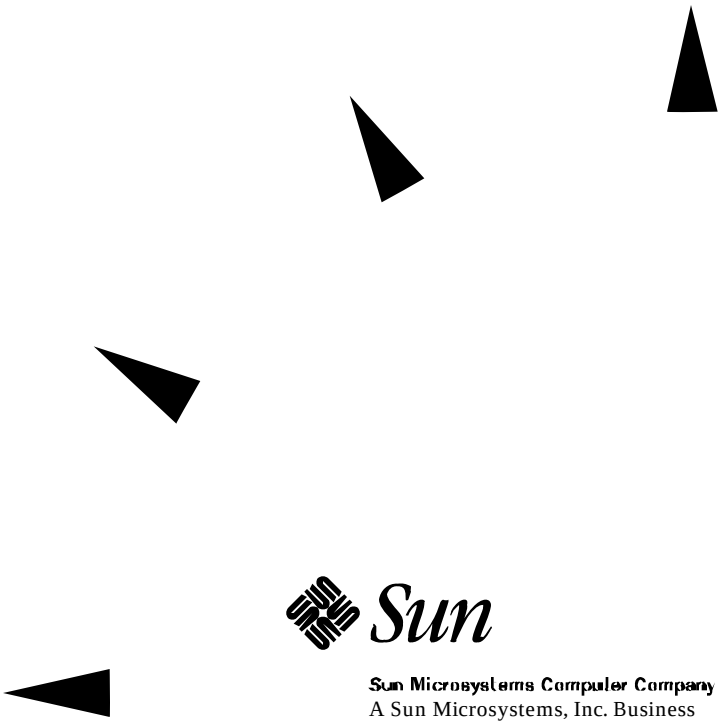
# SMCC NFS Server Performance and Tuning Guide



2550 Garcia Avenue
Mountain View, CA 94043
U.S.A.

Part No: 801-7289-10
Revision A, November 1994

Sun Microsystems Computer Company
A Sun Microsystems, Inc. Business

Please
Recycle

Adobe PostScript

# *Contents*

# *Figures*

# *Tables*

# *Preface*

The *SMCC NFS Server Performance and Tuning Guide* is about the NFS® distributed computing file system. It describes:

- NFS and network performance analysis and tuning
- NFS and network monitoring tools

This book is written with the assumptions that your server is:

- Running Solaris™ 2.x system software
- In a networked configuration
- A Sun-4™ system, SPARCserver™ system, SPARCcenter™ 2000 or 2000E system, or SPARCcluster™ system.

## *Who Should Use This Book*

This book is for system administrators and network specialists who will be configuring, analyzing performance of, and/or tuning servers that provide the NFS service to network clients.

## Related Books

You can find more detailed coverage of some of the topics discussed in this book in the following books:

- *Sun Performance and Tuning (SunSoft Press/Prentice Hall)*
- *Solaris 2.x Handbook for SMCC Peripherals*
- *SunOS 5.x Administering Security, Performance, and Accounting (SunSoft)*
- *SunOS 5.x Administering NFS (SunSoft)*
- *SPARCcluster 1 System Application Guide*
- *SunLink FDDI/S3.0 User's Guide (SunSoft)*

## What Typographic Changes and Symbols Mean

The following table describes the type changes and symbols used in this book.

*Table P-1* Typographic Conventions

| Typeface or Symbol | Meaning | Example |
|---|---|---|
| `AaBbCc123` | The names of commands, files, and directories; on-screen computer output | Edit your `.login` file.<br>Use `ls -a` to list all files.<br>`system% You have mail.` |
| **`AaBbCc123`** | What you type, contrasted with on-screen computer output | `system% su`<br>`Password:` |
| *AaBbCc123* | variable: replace with a real name or value | To delete a file, type `rm` *filename*. |
| *AaBbCc123* | Book titles, new words or terms, or words to be emphasized | Read Chapter 6 in *User's Guide*.<br>These are called *class* options.<br>You *must* be `root` to do this. |
| Code samples are included in boxes and may display the following: | | |
| % | UNIX C shell prompt | `system%` |
| $ | UNIX Bourne and Korn shell prompt | `system$` |
| # | Superuser prompt, all shells | `system#` |

# Introduction 1

This chapter briefly discusses NFS characteristics, the tuning cycle, and third party tools used for NFS monitoring.

## 1.1 NFS Characteristics

The NFS environment provides transparent file access to remote files over a network. File systems of remote devices appear to be local. Clients access remote file systems by using either the `mount` command or the automounter.

The NFS protocol enables multiple client retries and easy crash recovery. The client provides all the information for the server to perform the requested operation. The client retries the request until acknowledged by the server, or until it times out. The server acknowledges writes when the data has been flushed to nonvolatile storage.

The multithreaded kernel does not require the maintenance of multiple `nfsd` or asynchronous-block I/O daemon (`biod`) processes; they are both implemented as operating system kernel threads. There are no `biod`s on the client and one `nfsd` process on the server.

NFS traffic is characterized by its randomness. NFS requests are generated in bursts, and they are usually of many types. The capacity of an NFS server must address the bursty nature of NFS file service demands. Demand varies widely, but is relatively predictable during normal days.

With regard to the types of requests from applications (which may be local or remote), most follow this pattern:

1. The user reads in the sections of the application binary, then executes the code pages leading to a user dialog, which specifies a data set on which to operate.
2. The application reads the data set from the (probably remote) disk.
3. The user can then interact with the application, manipulating the in-memory representation of the data (this phase continues for most of the runtime of the application).
4. The modified dataset is saved to disk.

**Note –** More sections of the application binary may be paged in as the application continues to run in actions 2 through 4.

## 1.2 Tuning Cycle

These tasks comprise the tuning cycles:

1. Measuring current performance and analyzing the data. See Chapter 3, “Analyzing NFS Performance.”
2. Tuning for NFSops/second. See Chapter 4, “Configuration Recommendations for NFS Performance.”
3. Repeating tasks 1 and 2 until no further improvements are required.
4. Reconfiguring new hardware on the server, if needed.
5. Repeating tasks 1 through 3 again.

## 1.3 Third Party Tools Used for NFS Performance Monitoring

Some of the third party tools you can use for NFS/networks include:

- NetMetrix™ (Hewlett-Packard)
- SharpShooter™ (AIM Technology)

## 1.4 Terminology

See the Glossary for definitions of words and phrases used in this book.

# 1

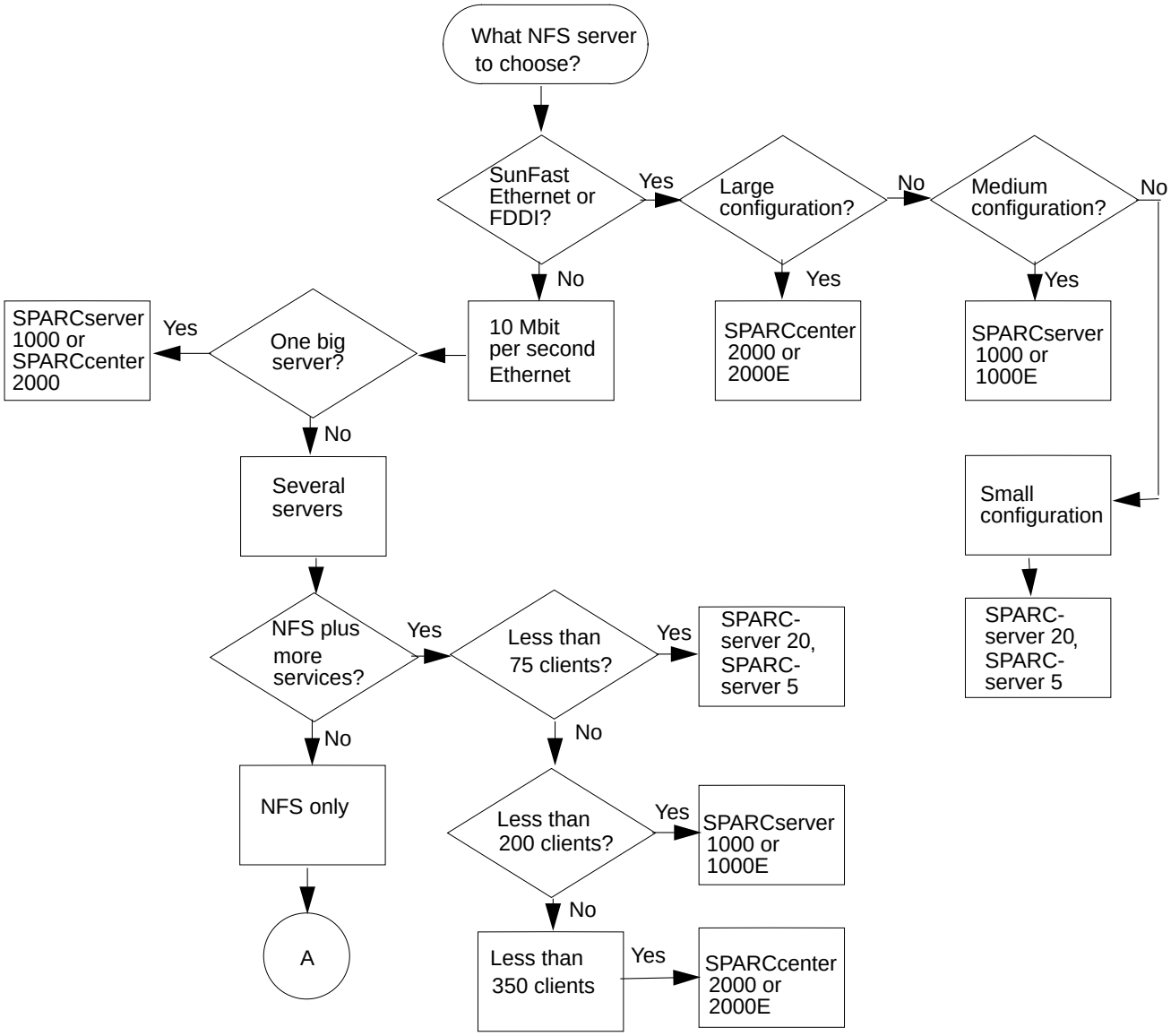
# Hardware Overview 2

This chapter describes an overview of these NFS servers and expansion units:

- SPARCserver™ 5 system
- SPARCserver 10 system
- SPARCserver 20 system
- SPARCcluster 1 system
- SPARCserver 1000 or SPARCserver 1000E system
- SPARCcenter™ 2000 or SPARCserver 2000E system
- SPARCstorage™ Array subsystem
- Multi-Disk Pack
- Desktop Disk Pack

## 2.1 NFS File Servers

This section provides a hardware overview of Sun NFS servers. Figure 2-1 and Table 2-1 illustrate under what conditions a particular Sun NFS file server will meet your needs.




Note: The number of clients depends on the load. This example assumes a medium load.

*Figure 2-1* NFS File Server Comparison Flow Diagram

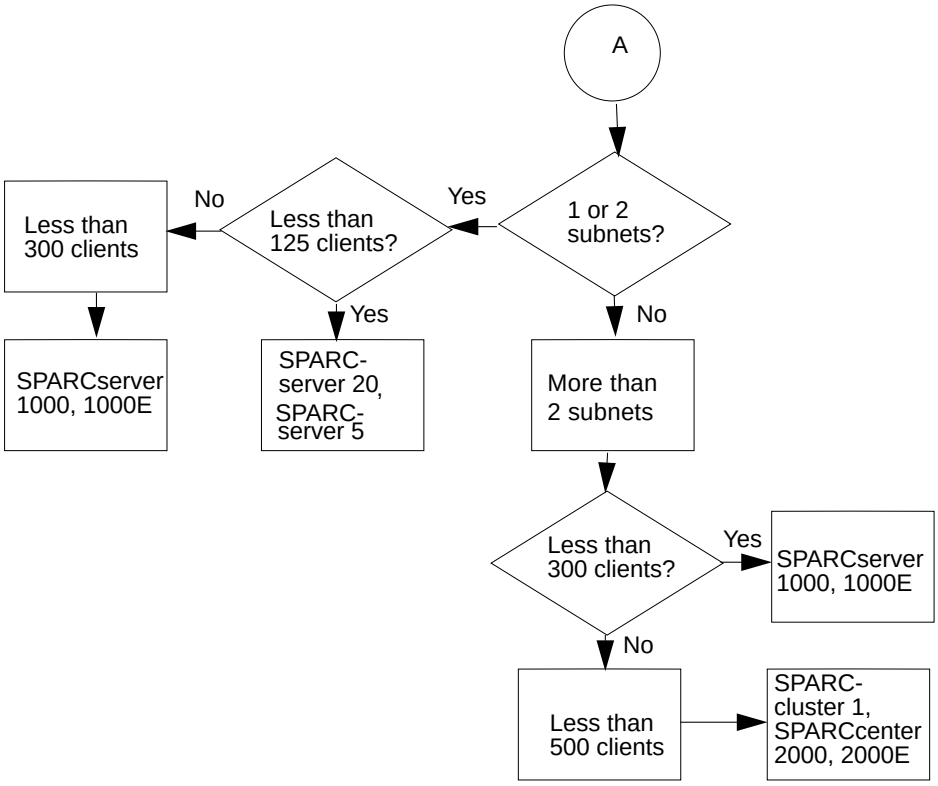



*Figure 2-1* NFS File Server Comparison Flow Diagram *(Continued)*

*Table 2-1* NFS Server Comparison Table

| Server | Maximum Specifications | Positioning | Key Advantages |
|---|---|---|---|
| **SPARCcenter 2000 or 2000E** | 500 NFS clients<br>36 subnets<br>731 Gbytes of disk storage<br>Ethernet, FDDI, SunATM, and token ring | Highest capacity<br>Multipurpose enterprise server (the total solution for an entire company) | Centralized administration<br>Maximum headroom for growth<br>Multiprocessor, I/O, and network performance scalability |
| **SPARCcluster 1** | 500 NFS clients<br>30 subnets<br>150 Gbytes of disk storage<br>Ethernet backbone | High capacity<br>Cluster of distributed servers<br>Dedicated NFS server | Simplified administration<br>Centralized control of distributed servers<br>Lowest cost per NFS ops |
| **SPARCserver 1000 or 1000E** | 300 NFS clients<br>12 subnets<br>395 Gbytes of disk storage<br>Ethernet, FDDI, SunATM, and token ring | High capacity<br>Multipurpose workgroup server | Excellent capacity performance<br>Multipurpose server (NFS, compute, database)<br>Affordable<br>Scalability<br>Integrated packaging |
| **SPARCserver 20** | 125 NFS clients<br>138 Gbytes of disk storage<br>4 subnets<br>Ethernet, FDDI, SunATM, and token ring | Low cost<br>Multipurpose workgroup server<br>PC LAN server | Low cost, yet powerful, flexible, and easily redeployed |
| **SPARCserver 5** | 60 clients<br>40 Gbytes of disk storage<br>4 subnets<br>Ethernet | Low cost<br>Multipurpose workgroup server<br>PC LAN server | Low cost<br>Very good NFS performance at a low cost |

## 2.1.1 SPARCserver 5 System

The SPARCserver 5 system is based on the microSPARC™ II CPU. It provides fast application performance and networking extensibility. It is ideal for a dedicated workgroup NFS server.

### 2.1.1.1 SPARCserver 5 System Features

The SPARCserver 5 system features are:

- 70 MHz or 85 MHz microSPARC II CPU
- 16 to 256 Mbytes of RAM (8 or 32 Mbyte SIMMs)
- 64-bit memory bus
- Two internal hard drives,
- 644 Mbyte CD-ROM drive (optional)
- 1.44 Mbyte diskette drive (optional)
- Twisted pair Ethernet (can be expanded between 5 and 9 Ethernet networks)
  - Optional AUI support
- Three SBus expansion slots
- Two serial ports, one parallel port
- Greater than 20 Gbytes mass storage capacity with external disk drives
- SBus Prestoserve NFS accelerator (optional)

A SPARCserver 5 system can be upgraded to a SPARCserver 20 system by substituting the system board with a SPARCserver 20 system board. Figure 2-2 shows a front view of the SPARCserver 5 system.

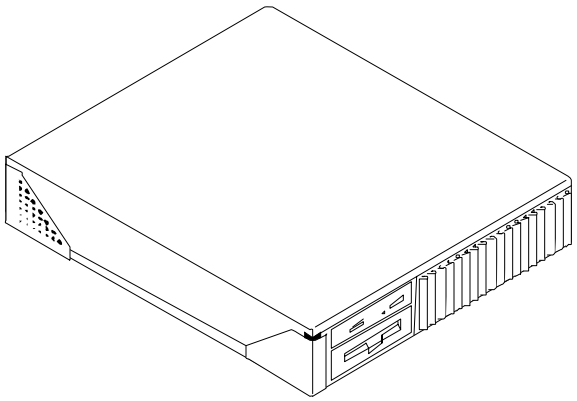


*Figure 2-2* SPARCserver 5 System Front View

### 2.1.2 SPARCserver 10 System

The SPARCserver 10 system is a high-performance workstation designed to be a workgroup NFS file server or a database server in an office environment.

The system is available in various uniprocessor and multiprocessor configurations. Some of the uniprocessor configurations available are:

- Model 30LC—single 36MHz SuperSPARC™ processor, no external cache
- Model 40—single 40MHz SuperSPARC processor, no external cache
- Model 41—single 40MHz SuperSPARC processor, 1 Mbyte of external cache
- Model 51—single 50MHz SuperSPARC processor, 1 Mbyte of external cache

Some of the multiprocessor configurations available are:

- Model 402—two 40 MHz SuperSPARC processors, no external cache
- Model 512—two 50 MHz SuperSPARC processors, each with 1 Mbyte of external cache
- Model 514—four 50 MHz SuperSPARC processors, each with 1 Mbyte of external cache

Figure 2-3 shows a front view of the system.

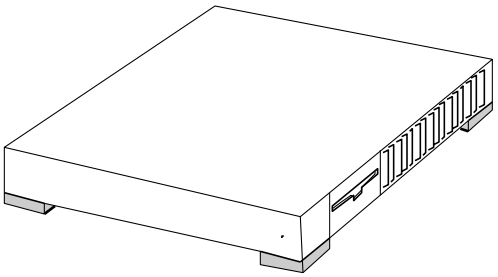


*Figure 2-3* SPARCserver 10 System Front View

### 2.1.2.1 SPARCserver 10 System Features

The features include:

- High performance throughput based on balanced coherent bus, graphic, and networking components
- SuperSPARC superscaler processors set in a multiprocessor architecture
- Memory (RAM) ranging from 32 Mbytes to 512 Mbytes
- With four SPARCstorage Array subsystems (Model 101), the disk storage capacity of the disk arrays is 126 Gbytes.
- Twisted Pair Ethernet (standard) and Thicknet (AUI) with an optional adapter cable
- Four SBus expansion slots
- Two serial ports, one parallel port
- 16-bit audio
- NVRAM-NVSIMM and SBus Prestoserve NFS accelerators (optional)

### 2.1.3 SPARCserver 20 System

The SPARCserver 20 system is designed to be a workgroup NFS file server or a database server in an office environment. It is based on the same MBus and SBus technologies as the SPARCserver 10 system. Performance is increased over the SPARCserver 10 by using faster MBus and SBus technology, and faster SPARC® modules. The SPARCserver 20 system has increased compute and network performance.

The SPARCserver 20 system is available in three uniprocessor configurations and three multiprocessor configurations. The three uniprocessor configurations are:

- Model 50—50 MHz SuperSPARC processor
- Model 51—50 MHz SuperSPARC processor and 1 Mbyte SuperCache
- Model 61—60 MHz SuperSPARC processor and 1 Mbyte SuperCache

The three multiprocessor configurations are:

- Model 502MP—two 50 MHz SuperSPARC processors
- Model 514MP—four 50 MHz SuperSPARC processors and 1 Mbyte SuperCache
- Model 612MP—two 60 MHz SuperSPARC processors and 1 Mbyte SuperCache

Figure 2-4 shows the front view of the SPARCserver 20 system. This system and the SPARCserver 5 system look alike from the front view.

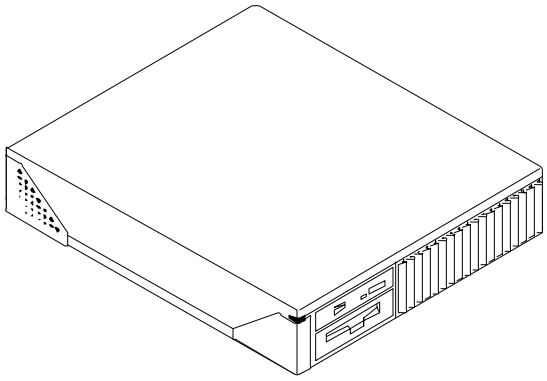


*Figure 2-4* SPARCserver 20 System Front View

### 2.1.3.1 SPARCserver 20 System Features

The SPARCserver 20 system features include:

- Over 2 Gbytes of internal hard disk storage (two 1.05 Gbyte internal disk drives)
- Provides up to 126 Gbytes of disk storage in the SPARCstorage Array (Model 101) subsystems when directly connected to four SPARCstorage Array subsystems
- 1.44 Mbyte diskette drive (optional)
- 644 Mbyte CD-ROM drive (optional)
- Two serial ports, one parallel port
- Twisted pair Ethernet
- Up to 512 Mbytes of memory (60 ns SIMMs)
- AUI Ethernet (optional) (can have up to 9 Ethernet networks)
- SBus or NVRAM-NVSIMM Prestoserve NFS accelerator (optional)

### 2.1.4 SPARCcluster 1 System

The SPARCcluster 1 system is a multiple host server designed for NFS management. The SPARCcluster 1, based on clustering technology, uses up to four SPARCserver 20 system cluster nodes to deliver file services to large departments. Thus, the SPARCcluster 1 system spreads NFS workload across multiple systems. The SPARCcluster 1 system is designed for medium to large NFS file server environments that require a dedicated system for file service. It supports up to 500 clients.

The SPARCcluster 1 system provides:

- A high-uptime server with a minimum of downtime
- A large capacity server that delivers superior response
- A server that can expand easily to accommodate many networks, disks, and memory, without compromising one for the other
- A server emphasizing networking, disks, and CPUs, thus eliminating bottlenecks in NFS file server applications

Figure 2-5 shows the front view of the SPARCcluster 1 system.

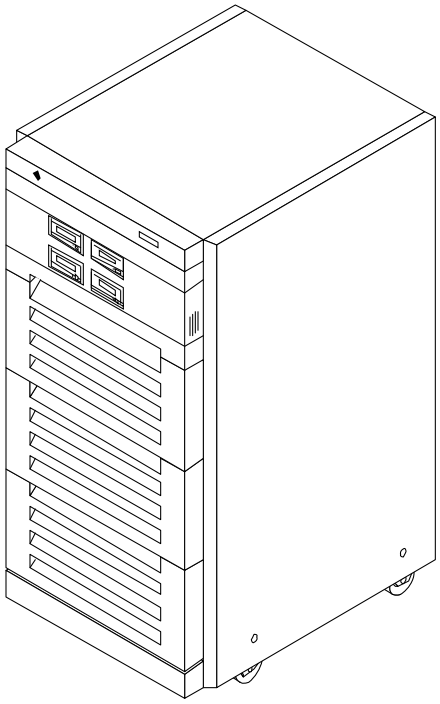


*Figure 2-5* SPARCcluster 1 System Front View

To fully understand how the system works, you must understand the logical architecture and the physical architecture, which are described later in this section.

### 2.1.4.1 SPARCcluster 1 System Features

The features include:

- Fast network file server
- NFS performance scalability
- High uptime though the cluster architecture
- Modular design allows new technologies to be easily incorporated
- Single integrated package configured for optimal NFS file servers
- Centralized system administration tools to manage users, data, and networks

The SPARCcluster 1 system contains the following hardware components:

- Two or four SPARCserver 20 desktop systems
- Two or four differential SCSI disk trays, each containing from two to six 2.9 Gbyte disk drives
- One Multi-Tape Backup Tray containing two to four 5.0 Gbyte 8 mm tape drives
- Terminal concentrator
- Ethernet switch with 6 to 15 ports
- Power distribution unit
- Cables required for internal SCSI, network, and power connections
- NVRAM-NVSIMM Prestoserve NFS accelerator (optional)

### 2.1.4.2 SPARCcluster 1 System Logical Architecture

The SPARCcluster 1 system contains two or four SPARCserver 20 systems, each with a differential SCSI disk tray and a single-ended SCSI tape backup device. The SPARCcluster 1 system can be attached to up to eleven NFS client networks, with each host available on all networks. A high-performance Ethernet network switch connects all the hosts to all client networks.

Figure 2-6 shows the logical architecture of the SPARCcluster 1 system using six client networks, as an example.

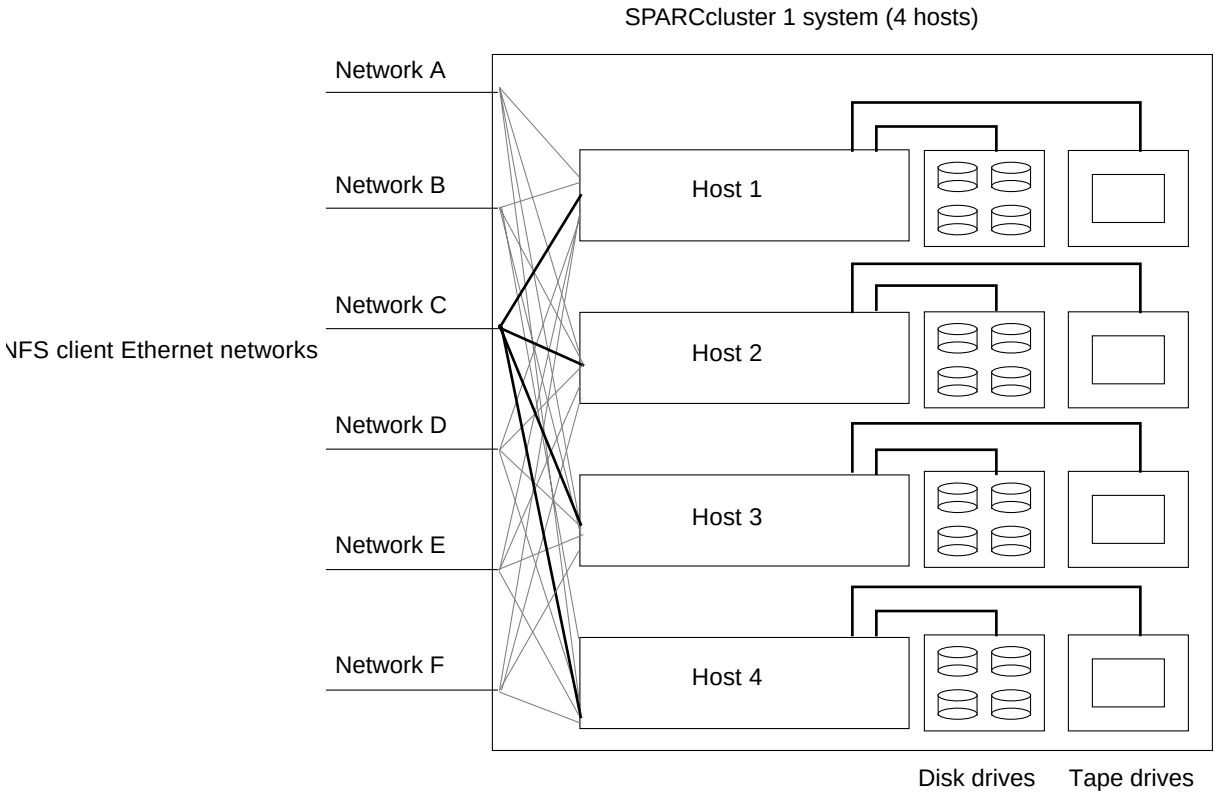



*Figure 2-6* SPARCcluster 1 System Logical Architecture

As Figure 2-6 shows, any workstation attached to any client network has free access to any of the SPARCcluster hosts. In addition, a terminal concentrator is connected to the console of each SPARCserver 20 host, allowing the system administrator to access each host for installation and maintenance from a single, networked workstation.

To implement this logical architecture, one host name and IP address is needed for each host-network logical connection. For example, a system with four hosts and six client networks requires 24 (4 hosts x 6 networks) IP addresses and host names.

### 2.1.4.3 SPARCcluster 1 System Physical Architecture

In contrast to the logical architecture, the SPARCcluster 1 system physical architecture uses only one internal network per server host. Each host is connected directly to the internal Ethernet switch. To complete the connection, all client networks are also connected to the internal Ethernet switch. For more information on the SPARCcluster 1 networking, see Section 2.1.4.4, "SPARCcluster 1 System Network Architecture."

The Ethernet switch and SPARCcluster 1 software perform the multiplexing functions that make each host appear to be connected to every network, even though each host has only one internal network connection. Figure 2-7 shows the physical architecture within the SPARCcluster 1 system.

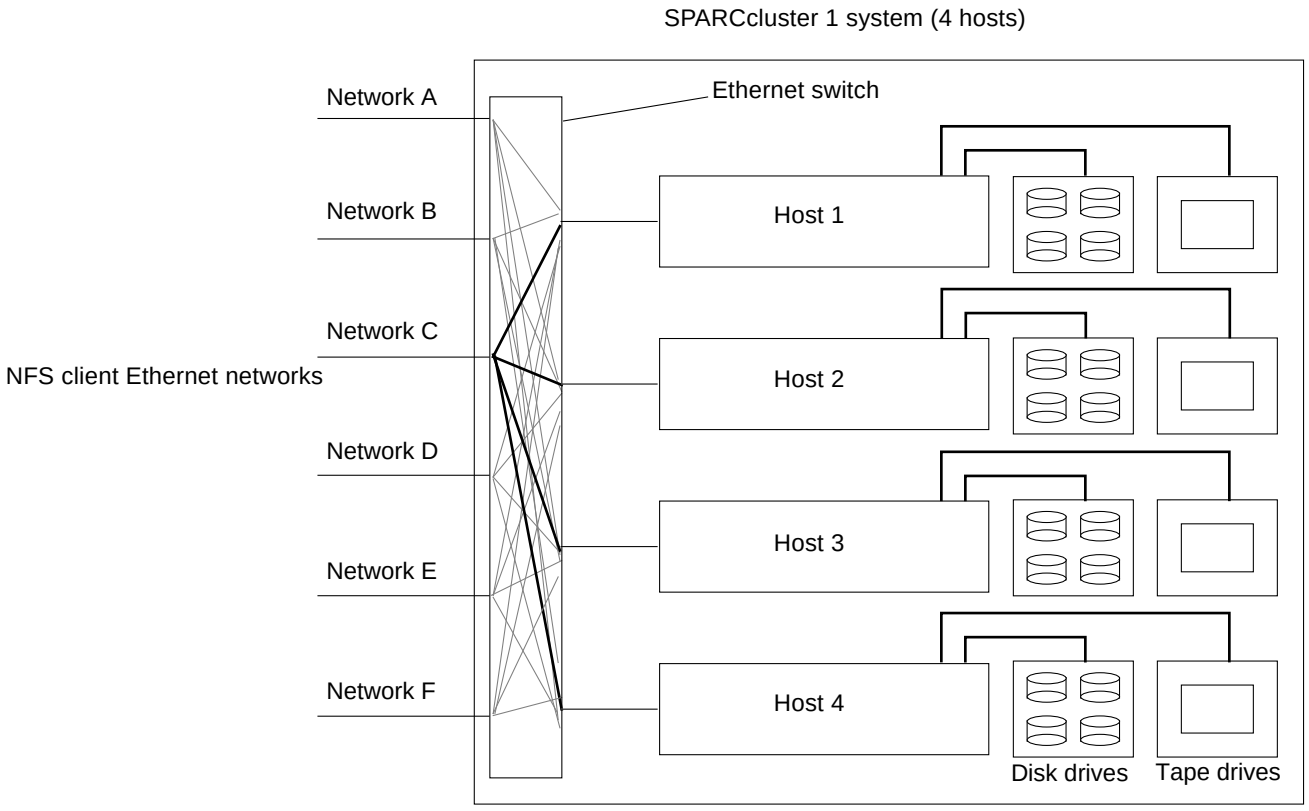



*Figure 2-7* SPARCcluster 1 System Physical Architecture

### 2.1.4.4 SPARCcluster 1 System Network Architecture

Two key concepts underlying the SPARCcluster operation are:

- Port groups
- Ethernet multiplexing

A *port group* is a collection of network input and output ports that are logically combined to appear as a single IP (Internet Protocol) network. *Ethernet multiplexing* is a function that combines the network traffic of several port groups over a single, physical Ethernet cable.

Interfaces on the Ethernet switch are divided into *client ports* and *server ports*. The server ports are also called mux ports, because they contain functionality that is used to multiplex several IP networks over the same physical cable. A SPARCcluster configuration, by default, has two or four server ports, and up to 11 client ports.

The Ethernet switch, connecting all the hosts to the client networks, is a modified, high-speed bridging device using information specific to the SPARCcluster 1 in packet headers to function more like a router than a bridge. The switch retains the benefits of bridging technology, namely low cost per port, extremely low latency (delay) per packet, and high scalability in terms of the number of ports and throughput of the switch.

Adding the host-resident SPARCcluster network multiplexing software and the switch modifications allows the switch to gain the primary advantage of a router: the ability to connect to multiple IP networks, instead of the single IP network restriction usually associated with a bridge. In short, the Ethernet switch combines the best features of a bridge and router so that many IP networks can be multiplexed together over a single Ethernet connection.

No IP routing is provided by the Ethernet switch. It essentially performs a selective forwarding of packets, operating more like a bridge than as a router. To move packets from one IP network to another, an external router or one of the SPARCcluster hosts must forward the packets and update destination information. The Ethernet switch has no knowledge of IP addresses, no store-and-forward capability and it makes no dynamic routing decisions.

Network multiplexing is done by the Ethernet mux (emux) device driver, which sits in between the Lance Ethernet (le) and IP drivers in the network protocol stack. Figure 2-8 shows the relationship between the emux driver, the le driver, and the TCP/IP code used by NFS.

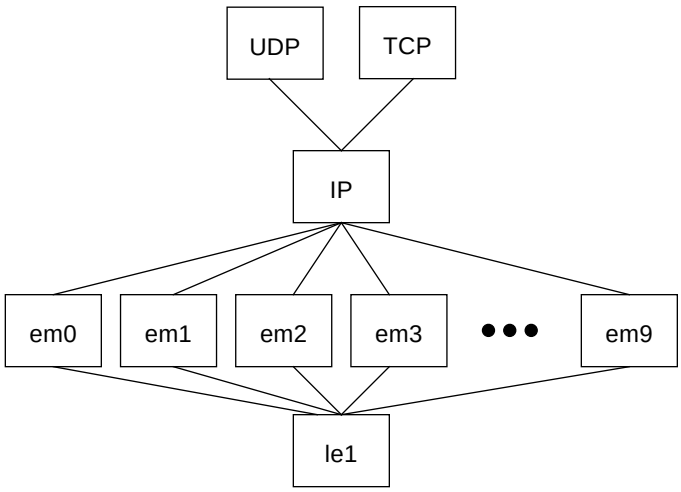



*Figure 2-8* Ethernet Mux Driver

Bridges may be used to join segments connected to the Ethernet switch and other segments that do not attach directly to the SPARCcluster, as shown in Figure 2-9.

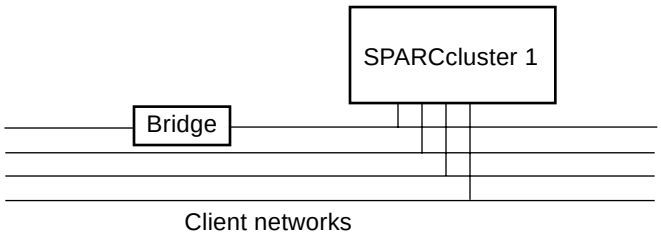



*Figure 2-9* Valid Bridge Configuration with a SPARCcluster 1 System

A common configuration for a SPARCcluster 1 system is a dedicated router, or a set of gateway machines in parallel with the clustered system. Routers and gateways provide high-speed, high-capacity inter-network paths while the

SPARCcluster 1 system is optimized for services between clients and the servers. A typical SPARCcluster 1 system network configuration is shown in Figure 2-10.

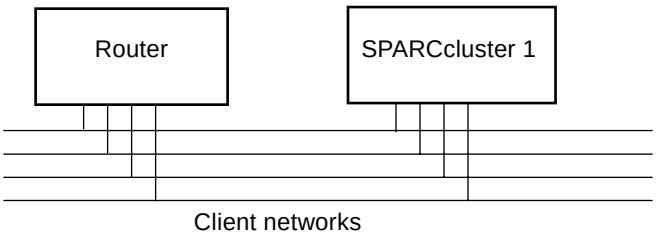



*Figure 2-10* Typical NFS Server and Router Configuration

Routing is done by the hosts in the cluster, not by the Ethernet switch. Therefore, each packet that is routed traverses a server network twice: once from the switch to a routing host, and once from the host to the switch after the packet has been updated with routing information. Routing should be kept to a minimum if the SPARCcluster 1 system is the primary router in the network.

### 2.1.4.5 SPARCcluster 1 System External Physical Architecture

The *primary network* is the client network that is used during installation of the SPARCcluster 1 system. An administration workstation with a SunCD™ drive must be connected to the SPARCcluster 1 system, via the primary network for installation and maintenance. In addition, the terminal concentrator is attached to the primary network. This external configuration assures that all system components are on the same network for installation and management functions.

Figure 2-11 shows the internal and external SPARCcluster 1 system connections.

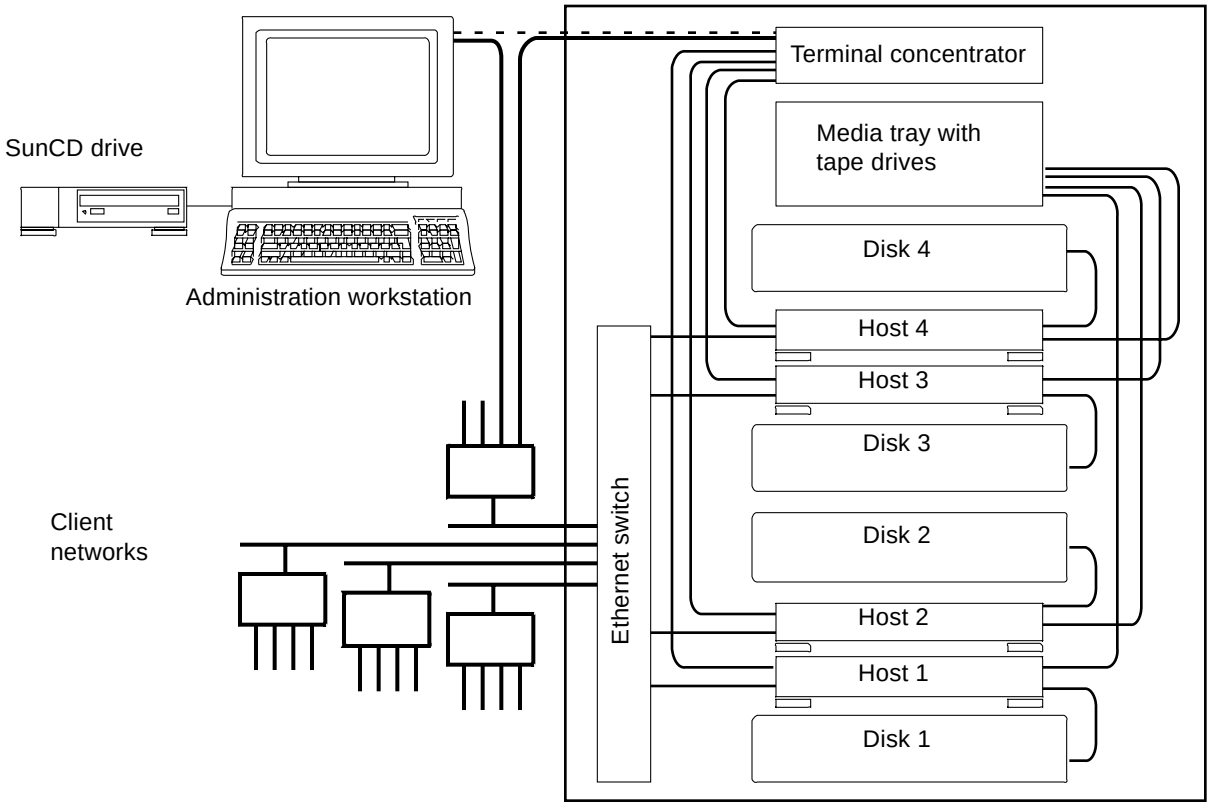



*Figure 2-11* SPARCcluster 1 System Model 4 Typical Physical Architecture

### 2.1.5 SPARCserver 1000 and SPARCserver 1000E Systems

The SPARCserver 1000 and SPARCserver 1000E systems are designed to be departmental NFS file servers with ultimate flexibility for dedicated or multifunctional application environments. They supports less than 200 NFS clients and provide file, database, timeshare, or computing services to a network and attached devices. The SPARCserver 1000 and the SPARCserver 1000E systems feature multiprocessor architecture allowing incremental system expansion.

Modular design makes expansion of system boards, memory and peripherals a routine task you can perform on-site. Processor expansion and upgrades are easily accomplished by adding or replacing SuperSPARC modules. Figure 2-12 shows the front view of the SPARCserver 1000 or the SPARCserver 1000E system.

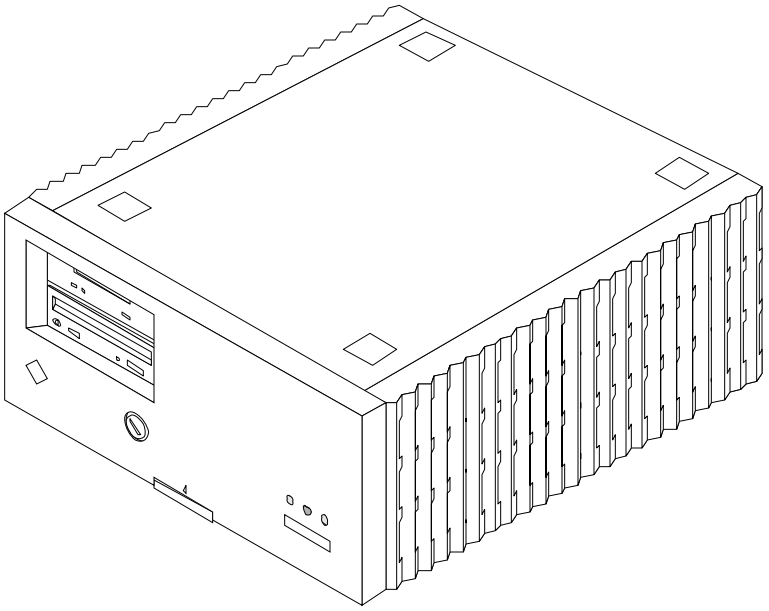


*Figure 2-12* SPARCserver 1000 or the SPARCserver 1000E System Front View

The SPARCserver 1000 and the SPARCserver 1000E systems are built around the XDBus, the system bus that provides very high data transfer bandwidth.

The CPU building blocks consist of:

- Up to 60 MHz SuperSPARC modules (50 MHz SuperSPARC on the SPARCserver 1000 system)
- SuperCache controller
- 1 Mbyte of cache

The system has three SBus slots per system board, which are accessed by the XDBus. The multiple memory banks process in parallel information from multiple CPUs and I/O devices.

### 2.1.5.1 SPARCserver 1000 and the SPARCserver 1000E System Features

The SPARCserver 1000 and the SPARCserver 1000E systems include the following features:

- Up to four system boards can be installed
- Up to eight SuperSPARC processors (two per system board) can be installed
- Up to 2 Gbytes of main memory (using 32 Mbyte SIMMs) can be installed
- Up to 16.8 Gbytes of internal storage
- 50 MHz system clock speed in the SPARCserver 1000E system (40 MHz in the SPARCserver 1000 system)
- 25 MHz SBus speed in the SPARCserver 1000E system (20 MHz in the SPARCserver 1000 system)
- When connected to 12 SPARCstorage Array subsystems, the SPARCserver 1000 or 1000E systems provide up to 378 Gbytes of external storage
- Up to 12 SBus slots
- Onboard SCSI-2 port and twisted pair Ethernet on each system board
- Internal 5 Gbyte 4mm tape drive (or 10 Gbyte 8mm tape drive)
- Internal CD-ROM drive
- NVRAM-NVSIMM Prestoserve NFS accelerator (optional)

### 2.1.6 SPARCcenter 2000 or SPARCcenter 2000E Systems

The SPARCcenter 2000 or the SPARCcenter 2000E systems provide the computing solution for a company. As such, the SPARCcenter 2000 system and the SPARCcenter 2000E system are multifunctional network NFS file servers. They supports less than 500 NFS clients and have the flexibility required for dedicated or multifunctional application environments.

The SPARCcenter 2000 system and the SPARCcenter 2000E system provide scalability and extensive expansion in these areas:

- CPU processor power
- Memory capability
- I/O connectivity

They meet the following requirements:

- High capacity I/O requirements of corporate data centers
- Computationally intensive demands of other organizations

The heart of the SPARCcenter 2000 system or the SPARCcenter 2000E system is a high-speed packet-switched bus complex that provides very high data transfer bandwidth. The backplane supports two distinct XDBuses operating in parallel.

The SPARCcenter 2000 system or the SPARCcenter 2000E system use up to twenty SuperSPARC modules in a shared-memory symmetric multiprocessing configuration, meeting most performance needs. You can expand or upgrade the processing capability by adding SuperSPARC modules.

Main memory is configured in multiple logical units that are installed in the bus complex.

The I/O is expandable. For example, you can configure up to 40 SBus slots on 10 independent busses. The large I/O capacity and configurability makes the SPARCcenter 2000 system or the SPARCcenter 2000E system suitable for very large applications.

### 2.1.6.1 SPARCcenter 2000 and the SPARCcenter 2000E System Features

The SPARCcenter 2000 system and the SPARCcenter 2000E system features are:

- Up to 20 SuperSPARC modules (each with up to 2 Mbytes of built in cache)
- Dual XDbus system bus
- Greater than 5 Gbyte main memory capacity (8 Mbyte or 32 Mbyte SIMMs)
- Up to 10 SBus channels (for a total of 40 SBus slots)
- Two RS232 serial ports on each system board (up to 20 total)
- Twisted pair Ethernet connections via SBus cards
- 10 Mbytes/second SCSI-2 channels via SBus cards
- Up to 52.2 Gbytes of internal SCSI disk storage
- 50 MHz system clock speed in the SPARCserver 2000E system (40 MHz in the SPARCserver 2000 system)
- 25 MHz SBus speed in the SPARCserver 2000E system (20 MHz in the SPARCserver 2000 system)
- When connected to 20 SPARCstorage Array subsystems, Model 101, the total disk capacity for the expansion subsystems is 630 Gbytes
- NVRAM-NVSIMM Prestoserve NFS accelerator (optional)

Figure 2-13 shows the front view of the SPARCcenter 2000 or the SPARCserver 2000E system.

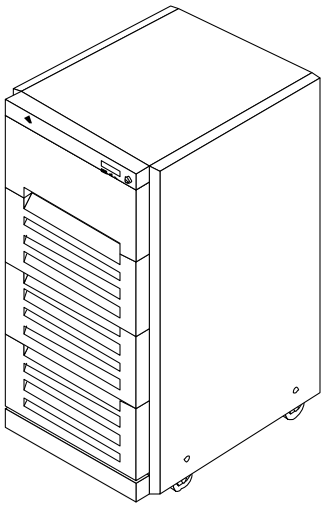


*Figure 2-13* SPARCcenter 2000 or the SPARCcenter 2000E System Front View

## 2.2 Disk Expansion Units

This section describes an overview of the following disk expansion units:

- SPARCstorage Array subsystem
- Desktop Storage Module
- Multi-Disk Pack
- Desktop Disk Pack

### 2.2.1 SPARCstorage Array Subsystem

To expand your disk storage, consider the SPARCstorage Array subsystem. This disk array is a high-performance and high-capacity companion unit for the SPARCcenter 2000 or 2000E, SPARCserver 1000 or 1000E, SPARCserver 10, and the SPARCserver 20 systems.

The Model 101 uses 1.05 Gbyte single connector 3.5-inch disk drives. Each disk array contains three drive trays. Each drive tray supports up to ten 3.5-inch single-connector SCSI disk drives. All disk drive SCSI addresses are hardwired. The position of the disk drive in the drive tray automatically sets the SCSI addresses. Each disk array uses six internal fast, wide SCSI buses. Each drive tray contains two SCSI buses that support five disk drives for each SCSI bus.

Figure 2-14 shows a front view of the SPARCstorage Array subsystem.

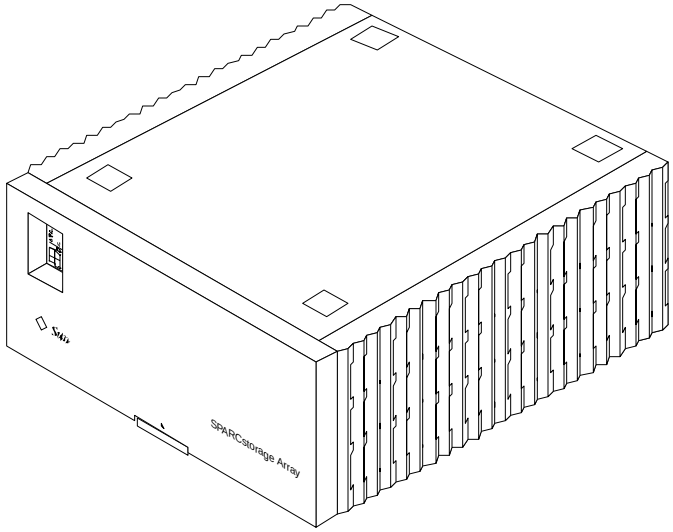
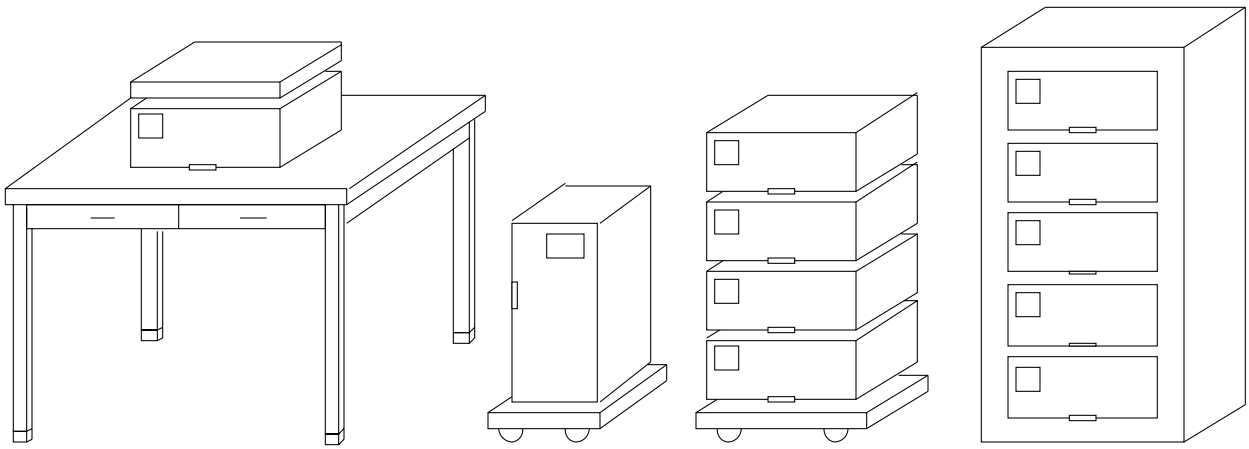



*Figure 2-14* Front View of the SPARCstorage Array Subsystem

Figure 2-15 shows various ways you can connect the SPARCstorage Array subsystem to your NFS server.



**Tabletop**
SPARCserver 20 shown on top.

**Tower**

**Stack**
Maximum of four arrays or SPARCserver 1000 systems.

**Sun Rack**
Maximum of five SPARCstorage Array subsystems.

*Figure 2-15* SPARCstorage Array Subsystem Installation Options

The SPARCstorage Array subsystem uses RAID (Redundant Array of Inexpensive Disks) technology. The levels of RAID available are:

- RAID 0 — Striping data without parity
- RAID 1 — Mirroring
- RAID 0+1 — Mirroring optimized stripes
- RAID 5 — Striping data with parity

Within the disk array, independent disks plus RAID levels 5, 1, 0, 0+1 are available at the same time. This allows you to easily match data layouts to meet the specific requirements for capacity, performance, high availability, and cost.

If any disk in a RAID 5, 1, or 0+1 group fails, an optional hot spare (if configured) is automatically swapped in to replace the failed disk. Continuous, redundant data protection is provided, even if a disk fails.

Warm plug service allows you to replace one or more disks without needing to power down the system, the disk array, or reboot the system. You can obtain warm plug service if multiple disk arrays are configured.

Using the SPARCstorage Array subsystem can improve NFS performance. Its processor manages and schedules disk I/O.

The SPARCstorage Manager software, provided with the disk array, provides similar functionality to Online: Disk Suite software. Since there are often many more disks to manage in the SPARCstorage Array subsystem, the SPARCstorage Manager software has an intuitive GUI interface.

### 2.2.2 Desktop Storage Module

The Desktop Storage Module (disk unit) is an external hard disk drive unit for desktop servers and the SPARCserver series. It contains one full-height 1.3 Gbyte disk drive. Figure 2-16 shows a front view of the Desktop Storage Module.

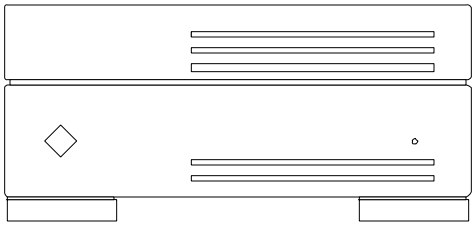


*Figure 2-16* Front View of the Desktop Storage Module 1.3 Gbyte Disk Drive Unit

### 2.2.3 Multi-Disk Pack

The Multi-Disk Pack is a disk expansion unit for the SPARCserver 5, SPARCserver 10, and SPARCserver 20 systems. It comes in two configurations. The first configuration is four 1.05 Gbyte fast SCSI disk drives totaling 4.2 Gbytes of disk storage. The second configuration is either two or four 2.1 Gbyte fast SCSI disk drives totaling 4.2 or 8.4 Gbytes of disk storage. The SCSI addresses are set by SCSI address jumpers on the disk drives. Figure 2-17 shows a front view of the Multi-Disk Pack.

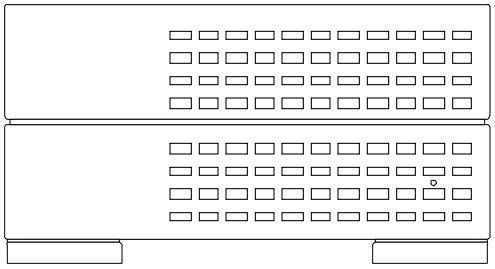


*Figure 2-17* Front View of the Multi-Disk Pack

### 2.2.4 *Desktop Disk Pack*

The Desktop Disk Pack is a disk expansion unit containing one SCSI disk. Some of the disk capacities are:

- 207 Mbyte low profile disk drive
- 424 Mbyte disk drive
- 535 Mbyte low profile disk drive
- 1.05 Gbyte low profile disk drive

The SCSI address is set by a SCSI address switch at the rear of the unit. This expansion unit is useful for the following desktop server systems:

- SPARCstation 5 system
- SPARCstation 10 system
- SPARCstation 20 system

Figure 2-18 illustrates a front view of the Desktop Disk Pack.

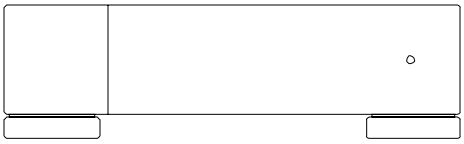


*Figure 2-18* Front View of the Desktop Disk Pack

# 2

# Analyzing NFS Performance 3

This chapter covers how to analyze NFS performance and briefly describes the general steps needed to tune your system. This chapter also describes how to check the network, server, and each client.

## 3.1 Tuning Steps

You will occasionally be required to tune your NFS server. You may want to tune a server for best performance when you set it up, and later to improve performance in response to a problem.

### ▼ General Performance Improvement Tuning Steps

When you tune to improve performance either on an existing system or on a newly setup system:

1. **Measure the current level of performance for the network, server, and each client.**
   See Section 3.2, “Checking the Network, Server, and Each Client.”
2. **Analyze the data gathered.**
3. **Tune the server.**
   See Chapter 4, “Configuration Recommendations for NFS Performance.”
4. **Repeat steps 1 through 3 until you achieve the desired performance level.**

## ▼ Performance Problem Resolution Tuning Steps

When you tune as the result of a performance problem:

1. **Observe symptoms and use tools to pinpoint the source of problem.**
2. **Measure the current level of performance for the network, server, and each client.**
   See Section 3.2, “Checking the Network, Server, and Each Client.”
3. **Analyze the data gathered.**
4. **Tune the server.**
   See Chapter 4, “Configuration Recommendations for NFS Performance.”
5. **Repeat steps 1 through 4 until you achieve the desired performance level.**

## 3.2 Checking the Network, Server, and Each Client

Before you can tune the NFS server, you must check the performance of the network, the NFS server, and each client. The first step is to check the performance of the network.

### 3.2.1 Checking the Network

If disks are operating normally, check network usage because a slow server and a slow network look the same to an NFS client.

Figure 3-1 illustrates the steps you must follow in sequence to check the network.

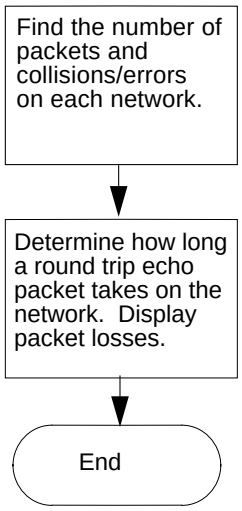



*Figure 3-1* Flow Diagram to Check the Network

### ▼ To find the number of packets and collisions/errors on each network

1. **Type** `netstat -i 15`.

```
server% netstat -i 15
      input   le0       output    input       (Total)    output
packets errs  packets errs  colls  packets errs  packets errs  colls
10798731 533   4868520 0     1078   24818184 555   14049209 157   894937
51      0     43      0     0      238     0     139     0     0
85      0     69      0     0      218     0     131     0     2
44      0     29      0     0      168     0     94      0     0
```

Use `-I` (Capital I) *interface* to look at other interfaces.

| | |
|---|---|
| `-i` | Shows the state of the interfaces that are used for TCP/IP traffic |
| `15` | Collects information every 15 seconds |

In the `netstat -i 15` display, a machine with active network traffic should show both input packets and output packets continually increasing.

**a. Calculate the network collision rate by dividing the number of output collision counts (`Output Colls` - `le`) by the number of output packets (le).**
A network-wide collision rate greater than 10 percent can indicate problems such as an overloaded network, a poorly configured network, or hardware problems.

**b. Calculate the input packet error rate by dividing the number of input errors (le) by the total number of input packets (le).**
If the input error rate is high (over 25 percent), the host may be dropping packets.

Other hardware on the network, as well as heavy traffic and low-level hardware problems, can introduce transmission problems. Bridges and routers can drop packets, forcing retransmissions and causing degraded performance.

Bridges also cause delays when they examine packet headers for Ethernet addresses. During these examinations, bridge network interfaces may drop packet fragments.

To compensate for bandwidth-limited network hardware:

- Reduce the packet size specifications.
- Set the read buffer size, `rsize`, and the write buffer size, `wrsize`, when using `mount` or in the `/etc/vfstab` file. Reduce the appropriate variable(s) (depending on the direction of data passing through the bridge) to 2048.
  If data passes in both directions through the bridge or other device, reduce both variables:

```
server:/home  /home/server nfs rw,rsize=2048,wsize=2048 0 0
```

If a lot of read and write requests are dropped and the client is communicating with the server using the User Datagram Protocol (UDP), then the entire packet will be retransmitted, instead of the dropped packets.

### ▼ To determine how long a round trip echo packet takes on the network and to display packet losses

♦ **Type** `ping -sRv` *servername* **from the client to show the route taken by the packets.**
If the round trip takes more than a few milliseconds (ms), the network is slow, there are slow routers on the network, or the network is very busy. Ignore the results from the first `ping` command

```
client% ping -sRv servername
PING dreadnought: 56 data bytes
64 bytes from server (129.145.72.15): icmp_seq=0. time=5. ms
  IP options:  <record route> router (129.145.72.1), server
(129.145.72.15), client (129.145.70.114),  (End of record)
64 bytes from server (129.145.72.15): icmp_seq=1. time=2. ms
  IP options:  <record route> router (129.145.72.1), server
(129.145.72.15), client (129.145.70.114),  (End of record)
```

`s` Sends one datagram per second and prints one line of output for every echo response it receives. If there is no response, no output is produced.

`R` Record route. Sets the Internet Protocol (IP) record option which stores the route of the packet inside the IP header.

`v` Verbose option. Lists any ICMP packets other than echo response that are received.

If you suspect a physical problem, use `ping -sRv` to find the response time of several hosts on the network. If the response time (ms) from one host is not what you would expect, investigate that host.

The `ping` command uses the ICMP protocol's echo request datagram to elicit an ICMP echo response from the specified host or network gateway. It can take a long time on a time-shared NFS server to obtain the ICMP echo. The distance from the client to the NFS server is a factor for how long it takes to obtain the ICMP echo from the server.

Figure 3-2 shows the possible responses or the lack of response to the `ping -sRv` command.

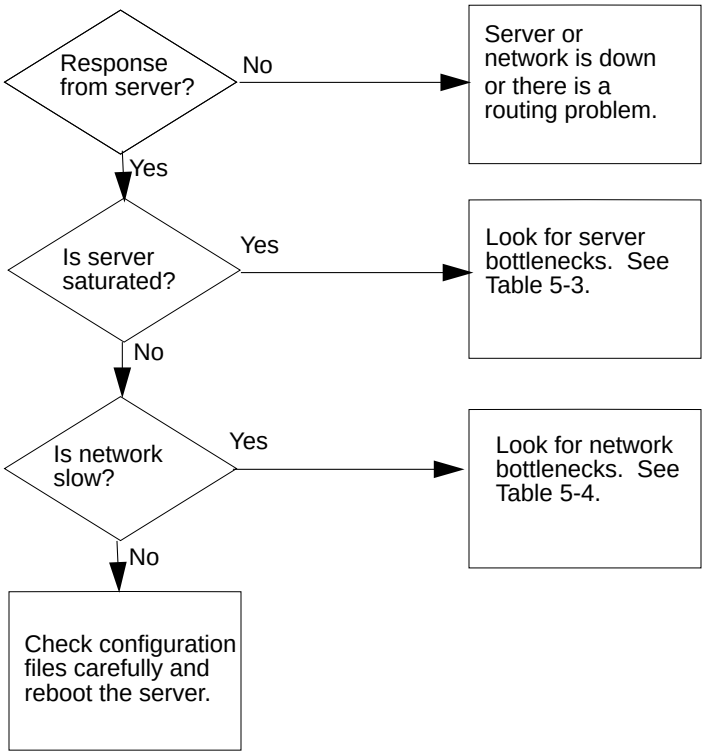



*Figure 3-2* Flow Diagram of Possible Responses to the `ping -sRv` Command

## 3.2.2 Checking the NFS Server

**Note –** The server used in the following examples is a large SPARCserver 690 configuration.

Figure 3-3, which follows, illustrates the steps you must follow in sequence to check the NFS server.

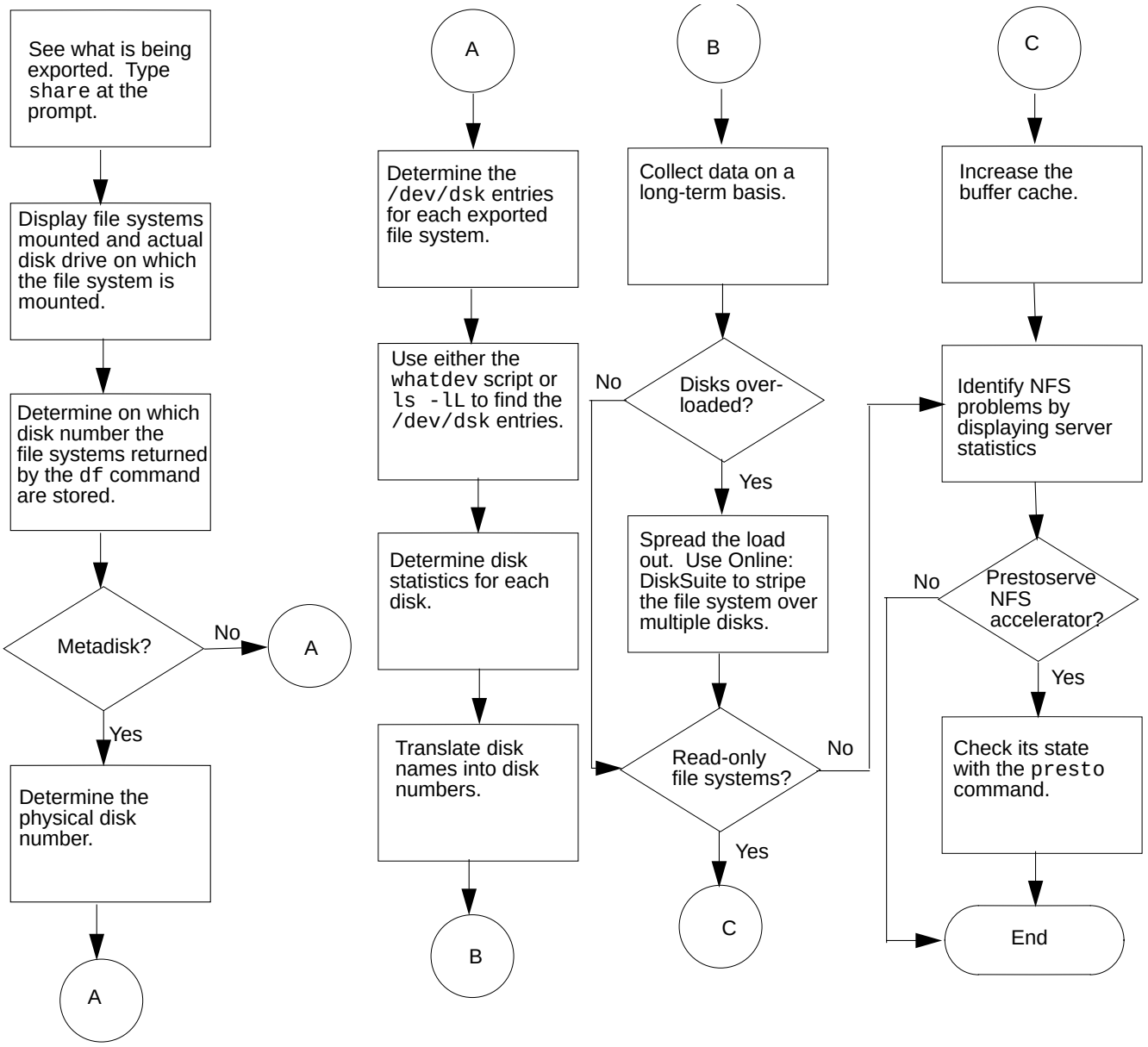



*Figure 3-3* Flow Diagram to Check the NFS Server

### ▼ To see what is being exported

♦ **Type** `share` **at the** `%` **prompt.**

```
server% share
-               /export/home    rw=netgroup    ""
-               /var/mail    rw=netgroup    ""
-               /cdrom/solaris_2_3_ab    ro    ""
```

### ▼ To display the file systems mounted and the actual disk drive on which the file system is mounted

♦ **Type** `df -k` **at the** `%` **prompt.**
If an file system is over 100 percent full, it may cause NFS write errors on the clients.

```
server% df -k
Filesystem            kbytes    used   avail capacity  Mounted on
/dev/dsk/c1t0d0s0     73097     36739  29058     56%   /
/dev/dsk/c1t0d0s3     214638    159948 33230     83%   /usr
/proc                  0          0       0         0% /proc
fd                     0          0       0         0% /dev/fd
swap                  501684     32     501652    0%   /tmp
/dev/dsk/c1t0d0s4     582128     302556  267930    53%   /var/mail
/dev/md/dsk/d100      7299223    687386  279377    96%   /export/home
/vol/dev/dsk/c0t6/solaris_2_3_ab
                     113512     113514  0       100%   /cdrom/solaris_2_3_ab
```

**Note –** For this example, the `/var/mail` and `/export/home` file systems are used.

### ▼ Determine on which disk number the file systems returned by the `df -k` command are stored

In the previous example, you'll note that `/var/mail` is stored on `/dev/dsk/c1t0d0s4` and `/export/home` is stored on `/dev/md/dsk/d100`, an Online: DiskSuite™ meta disk.

### ▼ If an Online: DiskSuite metadisk is returned by the `df -k` command, determine the disk number

♦ **Type** `/usr/opt/SUNWmd/sbin/metastat` *<disknumber>*.
In the previous example, `/usr/opt/SUNWmd/sbin/metastat d100` determines what physical disk `/dev/md/dsk/d100` uses.

---

**Note –** The d100 disk is a mirrored disk. Each mirror is made up of three striped disks of one size concatenated with four striped disks of another size. There is also a hot spare disk. This system uses IPI disks, `id`*X*. SCSI disks, `sd`*X*, are treated identically.

---

```
server% /usr/opt/SUNWmd/sbin/metastat d100
d100: metamirror
    Submirror 0: d10
      State: Okay
    Submirror 1: d20
      State: Okay
    Regions which are dirty: 0%
    Pass = 1
    Read option = round-robin (default)
    Write option =  parallel (default)
    Size: 15536742 blocks
d10: Submirror of d100
    State: Okay
    Hot spare pool: hsp001
    Size: 15536742 blocks
    Stripe 0: (interlace : 96 blocks)
    Device              Start Block  Dbase State        Hot Spare
    /dev/dsk/c1t1d0s7          0     No    Okay
    /dev/dsk/c2t2d0s7          0     No    Okay
    /dev/dsk/c1t3d0s7          0     No    Okay
    Stripe 1: (interlace : 64 blocks)
    Device              Start Block  Dbase State        Hot Spare
    /dev/dsk/c3t1d0s7          0     No    Okay
    /dev/dsk/c4t2d0s7          0     No    Okay
    /dev/dsk/c3t3d0s7          0     No    Okay
    /dev/dsk/c4t4d0s7          0     No    Okay
d20: Submirror of d100
    State: Okay
    Hot spare pool: hsp001
Size: 15536742 blocks
    Stripe 0: (interlace : 96 blocks)
    Device              Start Block  Dbase State        Hot Spare
    /dev/dsk/c2t1d0s7          0     No    Okay
    /dev/dsk/c1t2d0s7          0     No    Okay
    /dev/dsk/c2t3d0s7          0     No    Okay
    Stripe 1: (interlace : 64 blocks)
    Device              Start Block  Dbase State        Hot Spare
    /dev/dsk/c4t1d0s7          0     No    Okay
    /dev/dsk/c3t2d0s7          0     No    Okay
    /dev/dsk/c4t3d0s7          0     No    Okay
    /dev/dsk/c3t4d0s7        0    No   Okay     /dev/dsk/c2t4d0s7
```

### ▼ To determine the /dev/dsk entries for each exported file system

Use the `whatdev` script to find the instance or nickname for the drive or type `ls -lL /dev/dsk/c1t0d0s4` and `more /etc/path_to_inst` to find the `/dev/dsk` entries.

Follow either the procedure in the section "Using the whatdev Script" or the section "Using ls -lL to Identify /dev/dsk Entries."

#### *Using the* whatdev *Script*

**a. Type the `whatdev` script using a text editor.**

```
#!/bin/csh
# print out the drive name - st0 or sd0 - given the /dev entry
# first get something like "/iommu/.../.../sd@0,0"
set dev = `/bin/ls -l $1 | nawk '{ n = split($11, a, "/");
split(a[n],b,":"); for(i = 4; i < n; i++) printf("/%s",a[i]);
printf("/%s\n", b[1]) }'`
if ( $dev == "" ) exit
# then get the instance number and concatenate with the "sd"
nawk -v dev=$dev '$1 ~ dev { n = split(dev, a, "/"); split(a[n], \
b, "@"); printf("%s%s\n", b[1], $2) }' /etc/path_to_inst
```

**b. Type `df /`*<filesystemname>* to determine the `/dev/dsk` entry for the file system.**

In this example you would type `df /var/mail`.

```
furious% df /var/mail
Filesystem            kbytes    used   avail capacity  Mounted on
/dev/dsk/c1t0d0s4     582128  302556  267930    53%    /var/mail
```

**c. Type** `whatdev` *diskname* **(the disk name returned by the** `df /`*<filesystemname>* **command) to determine the disk number.**
In this example you would type `whatdev /dev/dsk/c1t0d0s4`.

Disk number `id8` is returned. This is IPI disk 8.

```
server% whatdev /dev/dsk/c1t0d0s4
id8
```

**d. Repeat steps b and c for each file system not stored on a meta disk (**`dev/md/dsk`**).**

**e. If the file system is stored on a meta disk, (**`dev/md/dsk`**), look at the** `metastat` **output and run the** `whatdev` **script on** *all* **the drives making up the meta disk.**
In this example type `whatdev /dev/dsk/c2t1d0s7`.

There are 14 disks in the `/export/home` file system. Running the `whatdev` script on the `/dev/dsk/c2t1d0s7` disk, one of the 14 disks comprising the `/export/home` file system, returns the following display.

```
server% whatdev /dev/dsk/c2t1d0s7
id17
```

Note that `/dev/dsk/c2t1d0s7` is disk `id17`. This is IPI disk 17.

**f. Go to the procedure "To see the disk statistics for each disk" on page 3-15."**

### *Using* `ls -lL` *to Identify* `/dev/dsk` *Entries*

If you followed the procedure "Using the whatdev Script" skip this section. Go to the procedure "To see the disk statistics for each disk."

If you did not follow the procedure outlined in"Using the whatdev Script:"

**a. Type** `ls -lL` <*disknumber*> **to list the drive and its major and minor device numbers.**
For example, for the `/var/mail` file system, type:
`ls -lL /dev/dsk/c1t0d0s4`.

```
ls -lL /dev/dsk/c1t0d0s4
brw-r-----  1 root      66,  68 Dec 22 21:51 /dev/dsk/c1t0d0s4
```

**b. In the** `ls -lL` **output, locate the minor device number.**
In the previous screen example, the first number following the file ownership (`root`), `66`, is the major number. The second number, `68`, is the minor device number.

**c. Determine the disk number.**
Divide the minor device number, 68 in the previous example, by 8.

Disk number = 68/8

The answer is 8.5. Truncate the fraction. The number, 8, is the disk number.

**d. Determine the slice (partition) number.**
Look at the number following the s (for slice) in the disk number. For example, in `/dev/dsk/c1t0d0s4`, the 4 following the s refers to slice 4.

Now you know that the disk number is 8 and the slice number is 4. This disk is either `sd8` (SCSI) or `ip8` (IPI).

**e. Go to the procedure "To see the disk statistics for each disk" which follows.**

### ▼ To see the disk statistics for each disk

♦ **Type** `iostat -x 15`.
The `-x` option supplies extended disk statistics. The `15` means disk statistics are gathered every 15 seconds.

```
server% iostat -x 15
extended disk statistics
disk      r/s  w/s   Kr/s   Kw/s wait actv  svc_t  %w  %b
id10      0.1  0.2    0.4    1.0  0.0  0.0   24.1   0   1
id11      0.1  0.2    0.4    0.9  0.0  0.0   24.5   0   1
id17      0.1  0.2    0.4    1.0  0.0  0.0   31.1   0   1
id18      0.1  0.2    0.4    1.0  0.0  0.0   24.6   0   1
id19      0.1  0.2    0.4    0.9  0.0  0.0   24.8   0   1
id20      0.0  0.0    0.1    0.3  0.0  0.0   25.4   0   0
id25      0.0  0.0    0.1    0.2  0.0  0.0   31.0   0   0
id26      0.0  0.0    0.1    0.2  0.0  0.0   30.9   0   0
id27      0.0  0.0    0.1    0.3  0.0  0.0   31.6   0   0
id28      0.0  0.0    0.0    0.0  0.0  0.0    5.1   0   0
id33      0.0  0.0    0.1    0.2  0.0  0.0   36.1   0   0
id34      0.0  0.2    0.1    0.3  0.0  0.0   25.3   0   1
id35      0.0  0.2    0.1    0.4  0.0  0.0   26.5   0   1
id36      0.0  0.0    0.1    0.3  0.0  0.0   35.6   0   0
id8       0.0  0.1    0.2    0.7  0.0  0.0   47.8   0   0
id9       0.1  0.2    0.4    1.0  0.0  0.0   24.8   0   1
sd15      0.1  0.1    0.3    0.5  0.0  0.0   84.4   0   0
sd16      0.1  0.1    0.3    0.5  0.0  0.0   93.0   0   0
sd17      0.1  0.1    0.3    0.5  0.0  0.0   79.7   0   0
sd18      0.1  0.1    0.3    0.5  0.0  0.0   95.3   0   0
sd6       0.0  0.0    0.0    0.0  0.0  0.0  109.1   0   0
```

The `iostat -x 15` command lets you see the disk number for the extended disk statistics. In the next procedure you will use a `sed` script to translate the disk names into disk numbers.

The output for the extended disk statistics is:

| | |
|---|---|
| `r/s` | Reads per second |
| `w/s` | Writes per second |
| `Kr/s` | Kbytes read per second |
| `Kw/s` | Kbytes written per second |
| `wait` | Average number of transactions waiting for service (queue length) |
| `actv` | Average number of transactions actively being serviced |
| `svc_t` | Average service time, (milliseconds) |
| `%w` | Percentage of time the queue is not empty |
| `%b` | Percentage of time the disk is busy |

### ▼ To translate the disk names into disk numbers

Use `iostat` and `sar`. One quick way to do this is to use a `sed` script.

1. **Type in a `sed` script using a text editor similar to the following `d2fs.server sed` script.**
   Your `sed` script should substitute the file system name for the disk number.

   In this example, disk `id8` is substituted for `/var/mail` and disks `id9, id10, id11, id17, id18, id19, id25, id26, id27, id28, id33, id34, id35,` and `id36` are substituted for `/export/home`.

```
sed 's/id8 /var/mail/
    s/id9 /export/home/
    s/id10 /export/home/
    s/id11 /export/home/
    s/id17 /export/home/
    s/id18 /export/home/
    s/id25 /export/home/
    s/id26 /export/home/
    s/id27 /export/home/
    s/id28 /export/home/
    s/id33 /export/home/
    s/id34 /export/home/
    s/id35 /export/home/
    s/id36 /export/home/'
```

**2. Type `iostat -xc 15 | d2fs.server` to run the `iostat -xc 15` command through the `sed` script.**

`-x` Supplies extended disk statistics.

`-c` Reports the percentage of time the system was in user mode (`us`), system mode (`sy`), waiting for I/O (`wt`), and idling (`id`).

`15` Means disk statistics are gathered every 15 seconds.

```
% iostat -xc 15 | d2fs.server
extended disk statistics          cpu
disk           r/s  w/s   Kr/s   Kw/s wait actv  svc_t  %w  %b  us sy wt id
export/home      0.1  0.2   0.4    1.0  0.0  0.0   24.1   0   1   0 11  2 86
export/home      0.1  0.2    0.4    0.9  0.0  0.0   24.5   0   1
export/home      0.1  0.2    0.4    1.0  0.0  0.0   31.1   0   1
export/home      0.1  0.2    0.4    1.0  0.0  0.0   24.6   0   1
export/home      0.1  0.2    0.4    0.9  0.0  0.0   24.8   0   1
id20             0.0  0.0    0.1    0.3  0.0  0.0   25.4   0   0
export/home      0.0  0.0    0.1    0.2  0.0  0.0   31.0   0   0
export/home      0.0  0.0    0.1    0.2  0.0  0.0   30.9   0   0
export/home      0.0  0.0    0.1    0.3  0.0  0.0   31.6   0   0
export/home      0.0  0.0    0.0    0.0  0.0  0.0    5.1   0   0
export/home      0.0  0.0    0.1    0.2  0.0  0.0   36.1   0   0
export/home      0.0  0.2    0.1    0.3  0.0  0.0   25.3   0   1
export/home      0.0  0.2    0.1    0.4  0.0  0.0   26.5   0   1
export/home      0.0  0.0    0.1    0.3  0.0  0.0   35.6   0   0
var/mail         0.0  0.1    0.2    0.7  0.0  0.0   47.8   0   0
id9              0.1  0.2    0.4    1.0  0.0  0.0   24.8   0   1
sd15             0.1  0.1    0.3    0.5  0.0  0.0   84.4   0   0
sd16             0.1  0.1    0.3    0.5  0.0  0.0   93.0   0   0
sd17             0.1  0.1    0.3    0.5  0.0  0.0   79.7   0   0
sd18             0.1  0.1    0.3    0.5  0.0  0.0   95.3   0   0
sd6              0.0  0.0    0.0    0.0  0.0  0.0  109.1   0   0
```

`disk` Name of disk device.
`r/s` Average read operations per second.
`w/s` Average write operations per second.
`Kr/s` Average Kbytes read per second
`Kw/s` Average Kbytes written per second.
`wait` Number of requests outstanding in the device driver queue.
`actv` Number of requests active in the disk hardware queue.
`%w` Occupancy of the wait queue.

| | |
|---|---|
| %b | Occupancy of the active queue—device busy. |
| svc_t | Average service time in milliseconds for a complete disk request. This includes wait time, active queue time, seek rotation, and transfer latency. |
| us | CPU time. |
| sy | System time. |
| wt | Wait for I/O time. |
| id | Idle time. |

**3. Type `sar -d 15 1000 | d2fs.server` to run the `sar -d 15 1000` command through the `sed` script.**

```
server% sar -d 15 1000 | d2fs.server
12:44:17  device   %busy     avque   r+w/s  blks/s avwait  avserv
12:44:18  export/home   0        0.0       0       0     0.0   0.0
          export/home   0        0.0       0       0     0.0   0.0
          export/home   0        0.0       0       0     0.0   0.0
          export/home   0        0.0       0       0     0.0   0.0
          export/home   0        0.0       0       0     0.0   0.0
          id20          0        0.0       0       0     0.0   0.0
          export/home   0        0.0       0       0     0.0   0.0
          export/home   0        0.0       0       0     0.0   0.0
          export/home   0        0.0       0       0     0.0   0.0
          export/home   0        0.0       0       0     0.0   0.0
          export/home   0        0.0       0       0     0.0   0.0
          export/home   0        0.0       0       0     0.0   0.0
          export/home   0        0.0       0       0     0.0   0.0
          export/home   0        0.0       0       0     0.0   0.0
          export/home   0        0.0       0       0     0.0   0.0
          var/mail      0        0.0       0       0     0.0   0.0
          export/home   0        0.0       0       0     0.0   0.0
          sd15          7        0.1       4     127     0.0  17.6
          sd16          6        0.1       3     174     0.0  21.6
          sd17          5        0.0       3     127     0.0  15.5
```

The `sar -d` option reports the activities of disk devices. The `15` means that data is collected every 15 seconds. The `1000` means that data is collected 1000 times.

| | |
|---|---|
| device | Name of the disk device being monitored. |
| %busy | Percentage of time the device spent servicing a transfer request (same as `iostat %b`). |

| | |
|---|---|
| `avque` | Average number of requests outstanding during the monitored period (measured only when the queue was occupied) (same as `iostat actv`). |
| `r+w/s` | Number of read and write transfers to the device, per second (same as `iostat r/s + w/s`). |
| `blks/s` | Number of 512-byte blocks transferred to the device, per second (same as `iostat` 2*(`Kr/s + Kw/s`). |
| `avwait` | Average time, in milliseconds, that transfer requests wait idly in the queue (measured only when the queue is occupied) (same as `iostat wait`). |
| `avserv` | Average time, in milliseconds, for a transfer request to be completed by the device (for disks, this includes seek, rotational latency, and data transfer times). |

4. **For the file systems that are exported by way of NFS, check the** `%b/%busy` **value. If it is more than 30%, check the** `svc_t` **value.**
   The `%b` value, the percentage of time the disk is busy, is returned by the `iostat` command. The `%busy` value, the percentage of time the device spent servicing a transfer request, is returned by the `sar` command. If the `%b` and the `%busy` values are greater than 30 percent, go to step 5. Otherwise, go to the procedure "To collect data on a long-term basis," which follows.

5. **Calculate the** `svc_t/avserv` **value.**
   The `svc_t` value, the average service time in milliseconds, is returned by the `iostat` command. The `avserv` value, the average time (milliseconds) for a transfer request to be completed by the device, is returned by the `sar` command.

   If the `svc_t` value, the average total service time in milliseconds, is more than 40 ms, this disk is taking a long time to respond. An NFS request that involves a disk I/O will be seen as slow by the NFS clients. The NFS response time should be less than 50 ms on average to allow for NFS protocol processing and network transmission time. The disk response should be less than 40 ms.

   The average service time in milliseconds is a function of the disk. If you have fast disks, the average service time should be less than if you have slow disks.

### ▼ To collect data on a long-term basis

- **Uncomment the lines in the user's `sys crontab` file so that `sar` collects the data for one month.**
  This continuously collects performance data and provides you with a history of `sar` results.

**Note –** A few hundred Kbytes will be used at most in `/var/adm/sa`.

```
root# crontab -l sys
#ident"@(#)sys1.592/07/14 SMI"/* SVr4.0 1.2*/
#
# The sys crontab should be used to do performance collection.
# See cron and performance manual pages for details on startup.
0 * * * 0-6 /usr/lib/sa/sa1
20,40 8-17 * * 1-5 /usr/lib/sa/sa1
5 18 * * 1-5 /usr/lib/sa/sa2 -s 8:00 -e 18:01 -i 1200 -A
```

### ▼ If disks are overloaded, spread the load out

- **Use Online: DiskSuite to stripe the file system over multiple disks. Use a Prestoserve write cache to reduce the number of accesses and spread peak access loads out in time.**
  See Section 4.3.4, "Using Online: Disk Suite to Spread Disk Access Load."

### ▼ If you have read-only file systems

- **Increase the buffer cache.**
  See Section 4.7.5, "Adjusting the Buffer Cache: bufhwm."

### ▼ To identify NFS problems, display server statistics

♦ **Type** `nfsstat -s` **at the** `%` **prompt.**
The `-s` option displays server statistics.

```
server% nfsstat -s
Server rpc:
calls      badcalls   nullrecv   badlen     xdrcall
480421     0          0          0          0
Server nfs:
calls      badcalls
480421     2
null       getattr    setattr    root       lookup     readlink   read
95  0%     140354 29% 10782  2%  0  0%      110489 23% 286  0%    63095 13%
wrcache    write      create     remove     rename     link       symlink
0  0%      139865 29% 7188  1%   2140  0%   91  0%     19  0%     231  0%
mkdir      rmdir      readdir    statfs
435  0%    127  0%    2514  1%   2710  1%
```

The server NFS display shows the number of NFS calls received (`calls`) and rejected (`badcalls`), and the counts and percentages for the various calls that were made. The number and percentage of calls returned by the `nfsstat -s` command are described in Table 3-2, later in this chapter.

| | |
|---|---|
| `calls` | Total number of RPC calls received |
| `badcalls` | Total number of calls rejected by the RPC layer (the sum of `badlen` and `xdrcall`) |
| `nullrecv` | Number of times an RPC call was not available when it was thought to be received |
| `badlen` | Number of RPC calls with a length shorter than a minimum-sized RPC call |
| `xdrcall` | Number of RPC calls whose header could not be XDR decoded |

Table 3-1 explains the `nfsstat -s` command output and what actions to take.

*Table 3-1* Description of the `nfsstat -s` Command Output

| If | Then |
|---|---|
| writes > 5% | Install a Prestoserve NFS accelerator (SBus card or NVRAM-NVSIMM) for peak performance. See Section 4.6, "Prestoserve NFS Accelerator." The number of writes, 29% in the previous example, is very high. |
| There are any `badcalls` | `Badcalls` are calls rejected by the RPC layer and are the sum of `badlen` and `xdrcall`. The network may be overloaded. Identify an overloaded network using network interface statistics. |
| `readlink` > 10% of total `lookup` calls on NFS servers | NFS clients are using excessively symbolic links that are on the file systems exported by the server. Replace the symbolic link with a directory. Mount both the underlying file system and the symbolic link's target on the NFS client. See the procedure that follows "If symlink is greater than 10 percent in the output of the nfsstat -s command, eliminate symbolic links." |
| `getattr` > 40% | Increase the client attribute cache using the `actimeo` option. Make sure that the DNLC and inode caches are large. Use `vmstat -s` to determine the percent hit rate (cache hits) for the DNLC and if needed increase `ncsize` in the `/etc/system` file. See the procedure "To show the Directory Name Lookup Cache (DNLC) hit rate," in this chapter and Section 4.7.6, "Directory Name Lookup Cache (DNLC)" in Chapter 4. |

### ▼ If `symlink` is greater than 10 percent in the output of the `nfsstat -s` command, eliminate symbolic links

In this example, `/usr/tools/dist/sun4` is a symbolic link for `/usr/dist/bin`.

1. **Type `rm /usr/dist/bin` to eliminate the symbolic link for `/usr/dist/bin`.**

```
# rm /usr/dist/bin
```

2. **Type** `mkdir /usr/dist/bin` **to make** `/usr/dist/bin` **a directory.**

```
# mkdir /usr/dist/bin
```

**Mount the directories. Type:**

```
client# mount server: /usr/dist/bin
client# mount server: /usr/tools/dist/sun4
client# mount
```

### ▼ To show the Directory Name Lookup Cache (DNLC) hit rate

1. **Type** `vmstat -s`.
   This command returns the hit rate (cache hits).

```
% vmstat -s
... lines omitted
79062 total name lookups (cache hits 94%)
16 toolong
```

2. **If the hit rate is less than 90 percent and there is not a problem with too many longnames, increase the variable,** `ncsize`**, in the** `/etc/system` **file**

```
set ncsize=5000
```

Directory names less than 30 characters long are cached and names that are too long to be cached are also reported.

The default value of `ncsize` is:

`ncsize` (name cache) = 17 * `maxusers` + 90

- For NFS server benchmarks it has been set as high as 16000.
- For `maxusers` = 2048 it would be set at 34906.

For more information on the Directory Name Lookup Cache, see Section 4.7.6, “Directory Name Lookup Cache (DNLC).”

3. **Reboot the system.**

### ▼ If the system has a Prestoserve NFS accelerator, check its state

1. **Type** `/usr/sbin/presto` **at the** `%` **prompt.**
   Verify that it is in the UP state.

```
server% /usr/sbin/presto
state = UP, size = 0xfff80 bytes
statistics interval: 1 day, 23:17:50  (170270 seconds)
write cache efficiency: 65%
All 2 batteries are ok
```

2. **If it is in the DOWN state, type** `presto -u` **at the** `%` **prompt.**

```
server% presto -u
```

3. **If it is in the error state, see the** *Prestoserve User's Guide***.**

*Table 3-2* NFS Operations

| Operation | Function |
|---|---|
| `create` | Create a file system node; may be a file or symbolic link. |
| `statfs` | Get dynamic file system information. |
| `getattr` | Get file / directory attributes such as file type, size, permissions, and access times. |
| `link` | Create a hard link in the remote file system. |
| `lookup` | Search directory for file and return file handle. |
| `mkdir` | Create a directory. |
| `null` | Do nothing. Used for testing and timing of server response. |
| `read` | Read an 8 KByte block of data. |
| `readdir` | Read a directory entry. |
| `readlink` | Follow a symbolic link on the server. |
| `rename` | Change the file's directory name entry. |
| `remove` | Remove a file system node. |
| `rmdir` | Remove a directory. |
| `root` | Retrieve the root of the remote file system (not presently used). |

*Table 3-2* NFS Operations (Continued)

| Operation | Function |
|---|---|
| `setattr` | Change file / directory attributes. |
| `symlink` | Make a symbolic link in a remote file system. |
| `wrcache` | Write an 8 KByte block of data to the remote cache (not presently used). |
| `write` | Write an 8 KByte block of data. |

### 3.2.3 Checking Each Client

The overall tuning process must include client tuning. Sometimes, tuning the client yields more improvement than fixing the server. For example, adding 4 Mbytes of memory to each of 100 clients dramatically decreases the load on an NFS server.

Figure 3-4 illustrates the steps you must follow in sequence to check each client.

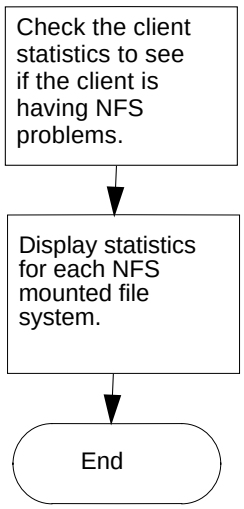



*Figure 3-4* Flow Diagram to Check Each Client

### ▼ To check the client statistics to see if the client is having NFS problems

♦ **Type** `nfsstat -c` **at the** % **prompt.**
Look for errors and retransmits.

```
client % nfsstat -c
Client rpc:
calls      badcalls   retrans    badxids    timeouts   waits      newcreds
384687     1          52         7          52         0          0
badverfs   timers     toobig     nomem      cantsend   bufulocks
0          384        0          0          0          0
Client nfs:
calls      badcalls   clgets     cltoomany
379496     0          379558     0
Version 2: (379599 calls)
null       getattr    setattr    root       lookup     readlink   read
0 0%       178150 46% 614 0%     0 0%       39852 10%  28 0%      89617 23%
wrcache    write      create     remove     rename     link       symlink
0 0%       56078 14%  1183 0%    1175 0%    71 0%      51 0%      0 0%
mkdir      rmdir      readdir    statfs
49 0%      0 0%       987 0%     11744 3%
```

The output shows that there were only 52 retransmits (`retrans` ) and 52 time-outs (`timeout`) out of 384687 calls.

The `nfsstat -c` display shows the following fields:

| | |
|---|---|
| `calls` | Total number of calls sent |
| `badcalls` | Total number of calls rejected by RPC |
| `retrans` | Total number of retransmissions |
| `badxid` | Number of times that a duplicate acknowledgment was received for a single NFS request |
| `timeout` | Number of calls that timed out |
| `wait` | Number of times a call had to wait because no client handle was available |
| `newcred` | Number of times the authentication information had to be refreshed |

Table 3-2, earlier in this chapter, describes the NFS operations. Table 3-3 explains the output of the `nfsstat -c` command and what action to take.

*Table 3-3* Description of the `nfsstat -c` Command Output

| If | Then |
|---|---|
| `retrans` > 5% of the calls | The requests are not reaching the server. |
| `badxid` is approximately equal to `badcalls` | The network is slow. Determine the cause of the slow network. To remedy the problem consider installing a faster network or installing subnets. |
| `badxid` is approximately equal to `timeouts` | Most requests are reaching the server but the server is slower than you expect. Watch expected times using `nfsstat -m`. |
| `badxid` is close to 0 | The network is dropping requests. Reduce `rsize` and `wsize` in the `mount` options. |
| `null` > 0 | A large amount of `null` calls suggests that the automounter is retrying the mount frequently. The timeout values for the mount are too short. Increase the mount timeout parameter, `timeo`, on the automounter command line |

Some of the third party tools you can use for NFS/networks include:

- NetMetrix (Hewlett-Packard)
- SharpShooter (AIM Technology)

### ▼ To display statistics for each NFS mounted file system

♦ **Type** `nfsstat -m` **at the % prompt.**
The statistics include the server name and address, mount flags, current read and write sizes, transmission count, and the timers used for dynamic transmission.

```
client % nfsstat -m
/export/home from server:/export/home
 Flags:
vers=2,hard,intr,dynamic,rsize=8192,wsize=8192,retrans=5
 Lookups: srtt=10 (25ms), dev=4 (20ms), cur=3 (60ms)
 Reads:   srtt=9 (22ms), dev=7 (35ms), cur=4 (80ms)
 Writes:  srtt=7 (17ms), dev=3 (15ms), cur=2 (40ms)
 All:     srtt=11 (27ms), dev=4 (20ms), cur=3 (60ms)
```

| | |
|---|---|
| `srtt` | Smoothed round-trip time |
| `dev` | The estimated deviation |
| `cur` | Current backed-off timeout value |

The numbers in parentheses are the actual times in milliseconds. The other values are unscaled values kept by the operating system kernel. You can ignore the unscaled values. Response times are shown for lookups, reads, writes and a combination of all of these operations (all). Table 3-4 shows the appropriate action for the `nfsstat -m` command.

*Table 3-4* Results of the `nfsstat -m` Command

| If | Then |
|---|---|
| `srtt` > 50 ms | That mount point is slow. Check the network and the server for the disk(s) that provide that mount point. See the steps earlier in this chapter. |
| You have been noticing the message "`NFS server not responding`" | Try increasing the `timeo` parameter, in the `/etc/vfstab` file, to eliminate the messages and to improve performance. Doubling the initial `timeo` parameter value is a good baseline.<br><br>After changing the `timeo` value in the `vfstab` file, invoke the `nfsstat -c` command and observe the `badxid` value returned by the command. Follow the recommendations for the `nfsstat -c` command earlier in this section. |
| `Lookups: cur` > 80 ms | The requests are taking too long to get processed. This indicates a slow network or a slow server. |
| `Reads: cur` > 150 ms | The requests are taking too long to get processed. This indicates a slow network or a slow server. |
| `Writes: cur` > 250 ms | The requests are taking too long to get processed. This indicates a slow network or a slow server. |

# 3

# Configuration Recommendations for NFS Performance

4

This chapter provides configuration recommendations for NFS performance. For troubleshooting tips see Chapter 5, “Troubleshooting.”

## 4.1 Tuning for NFS Performance Improvement

This section covers tuning your NFS server to provide adequate performance during normal operation. Check the following when tuning the system:



Once you have profiled the performance capabilities of your server, begin tuning the system. Tuning an NFS server requires a basic understanding of how networks, disk drives, CPUs, and memory affect performance. Tuning the system also requires determining what parameters to adjust to improve balance.

There are three steps to monitor and tune the server performance:

1. Collect statistics.
   See Chapter 3, “Analyzing NFS Performance.”
2. Identify a constraint or overutilized resource and reconfigure around it.
   See Chapter 3, “Analyzing NFS Performance,” and this chapter for tuning recommendations.
3. Measure the performance gain over a long evaluation period.

This chapter discusses tuning recommendations for:

- Attribute-intensive environments, which are applications or environments where primarily small files (one to two hundred bytes) are accessed. Software development is an example of an attribute-intensive environment.
- Data-intensive environments, which are applications or environments, where primarily large files are accessed. A *large* file can be defined as a file that takes one or more seconds to transfer (roughly 1 Mbyte). CAD or CAE are examples of data-intensive environments.

### 4.1.1 Balancing NFS Server Workload

All NFS processing takes place inside the operating system kernel at a higher priority than user-level tasks. When the NFS load is high, any additional tasks performed by an NFS server will run slowly. For this reason, it is a bad idea to combine databases or time-shared loads on an NFS server.

Non-interactive workloads such as mail delivery and printing (excluding the SPARCprinter or other Sun printers based on the NeWSprint software) are good candidates for using the server for dual purpose (such as NFS and other tasks). If you have lots of spare CPU power and a light NFS load, then interactive work will run normally.

## 4.2 Networks

Make sure that network traffic is well balanced across all client networks and that no network is excessively loaded. If one client network is excessively loaded, watch the NFS traffic on that segment. Identify the hosts that are making the largest demands on the servers. Partition the work load, or move clients from one segment to another.

Simply adding disks to a system does not improve its NFS performance unless the system is truly disk I/O-bound. The network itself is likely to be the constraint as the file server grows, requiring adding more network interfaces to keep the system in balance.

Instead of attempting to move more data blocks over a single network, consider characterizing the amount of data consumed by a typical client and balance the NFS reads and writes over multiple networks.

### 4.2.1 Networking Requirements for Data-Intensive Applications

Data-intensive applications demand relatively few networks. However, the networks must be high-bandwidth (such as FDDI, CDDI, SunATM™, or SunFastEthernet™). For more information on FDDI, see the *FDDI/S3.0 User's Guide*.

If your configuration has either of the following characteristics, then your applications require high-speed networking:

- Your clients require aggregate data rates of more than 1 Mbyte per second.
- More than one client must be able to simultaneously consume 1 Mbyte per second of network bandwidth.

#### ▼ To configure networking when your server's primary application is data-intensive

Following are suggestions to configure networking when your server's primary application is data-intensive. You do not have to perform these suggestions in sequential order.

- **Configure FDDI or another high-speed network.**
  If fiber cabling can't be used for logistical reasons, consider FDDI, CDDI, or SunFastEthernet over twisted-pair implementations. SunATM uses the same size fiber cabling as FDDI.

- **Configure one FDDI ring for each five to seven concurrent fully NFS-active clients (in an NFS sense).**
  Few data-intensive applications make continuous NFS demands. In typical data-intensive EDA and earth-resources applications, this results in 25-40 clients per ring.

  A typical use consists of loading a big block of data which is manipulated then writing the data back. These environments can have very high write percentages due to the step of writing the data back.

- **If your installation has Ethernet cabling, configure one Ethernet for every two active clients; maximum four to six clients per network.**
  This almost always requires using a SPARCcluster 1, SPARCserver 1000, SPARCserver 1000E, SPARCcenter 2000 or 2000E system, since useful communities require many networks.

### 4.2.2 Networking Requirements for Attribute-Intensive Applications

In contrast, most attribute-intensive applications are easily handled with less expensive networks. But, attribute-intensive applications will need many networks. Use lower-speed networking media, such as Ethernet or token ring.

#### ▼ To configure networking when your server's primary application is attribute-intensive

Following are suggestions to configure networking when your server's primary application is attribute-intensive. You do not have to perform these suggestions in sequential order.

- **Configure on Ethernet or token ring.**

- **Configure one Ethernet for eight to ten** *fully active clients***. More than 20 to 25 clients per Ethernet results in severe degradation when many clients are active.**
  As a check, an Ethernet is able to sustain about 250-300 NFS ops/second on the SPECnfs_097 (LADDIS) benchmark, albeit at high collision rates. It is unwise to exceed 200 NFS ops/second on a sustained basis.

- **Configure one token ring network for each ten to fifteen active clients. If necessary, 50 to 80 total clients per network are feasible on token ring networks, due to their superior degradation characteristics under heavy load (compared to Ethernet).**

### 4.2.3 *Networking Requirements for Systems With More Than One Class of Users*

Mixing network types is reasonable. For example, both FDDI and token ring are appropriate for a server that supports both a document imaging application (data-intensive) and a group of PCs running a financial analysis application (most likely attribute-intensive).

The platform you choose is often dictated by the type and number of networks, as they may require many network interface cards.

▼ **To configure networking for servers that have more than one class of users**

- **Mix network types.**

## 4.3 Disk Drives

Disk drive utilization is frequently the tightest constraint in an NFS server. Even a sufficiently large memory configuration may not improve performance if the cache cannot be filled quickly enough from the file systems. Use `iostat` to determine disk utilization. Look at the number of read and write operations per second. See "Checking the NFS Server" in Chapter 3.

Because there is little dependence in the stream of NFS requests, the disk activity generated contains large numbers of random access disk operations. The maximum number of random I/O operations per second ranges from 40-90 per disk.

Driving a single disk at more than 60 percent of its random I/O capacity creates a disk bottleneck.

### ▼ To ease the disk bottleneck

Following are suggestions to ease the disk bottleneck. You do not have to perform these suggestions in sequential order.

- **Balance the I/O load across all disks on the system.**
  If one disk is heavily loaded and others are operating at the low end of their capacity, shuffle directories or frequently accessed files to less busy disks.
- **Partition the file system(s) on the heavily used disk. Spread the file system(s) over several disks.**
  Adding disks provides additional disk capacity and disk I/O bandwidth.
- **If the file system used is read-only by the NFS clients, and contains data that doesn't change constantly, replicate the file system to provide more network-to-disk bandwidth for the clients.**
  See Section 4.3.1, "Replicating File Systems."
- **Size the operating system caches correctly, so that frequently needed file system data may be found in memory.**
  Caches for inodes (file information nodes), file system meta-data such as cylinder group information, and name-to-inode translations must be sufficiently large, or additional disk traffic is created on cache misses. For example, if an NFS client opens a file, that operation generates several name-to-inode translations on the NFS server.

If any operation misses the Directory Name Lookup Cache (DNLC), the server must walk through the disk-based directory entries to locate the appropriate entry name. What would nominally be a memory-based operation degrades into several disk operations. Also, there will not be any cached pages associated with the file.

Disk bandwidth on an NFS server has the greatest effect on NFS client performance. Providing sufficient bandwidth and memory for file system caching is crucial to providing the best possible file server performance. Note that read/write latency is also important.

### 4.3.1 Replicating File Systems

Commonly used file systems, such as the following, are frequently the most heavily used file systems on an NFS server:

- `/usr` directory for diskless clients
- Local tools and libraries
- Popular third party packages
- Read-only source code archives

The best way to improve performance for these file systems is to replicate them. One NFS server is limited by disk bandwidth when handling requests for only one file system. Replicating the data increases the size of the aggregate "pipe" from NFS clients to the data.

Replication *is not* a viable strategy for improving performance with writable data, such as a file system of home directories. Use replication with read-only data.

#### ▼ To replicate file systems

You do not have to perform these guidelines in sequential order.

♦ **Identify the file or file systems to be replicated.**
If several individual files are candidates, consider merging them in a single file system. The potential decrease in performance that arises from combining heavily used files on one disk is more than offset by performance gains through replication.

- **Use a publicly available tool, `nfswatch`, to help identify the most commonly used files and file systems in a group of NFS servers.**
  Table A-1 in the *Appendix* lists performance monitoring tools, including `nfswatch`, and explains how to obtain `nfswatch`.

- **Determine how clients will choose a replica.**
  Specify a server name in the `/etc/vfstab` file to create a permanent binding from NFS client to the server. Alternatively, listing all server names in an automounter map entry allows completely dynamic binding, but may also lead to a client imbalance on some NFS servers. Enforcing "workgroup" partitions, in which groups of clients have their own replicated NFS server, strikes a middle ground between the extremes and often provides the most predictable performance.

- **Choose an update schedule and mechanism, if required. Evaluate what penalties, if any, are involved if users access stale data on a replica that is out-of-date. One possible way of doing this is with the Solaris 2.x JumpStart™ facilities in combination with `cron`.**
  The frequency of change of the read-only data should determine the schedule and the mechanism for distributing the new data. File systems that undergo a complete change in contents, for example, a flat file with historical data that is updated monthly, may be best handled by copying data from the distribution media on each machine, or using a combination of `ufsdump` and `restore`. File systems with few changes can be handled using management tools such as `rdist`.

### 4.3.2 Adding the Cache File System

Adding the cache file system to client mounts provides a local replica for each client. Unlike a replicated server, the cache file system can be used with writable file systems, but performance will degrade as the percent of writes climb. If the percent of writes is too high, the cache file system may decrease NFS performance. The `/etc/vfstab` entry for the cache file system looks like the following:

```
#device          device   mount      FS       fsck   mount    mount
#to mount        to fsck  point      type     pass   at boot  options
server:/usr/dist/cache    /usr/dist  cachefs  3      yes
ro,backfstype=nfs,cachedir=/cache
```

### 4.3.3 Disk Drive Configuration Rules

▼ **To configure disk drives**

***In data-intensive environments***

- **Configure no more than four to five fully active 2.9 Gbyte disks per fast/wide SCSI host adapter.**
  Configure at least three disk drives for every active client on FDDI, one drive for every client on Ethernet or token ring.

***In attribute-intensive environments***

- **Configure no more than four to five fully active 1.05 Gbyte or 535 Mbyte disks per fast SCSI host adapter.**
  Configure at least one disk drive for every two fully active clients (on any type of network.)
- **Configure no more than six to seven 2.9 Gbyte disk drives for each fast/wide SCSI host adapter.**

***General disk drive configuration rules***

- **Configure additional drives on each host adapter without degrading performance (as long as the number of active drives does not exceed SCSI standard guidelines.)**
- **Use Online: Disk Suite to spread disk access load across many disks.**
  See Section 4.3.4, “Using Online: Disk Suite to Spread Disk Access Load.”
- **Use the fastest zones of the disk when possible.**
  See Section 4.3.5, “Using the Optimum Zones of the Disk.”

### 4.3.4 Using Online: Disk Suite to Spread Disk Access Load

A common problem in NFS servers is poor load balancing across disk drives and disk controllers. Balance loads by physical usage, instead of logical usage. Use Online: DiskSuite to spread disk access across disk drives transparently by using its striping and mirroring functions. Disk concatenation accomplishes a minimum amount of load balancing as well but only when disks are relatively full.

If your environment is data-intensive, stripe with a small interlace to improve disk throughput and distribute the service load. Disk striping improves read and write performance for serial applications. Use 64 Kbytes per number of disks in the stripe as a starting point for interlace size.

If your environment is attribute-intensive, where random access dominates disk utilization, the default interlace (one disk cylinder) is most appropriate.

Online: DiskSuite's disk mirroring feature also improves disk access time and reduces disk utilization by providing access to two or three copies of the same data. This is true particularly in environments dominated by read operations. Write operations are normally slower on a mirrored disk since two or three writes must be accomplished for each logical operation requested.

Use the `iostat` and `sar` commands to report disk drive utilization. Attaining even disk utilization usually requires some iterations of monitoring and data reorganization. In addition, usage patterns change over time. A data layout that works well at installation may perform very poorly a year later. For more information on checking disk drive utilization, see "Checking the NFS Server" in Chapter 3.

#### 4.3.4.1 Using Log Based File Systems with Online: Disk Suite 3.0

The combination of the Solaris 2.4 software environment and the Online: Disk Suite 3.0 software supports a log-based extension to the standard UNIX file system which behaves like a disk-based Prestoserve NFS accelerator. In addition to the main file system disk, a small (typically 10 Mbytes) section of disk is used as a sequential log for writes. This speeds up the same kind of operations as a Prestoserve NFS accelerator with two advantages:

- In dual machine high-available configurations, the Prestoserve NFS accelerator cannot be used. The log can be shared so that it can be used.

- After an operating system crash, the `fsck` of the log-based file system involves a sequential read of the log only, so it is almost instantaneous, even on very large file systems.

### 4.3.5 *Using the Optimum Zones of the Disk*

When you are analyzing your disk data layout, consider *zone bit recording*. All of Sun's current disks (except the 207 Mbyte disk) utilize this encoding mechanism which uses the peculiar geometric properties of a spinning disk to pack more data into the parts of the platter closest to its edge. This results in the lower disk addresses (corresponding to the outside cylinders) usually outperforming the inside addresses by 50 percent. You put the data in the lowest-numbered cylinders, since the zone bit recording data layout makes those cylinders the fastest ones.

This margin is most often realized in serial transfer performance, but also affects random access I/O. Data on the outside cylinders (zero) not only moves past the read/write heads more quickly, but the cylinders are also larger. Data will be spread over fewer large cylinders, resulting in fewer and shorter seeks.

## 4.4 Central Processor Units

This section explains how to determine CPU utilization, provides guidelines to configure CPUs in NFS servers, and explains how to increase CPUs in the SPARCcluster 1 system by adding hosts to the system.

### ▼ To determine CPU utilization

♦ **Type `mpstat 30` at the % prompt to get 30 second averages.**

```
zardoz-12% mpstat 30
CPU minf mjf xcal  intr ithr  csw icsw migr smtx  srw syscl  usr sys  wt idl
  0    6   0    0   114   14   25    0    6    3    0    48    1   2  25  72
  1    6   0    0    86   85   50    0    6    3    0    66    1   4  24  71
  2    7   0    0    42   42   31    0    6    3    0    54    1   3  24  72
  3    8   0    0     0    0   33    0    6    4    0    54    1   3  24  72
```

The `mpstat 30` command reports per processor statistics. Each row of the table represents the activity of one processor. The first table summarizes all activities since the system was last booted. Each subsequent table summarizes activity for the preceding interval. All values are rates (events per second).

In the `mpstat` output look at:

- `usr` Percent user time
- `sys` Percent system time
- `wt` Percent wait time
- `idl` Percent idle time

See Table 4-1. The percent system time, `sys`, can be caused by NFS processing.

*Table 4-1* Results of the `mpstat 30` Command

| If | Then |
|---|---|
| `sys` is greater than 50% | There is a lot of NFS processing going on. Add CPU power to improve NFS performance. |
| `wt` (wait time) is high and `idl` (idle time) is low | Check for too few disks or networks. |

Table 4-2 describes guidelines to configure CPUs in NFS servers.

*Table 4-2* Guidelines to Configure CPUs in NFS Servers

| If | Then |
|---|---|
| Your environment is predominantly attribute-intensive, and you have 1 to 3 medium-speed Ethernet or token ring networks | A uniprocessor system is sufficient. For smaller systems, the SPARCserver 5 or SPARCserver 2 systems have sufficient processor power to be used as a server. |
| Your environment is predominantly attribute-intensive, and you have between 4 to 60 medium-speed Ethernet or token ring networks | Use a SPARCserver 10 or SPARCserver 20 system. |
| You have larger attribute-intensive environments and if SBus and disk expansion capacity is sufficient | Use multiprocessor models of the SPARCserver 10 or the SPARCserver 20 systems. |
| You have larger attribute-intensive environments | Use dual-processor systems such as the SPARCserver 10 system Model 512, dual processor SPARCserver 20 system, or dual-processor configurations of the SPARCserver 1000or 1000E system, SPARCcenter 2000 or 2000E system, and SPARCcluster 1 system. Either the 40 MHz/1Mbyte or the 50MHz/2 Mbyte module work well for an NFS work load in the SPARCcenter 2000 system. You will get better performance from the 50 MHz/2Mbyte module. |
| Your environment is data-intensive and you have a high-speed network | Configure 1 SuperSPARC processor per high-speed network (such as FDDI). |
| Your environment is data-intensive and you must use Ethernet due to cabling restrictions | Configure 1 SuperSPARC processor for every 4 Ethernet or token ring networks. |
| Your environment is a pure NFS installation | It is not necessary for you to configure additional processors, beyond the recommended number, on your server(s). |
| Your servers perform tasks in addition to NFS processing | Add additional processors to increase performance significantly. |
| You have a SPARCcluster 1 system | Add additional hosts to the system.<br>This increases the overall throughput by adding memory, disk I/O bandwidth, and additional channels to tape devices, in addition to a new CPU. |
| You have a SPARCcluster 1 system and your NFS load is expected to increase over time | Upgrade from a two-host to a four-host SPARCcluster 1 system to increase overall system capacity. |

## 4.5 Memory

Since NFS is a disk I/O-intensive service, a slow server can be suffering from I/O bottlenecks. Adding memory eliminates the I/O bottleneck by increasing the file system cache size.

The system could be waiting for file system pages, or it may be paging process images to and from the swap device. The latter effect is only a problem if additional services are provided by the system, since NFS service runs entirely in the operating system kernel.

If the swap device is not showing any I/O activity, then all paging is due to file I/O operations from NFS reads, writes, attributes, and lookups.

### ▼ To determine if an NFS server is memory bound

Paging filesystem data from the disk into memory is a more common NFS server performance problem.

1. **Watch the scan rate reported by** `vmstat 30`**.**
   If the scan rate (`sr`, the number of pages scanned) is over 200 pages/second, then the system is short of memory (RAM). The system is trying to find unused pages to be reused and may be reusing pages that should be cached for rereading by NFS clients.

2. **Add memory.**
   Adding memory eliminates repeated reads of the same data and allows the NFS requests to be satisfied out of the page cache of the server. To calculate the memory required for your NFS server, see the section "To calculate memory according to general memory rules," which follows.

The memory capacity required for optimal performance depends on the average working set size of files used on that server. The memory acts as a cache for recently read files. The most efficient cache matches the current working set size as closely as possible.

---

**Note –** Because of this memory caching feature, it is not unusual for free memory in NFS servers to be in the 3-4 Mbyte range (0.5 Mbytes to 1.0 Mbytes for the Solaris 2.4 software environment) if the server has been active for a long time. Such behavior is not only normal, it is desired.

---

Having enough memory allows you to service multiple requests without blocking.

**Note –** The actual files in the working set may change over time. However, the size of the working set may remain relatively constant. NFS creates a *sliding window* of active files, with many files entering and leaving the working set throughout a typical monitoring period.

### ▼ To calculate memory according to general memory rules

- **Virtual memory = RAM (main memory) + Swap space**
- **Calculate the amount of memory according to the five minute rule.**
  That is, memory is sized at 16 Mbytes plus memory to cache the data which will be accessed more often than once in five minutes.

**Note –** This is different from SunOS 4.x operating system where virtual memory = swap space. With the SunOS 4.x operating system, swap space must always be greater than RAM.

### ▼ To calculate memory according to specific memory rules

#### ▼ If your server primarily provides user data for many clients

- **Configure relatively minimal memory.**
  For small communities, this will be 32 Mbytes; for large communities, this will be about 128 Mbytes. In multiprocessor configurations, provide at least 64 Mbytes per processor. Attribute-intensive applications normally benefit slightly more from memory than data-intensive applications.

#### ▼ If your server normally provides temporary file space for applications which utilize those files heavily

- **Configure server memory equal to about 75 percent of the size of the active temporary files in use on the server.**
  For example, if each client's temporary file is about 5 Mbytes, and the server is expected to handle 20 fully active clients, configure:

(20 clients x 5 Mbytes) / 75% = 133 Mbytes of memory
Note that 128 Mbytes is the most appropriate amount of memory easily configured.

### ▼ If your server's primary task is to provide only executable images

- **Configure server memory equal to approximately the combined size of the heavily-used binary files (including libraries).**
  For example, a server expected to provide `/usr/openwin` should have enough memory to cache the X server, CommandTool, `libX11.so`, `libview.so` and `libXt`. This NFS application is considerably different from the more typical `/home`, `/src` or `/data` server in that it normally provides the same files repeatedly to all of its clients; hence is able to effectively cache this data. Clients will not use every page of all of the binaries, which is why it is reasonable to configure only enough to hold the frequently-used programs and libraries. Use the Cachefs on the client, if possible, to reduce the load and RAM needs on the server.

### ▼ If the clients are DOS PCs or Macintoshes

- **Add more RAM cache on the Sun NFS server.**
  These systems do much less caching than UNIX system clients.

## ▼ To configure swap space

- **Configure at least 64 Mbytes virtual memory (RAM plus swap space).**
  For example:

| | |
|---|---|
| 16 Mbytes RAM | 48 Mbytes swap space |
| 32 Mbytes RAM | 32 Mbytes swap space |
| 64 or more Mbytes RAM | No swap space |

  Configure enough swap space to allow the user applications to run.
  For more information, see the manual *Solaris 2.x Administering File Systems.*

## 4.6 Prestoserve NFS Accelerator

Adding a Prestoserve™ NFS accelerator is yet another way to improve NFS performance. The NFS version 2 protocol requires that all writes must be written to stable storage before the operation is replied. The Prestoserve NFS accelerator allows high-speed NVRAM instead of slow disks to satisfy the stable storage requirement. The two types of NVRAM used by the Prestoserve NFS accelerator are the SBus and the NVRAM-NVSIMM.

In cases where you can use either NVRAM hardware, use the NVRAM-NVSIMM as the Prestoserve cache. The NVRAM-NVSIMM and SBus hardware are functionally identical. However, the NVRAM-NVSIMM hardware is slightly more efficient and doesn't use up one of your SBus slots. The NVRAM-NVSIMMs reside in memory and the NVRAM-NVSIMM cache is larger than the SBus hardware.

The NVRAM SBus board contains only a 1 Mbyte cache. The NVRAM-NVSIMM SIMM module contains 2 Mbytes of cache and is supported by the SPARCcenter 2000, SPARCcenter 2000E, SPARCserver 1000, SPARCserver 1000E,SPARCcluster 1, SPARCstation 20, and SPARCstation 10 systems. The SPARCserver 1000 and the SPARCcenter 2000 support up to 8 Mbytes and 16 Mbytes of NVRAM, respectively.

NFS servers can significantly increase performance of NFS write operations by using nonvolatile memory (NVRAM), which is typically much faster than local disk writes. The Prestoserve NFS accelerators take advantage of the fact that the NFS version 2 protocol requires synchronous writes to be written to nonvolatile memory, instead of requiring them to be written directly to disk. As long as the server returns data acknowledged from previous write operations, it can save the data in nonvolatile memory.

Both types of Prestoserve NFS accelerators speed up NFS server performance by:

- Providing faster selection of file systems
- Caching writes for synchronous I/O operations
- Intercepting synchronous write requests to disk and storing the data in nonvolatile memory

### 4.6.1 Adding the SBus Prestoserve NFS Accelerator

The SBus Prestoserve NFS accelerator resides on the SBus. You can add the SBus Prestoserve NFS accelerator to any SBus-based server, except the SPARCserver 1000 system or the SPARCcenter 2000 system. Some systems you can add the SBus Prestoserve NFS accelerator on are:

- SPARCclassic system
- SPARCserver LX system
- SPARCserver 5 system
- SPARCserver 10 system
- SPARCserver 20 system
- SPARCserver 600 series

### 4.6.2 Adding the NVRAM-NVSIMM Prestoserve NFS Accelerator

The NVRAM-NVSIMM Prestoserve NFS accelerator significantly improves the response time seen by NFS clients of heavily loaded or I/O-bound servers. Add the NVRAM-NVSIMM Prestoserve NFS accelerator to the following platforms:

- SPARCserver 10 system
- SPARCserver 20 system
- SPARCcluster 1 system
- SPARCserver 1000 or 1000E system
- SPARCcenter 2000 or 2000E system

## 4.7 Tuning Parameters

This section describes how to set the number of NFS threads. It also covers tuning the main NFS performance-related parameters in the `/etc/system` file. Tune these `/etc/system` parameters carefully, considering your server's physical memory size and kernel architecture type.

**Note –** Arbitrary tuning creates major instability problems, including an inability to boot.

### 4.7.1 Setting the Number of NFS Threads in `/etc/init.d/nfs.server`

It is very important for all NFS server configurations to set the number of NFS threads. Each thread is capable of processing one NFS request. A larger pool of threads allows the server to handle more NFS requests in parallel. The default setting, 16, in the Solaris 2.3 and 2.4 software environments results in less than desired NFS response times. Scale the setting with the number of processors and networks. Increase the number of NFS server threads by editing the invocation of `nfsd` in `/etc/init.d/nfs.server`:

```
/usr/lib/nfs/nfsd -a 64
```

This example specifies that the maximum allocation of demand-based NFS threads is 64.

There are three ways to size the number of NFS threads. Each method results in about the same number if you followed the configuration guidelines in this manual. Extra NFS threads do not cause a problem.

### ▼ To set the number of NFS threads

Take the *maximum* of the following three rules:

- **Use 2 NFS threads for each active client process.**
  A client workstation usually only has one active process. However, a time-shared system that is an NFS client may have many active processes.
- **Use 16 to 32 NFS threads for each CPU.**
  Use roughly 16 for a SPARCclassic or a SPARCstation 5 system. Use 32 NFS threads for a system with a 60 MHz SuperSPARC processor.
- **Use 16 NFS threads for each 10 Mbits of network capacity.**
  For example, is you have FDDI, set the number of NFS threads to 160 `nfsd`s.

## 4.7.2 *Identifying Buffer Sizes and Tuning Variables*

The number of fixed-size tables in the kernel has been reduced in each release of the Solaris software environment. Most are now dynamically sized or are linked to the `maxusers` calculation. For the Solaris 2.2, 2.3, and 2.4 software environments, the extra tuning recommended is to:

- Tune the pager
- Increase the DNLC and inode caches

## 4.7.3 *Using* `/etc/system` *to Modify Kernel Variables*

The `/etc/system` file is read by the operating system kernel at start-up. It configures the search path for loadable operating system kernel modules and allows kernel variables to be set. For more information, see the man page for `system(4)`.

---

**Note –** Be very careful with set commands in `/etc/system`. The commands in `/etc/system` cause automatic patches of the kernel.

---

If your machine will not boot and you suspect a problem with `/etc/system`, use the `boot -a` option. With this option the system prompts (with defaults) for its boot parameters. One of these is the configuration file `/etc/system`.

Either enter the name of a backup copy of the original `/etc/system` file or enter `/dev/null`. Fix the file and immediately reboot the system to make sure it is operating correctly.

### 4.7.4 *Adjusting Cache Size:* `maxusers`

The `maxusers` parameter determines the size of various kernel tables such as the process table. The `maxusers` parameter is set in the `/etc/system` file. For example:

```
set maxusers = 200
```

#### 4.7.4.1 *Solaris 2.2 Software Environment*

In the Solaris 2.2 software environment, `maxusers` is dynamically sized based upon the amount of RAM configured in the system. The automatic configuration of `maxusers` in the Solaris 2.2 software environment is based upon the value of *physmem*, which is the amount of memory (in pages) after the kernel has allocated its own code and initial data space of around 2 Mbytes. The automatic scaling stops when memory exceeds 128 Mbytes.

For systems with 256 Mbytes or more of memory, either set `maxusers` to the generic safe maximum of 200, or leave it and set the individual kernel resources directly. The actual safe maximum level is hardware dependent, and varies depending upon the operating system kernel architecture.

*Table 4-3* `Maxusers` Settings in the Solaris 2.2 Software Environment

| RAM Configuration | Maxusers | Processes | Name Cache |
|---|---|---|---|
| Up to and including 16 Mbytes | 8 | 138 | 226 |
| Up to and including 32 Mbytes | 32 | 522 | 634 |
| Up to and including 64 Mbytes | 40 | 650 | 770 |
| Up to and including 128 Mbytes | 64 | 1034 | 1178 |
| Over 128 Mbytes | 128 | 2058 | 2266 |

**Note –** The operating system kernel uses about 1.5 Mbytes. Therefore, over 130 Mbytes of memory will be required for `maxusers` to be set to 128.

The name cache size is the number used for:

`ncsize` Number of directory name entries in the DNLC

`ufs_ninode` Inactive inode cache size limit

#### 4.7.4.2 *Solaris 2.3 and 2.4 Software Environment*

In the Solaris 2.3 and the Solaris 2.4 software environments, `maxusers` is dynamically sized based upon the amount of RAM configured in the system. The sizing method used for `maxusers` is:

`maxusers` = Mbytes of RAM configured in the system

The number of Mbytes of RAM configured into the system is actually based upon *physmem* which does not include the 2 Mbytes or so that the kernel uses at boot time. The minimum limit is 8 and the maximum automatic limit is 1024, corresponding to systems with 1 Gbyte or more of RAM. It can still be set manually in `/etc/system` but the manual setting is checked and limited to a maximum of 2048. This is a safe level on all kernel architectures, but uses a large amount of operating system kernel memory.

#### 4.7.4.3 *Parameters Derived from* `maxusers`

Table 4-4 describes the default settings for the performance related inode cache and name cache operating system kernel parameters.

*Table 4-4* Default Settings for Inode and Name Cache Parameters

| Kernel Resource | Variable | Default Setting |
|---|---|---|
| Inode Cache | `ufs_ninode` | 17 * `maxusers` + 90 |
| Name Cache | `ncsize` | 17 * `maxusers` + 90 |

### 4.7.5 *Adjusting the Buffer Cache:* `bufhwm`

The `bufhwm` variable, set in the `/etc/system` file, controls the maximum amount of memory allocated to the buffer cache and is specified in Kbytes. The default value of `bufhwm` is 0, which allows up to 2 percent of system memory to be used. This can be increased up to 20 percent. It may need to be increased to 10 percent for a dedicated NFS file server with a relatively small memory system. To prevent the system from running out of the operating system kernel virtual address space, on a larger system it may need to be limited.

The buffer cache is used to cache inode, indirect block, and cylinder group related disk I/O only. Following is an example of a buffer cache ( `bufhwm`) setting in the `/etc/system` file that will allow up to 10 Mbytes of cache. This is the highest value you should set `bufhwm` to.

```
set bufhwm=10240
```

You can monitor the buffer cache using `sar -b` which reports a read (`%rcache`) and a write hit rate (`%wcache`) for the buffer cache.

```
# sar -b 5 10
SunOS hostname 5.2 Generic sun4c    08/06/93
23:43:39 bread/s lread/s %rcache bwrit/s lwrit/s %wcache pread/s pwrit/s
Average        0      25     100       3      22      88       0       0
```

Table 4-5 shows when to increase the buffer cache size, `bufhwm`.

*Table 4-5* When to Increase the Buffer Cache Based on the `sar -b` Output

| If | Then |
|---|---|
| There is a significant (more than 50) number of reads and writes per second and if the read hit rate (`%rcache`) falls below 90% or if the write hit rate (`%wcache`) falls below 65%. | Increase the buffer cache size (`bufhwm`). |

In the previous `sar -b 5 10` command output, the read hit rate (`%rcache`) and the write hit rate (`%wcache`) did not fall below 90 percent or 65 percent respectively.

| | |
|---|---|
| `b` | Checks buffer activity |
| `5` | Time. Every 5 seconds. (must be at least 5 seconds) |
| `10` | Number of times the command gathers statistics |

**Note –** If you increase your buffer cache too much, your server may hang, as other device drivers could suffer from a shortage of operating system kernel virtual memory.

### 4.7.6 Directory Name Lookup Cache (DNLC)

The directory name lookup cache (DNLC) is sized to a default value using `maxusers`. A large cache size (`ncsize`) significantly helps NFS servers, which have lots of clients.

#### ▼ To show the DNLC hit rate (cache hits)

♦ **Type** `vmstat -s`.

```
% vmstat -s
... lines omitted
79062 total name lookups (cache hits 94%)
16 toolong
```

Directory names less than 30 characters long are cached and names that are too long to be cached are reported as well. A cache miss means that a disk I/O may be needed to read the directory when traversing the path name components to get to a file. A hit rate of much less than 90 percent needs attention.

Cache hit rates can significantly affect NFS performance. `getattr`, `setattr` and `lookup` usually represent greater than 50 percent of all NFS calls. If the requested information isn't in cache, the request will generate a disk operation, resulting in a performance penalty as significant as that of a `read` or `write` request. The only limit to the size of the DNLC cache is available kernel memory.

Table 4-6 shows when to tune the variable, `ncsize`, based on the output of the `vmstat -s` command.

*Table 4-6* When to tune the Variable, `ncsize` Based on the `vmstat -s` Output

| If | Then |
|---|---|
| The hit rate (cache hits) is less than 90 % and there is no problem with too many longnames | Tune the variable, `ncsize`. See the procedure "To reset ncsize" which follows. |

The variable `ncsize` refers to the size of the DNLC in terms of the number of name and vnode translations that can be cached. Each DNLC entry causes about 50 bytes of extra kernel memory to be used.

### ▼ To reset `ncsize`

1. **Set `ncsize` in the `/etc/system` file to values higher than default (based on `maxusers`.)**
   As an initial guideline, since dedicated NFS servers do not need a lot of RAM, `maxusers` will be low and the DNLC will be small, so double its size.

```
set ncsize=5000
```

   The default value of `ncsize` is:

   `ncsize` (name cache) = 17 * `maxusers` + 90

   - For NFS server benchmarks it has been set as high as 16000.
   - For `maxusers = 2048` it would be set at 34906.

2. **Reboot the system.**

See Section 4.7.7, "Increasing the Inode Cache,"which follows.

### 4.7.7 Increasing the Inode Cache

A memory resident inode is used whenever an operation is performed on an entity in the file system. The inode read from disk is cached in case it is needed again. The maximum number of active and inactive inodes that the system will cache is set by `ufs_ninode`.

The inodes are kept on a linked list, rather than a fixed-size table. It is possible that the number of open files in the system can cause the number of active inodes to exceed the limit. Raising the limit allows *inactive* inodes to be cached in the Solaris 2.3 software environment in case they are needed again. In the Solaris 2.4 software environment, the `ufs_ninode` count applies only to *inactive* inodes.

Every entry in the DNLC cache points to an entry in the inode cache, so both caches should be sized together. The inode cache should be at least as big as the DNLC cache. For best performance, it should be twice the DNLC in the Solaris 2.2 and 2.3 software environments and the same size in the Solaris 2.4 software environment.

Since it is just a limit, `ufs_ninode` can be tweaked with `adb` on a running system with immediate effect. The only upper limit is the amount of kernel memory used by the inodes. The tested upper limit corresponds to `maxusers = 2048` which is the same as `ncsize` at 34906.

Use `sar -k` to report the size of the kernel memory allocation. Each inode uses 512 bytes of kernel memory from the lg_mem pool in the Solaris 2.2 and 2.3 software environments. In the Solaris 2.4 software environment, each inode uses 300 bytes of kernel memory from the lg_mem pool.

#### ▼ To increase the inode cache

##### ▼ If the inode cache hit rate is below 90 percent or if the DNLC requires tuning for local disk file I/O workloads

1. **Increase the size of the inode cache.**
   Change the variable `ufs_ninode` in your `/etc/system` file to the same size as the DNLC (`ncsize`). For example, for the Solaris 2.4 software environment:

```
set ufs_ninode=5000
```

For the Solaris operating system releases prior to the Solaris 2.4 software environment:

```
set ufs_ninode=10000
```

The default value of the inode cache is the same as that for `ncsize`:

`ufs_ninode` (default value) = 17 * `maxusers` + 90.

**Caution –** Do not set `ufs_ninode` less than `ncsize`.

The Solaris 2.x software environment versions, up to the Solaris 2.3 software environment, run more efficiently with the `ufs_ninode` set to twice `ncsize`. In the Solaris 2.4 software environment, `ufs_ninode` now limits only the number of inactive inodes, rather than the total number of active and inactive inodes.

2. **Reboot the system.**

### *4.7.8 Increasing Read Throughput*

If you are using NFS over a high speed network, such as FDDI, SunFastEthernet, or SunATM, you will obtain better read throughput by increasing the number of read-aheads on the NFS client. The read-ahead functionality is new for the Solaris 2.4 software environment.

Increasing read-aheads is *not* recommended if the:

- Client is very short of RAM
- Network is very busy
- File accesses are randomly distributed

When free memory is low, read-ahead will not be performed.

The read-ahead is set to 1 block by default (8 Kbytes). For example, a read-ahead set to 2 blocks fetches an additional 16 Kbytes from a file while you are reading the first 8 Kbytes from the file. In other words, the read-ahead stays one step ahead of you and fetches information in 8 Kbyte increments to stay ahead of information you need.

The read throughput stops getting faster after setting the number of read-aheads to 6 or 7 blocks. When setting the number of read-aheads, throughput does not usually increase with more than 8 read-aheads (8 blocks)

### ▼ To increase the number of read-aheads in the Solaris 2.4 software environment

1. **Add the following line to `/etc/system` on the NFS client.**
   For example:

```
set nfs:nfs_nra=2
```

2. **Reboot the system for the read-ahead value to take effect.**

# Troubleshooting 5

Table 5-1 lists actions to perform when you encounter a tuning problem.

*Table 5-1* Troubleshooting Actions

| Command/Tool | If the Command Output Is | Then |
|---|---|---|
| `netstat -i` | `Collis+Ierrs+Oerrs/Ipkts + Opkts` > 2% | Check Ethernet hardware. |
| `netstat -i` | `Collis/Opkts` > 10% | Add Ethernet interface and distribute client load. |
| `netsat -i` | `Ierrs/Ipks` > 25% | High input error rate. The host may be dropping packets. To compensate for bandwidth-limited network hardware: reduce the packet size, set the read buffer size, `rsize`, and/or the write buffer size, `wsize`, when using mount or in the `/etc/vfstab` file to 2048. See the procedure "To find the number of packets and collisions/errors on each network" in Chapter 3, "Analyzing NFS Performance." |
| `nfsstat -s` | `readlink` > 10% | Replace symbolic links with mount points. |
| `nfsstat -s` | writes > 5% | Install a Prestoserve NFS accelerator (SBus card or NVRAM-NVSIMM) for peak performance. See Section 4.6, "Prestoserve NFS Accelerator." |

*Table 5-1* Troubleshooting Actions *(Continued)*

| Command/Tool | If the Command Output Is | Then |
|---|---|---|
| `nfsstat -s` | There are any `badcalls`. | The network may be overloaded. Identify an overloaded network using network interface statistics. |
| `nfsstat -s` | `getattr` > 40% | Increase the client attribute cache using the `actimeo` option. Make sure that the DNLC and inode caches are large. Use `vmstat -s` to determine the percent hit rate (cache hits) for the DNLC and if needed increase `ncsize` in the `/etc/system` file. See Section 4.7.6, "Directory Name Lookup Cache (DNLC)" in Chapter 4, "Configuration Recommendations for NFS Performance." |
| `vmstat -s` | hit rate (cache hits) < 90% | Increase `ncsize` in the `/etc/system` file. |
| Ethernet monitor, for example: SunNet Manager, SharpShooter (AIM Technology), NetMetrix (Hewlett-Packard) | load > 35% | Add Ethernet interface and distribute client load. |

Table 5-2, Table 5-3, and Table 5-4 show potential bottlenecks by type and their solutions.

*Table 5-2* Potential Client Bottlenecks

| Symptom(s) | Command/Tool | Cause | Solution |
|---|---|---|---|
| `NFS server` *hostname* `not responding` or slow response to commands or when using NFS-mounted directories. | `nfsstat` | User's path variable | List directories on local file systems first, critical directories on remote file systems second, and then the rest of the remote file systems. |
| `NFS server` *hostname* `not responding` or slow response to commands or when using NFS-mounted directories. | `nfsstat` | Running executable from an NFS mounted file system | Copy the application locally (if used often). |
| `NFS server` *hostname* `not responding;` `badxid` >5% of total calls and `badxid` = `timeout`. | `nfsstat -rc` | Client times out before server responds | Check for server bottleneck. If server's response time isn't improved, increase the `timeo` parameter in the `/etc/vfstab` file of clients. Try increasing `timeo` to 25, 50, 100, 200 (tenths of seconds). Wait one day between modifications and check to see if number of time-outs is decreasing. |
| `badxid` = 0. | `nfsstat -rc` | Slow network | Increase `rsize` and `wsize` in the `/etc/vfstab` file. Check interconnection devices (bridges, routers, gateways). |

*Table 5-3* Potential Server Bottlenecks

| Symptom(s) | Command/Tool | Cause | Solution |
|---|---|---|---|
| `NFS server` *hostname* `not responding.` | `vmstat -s` or `iostat` | Cache hit rate is <90%. | Adjust the suggested parameters for DNLC, then run to see if the symptom disappears. If not, reset the parameters for DNLC. Adjust the parameters for the buffer cache, then the inode cache, following the same procedure as for the DNLC. |
| `NFS server` *hostname* `not responding.` | `netstat -m` *or* `nfsstat` | Server not keeping up with request arrival rate. | Check network. If the problem is not network, add appropriate Prestoserve NFS accelerator, or upgrade the server. |
| High I/O wait time or CPU idle time. Slow disk access times or `NFS server` *hostname* `not responding.` | `iostat -x` | I/O load not balanced across disks. The `svc_t` value is greater than 40 ms. | Take a large sample (~2 weeks). Balance the load across disks; add disks as necessary. Add a Prestoserve NFS accelerator for synchronous writes. To reduce disk and network traffic, use `tmpfs` for `/tmp` for both server and clients. Measure system cache efficiencies. Balance load across disks; add disks as necessary. |
| Slow response when accessing remote files. | `netstat -s` or `snoop` | Ethernet interface dropping packets. | If retransmissions are indicated, increase buffer size. For information on how to use snoop, see Section A.2.1, “Snoop.” |

*Table 5-4* Potential Network-Related Bottlenecks

| Symptoms | Command/Tool | Cause | Solution |
|---|---|---|---|
| Poor response time when accessing directories mounted on different subnets or `NFS server` *hostname* `not responding.` | `netstat -rs` | NFS requests being routed | Keep clients on subnet directly connected to server. |
| Poor response time when accessing directories mounted on different subnets or `NFS server` *hostname* `not responding.` | `nfsstat` | Dropped packets | Make protocol queues deeper. |
| Poor response time when accessing directories mounted on different subnets or `NFS server` *hostname* `not responding.` | `netstat -s` shows incomplete or bad headers, bad data length fields, bad checksums. | Network problems | Check network hardware. |
| Poor response time when accessing directories mounted on different subnets or `NFS server` *hostname* `not responding.` The sum of input and output packets per second for an interface is over 600 per second. | `netstat -i` | Network overloaded | The network segment is very busy. If this is a recurring problem, consider adding another (`le`) network interface. |
| Network interface collisions are over 120 per second. | `netstat -i` | Network overloaded | Reduce the number of machines on the network or check network hardware. |
| Poor response time when accessing directories mounted on different subnets or `NFS server` *hostname* `not responding.` | `netstat -i` | High packet collision rate (`Collis/Opkts` >.10) | If corrupted packets, may be due to corrupted MUX box; use Network General Sniffer or another protocol analyzer to find cause. Check for overloaded network. If there are too many nodes, create another subnet. Check network hardware; could be bad tap, transceiver, hub on 10base-T. Check cable length and termination. |

# 5

# *Using NFS Performance-Monitoring and Benchmarking Tools*

A

This appendix discusses tools that enable you to monitor NFS and network performance on your server. These tools generate information that may be used in tuning to improve performance. See Chapter 3, “Analyzing NFS Performance,” and Chapter 4, “Configuration Recommendations for NFS Performance.”

For more detailed information about these tools, refer to their `man` pages (where applicable) and *SunOS 5.x Administering Security, Performance, and Accounting*. For third-party tools, refer to the product documentation.

This chapter also describes LADDIS, a vendor-neutral NFS file server benchmarking tool.

# A

## A.1 NFS Monitoring Tools

Table A-1 describes the tools that can be used to monitor NFS operations and performance.

*Table A-1* NFS Operations and Performance-Monitoring Tools

| Tool | Function |
|---|---|
| `iostat` | Reports I/O statistics, among them disk I/O activity. |
| `nfsstat` | Reports NFS statistics: NFS and RPC (Remote Procedure Call) interfaces to the kernel. Can also be used to initialize the statistical information. |
| `nfswatch` | Shows NFS transactions classified by file system. `nfswatch` is a public domain tool with source code available on `uunet.uu.net` under the directory `usr/spool/ftp/networking`. The file name is `nfswatch3.0.tar.Z`. |
| `sar` | Reports system activity, such as CPU utilization, buffer activity, disk and tape device activity. |
| SharpShooter (AIM Technology) | Pinpoints bottlenecks, balances NFS load across clients and servers. Shows effect of distributed applications and balances network traffic across servers. Accounts for disk usage by user or group. |
| `vmstat` | Reports virtual memory statistics, among them, disk activity. |

## A.2 Network Monitoring Tools

Table A-2 describes the tools that can be used to monitor network performance as it relates to NFS.

*Table A-2* Network Monitoring Tools

| Tool | Function |
|---|---|
| `snoop` | Displays information about specified packets on Ethernet. |
| `netstat` | Displays the contents of network-related data structures. |
| `ping` | Sends ICMP ECHO_REQUEST packets to network hosts. |
| NetMetrix Load Monitor | Allows network load monitoring real time or over time, and characterization of load in terms of time, source, destination, protocol and packet size. |
| SunNet Manager | Performs network device monitoring and troubleshooting. SunNet Manager is a management and monitoring tool. |
| LAN analyzers: Network General Sniffer, Novell/Excelan Lanalyzer | Performs packet analysis. |

### A.2.1 Snoop

The `snoop` command is part of the Solaris 2.x software environment. The `snoop` command must run by `root` (#) to capture packets in promiscuous mode. In order to capture packets in non-promiscuous mode or to analyze packets from a capture file, you do not need to be `root`.

In promiscuous mode, the interface turns off its filter and lets you see all packets on the subnet, whether or not they are addressed to your system. You can passively observe other packets not destined for your system. Promiscuous mode is limited to `root`.

Using the snoop command turns a Sun system into a network sniffer, able to detect network problems. The snoop command also captures a certain number of network packets, allows you to trace the calls from each client to each server, and displays the contents of the packets. You can also save the contents of the packets to a file to inspect later on.

Some of the features of the snoop command are:

- Logs or displays packets selectively
- Provides accurate timestamps for checking network RPC (for example, NIS) response time
- Formats packets and protocol information in a user-friendly manner

The snoop command can display packets in a single-line summary or in expanded form. In summary form, only the data pertaining to the highest level protocol is displayed. For example, an NFS packet will have only NFS information displayed. The underlying RPC (Remote Procedure Call), UDP (User Datagram Protocol), IP (Internet Protocol), and network frame information is suppressed, but can be displayed if you choose either of the verbose (`-v` or `-V`) options.

The snoop command uses both the packet filter and buffer modules of the Data Link Provider Interface (DLPI) to provide efficient capture of packets transmitted to or received from the network. To view or capture all traffic between any two systems, run snoop on a third system.

The snoop command is a useful tool if you are considering subnetting, since it is a packet analysis tool. The output of the snoop command can be used to drive scripts that accumulate load statistics. The program is capable of breaking the packet header out in order to debug it, and to see what might be the source of incompatibility difficulties.

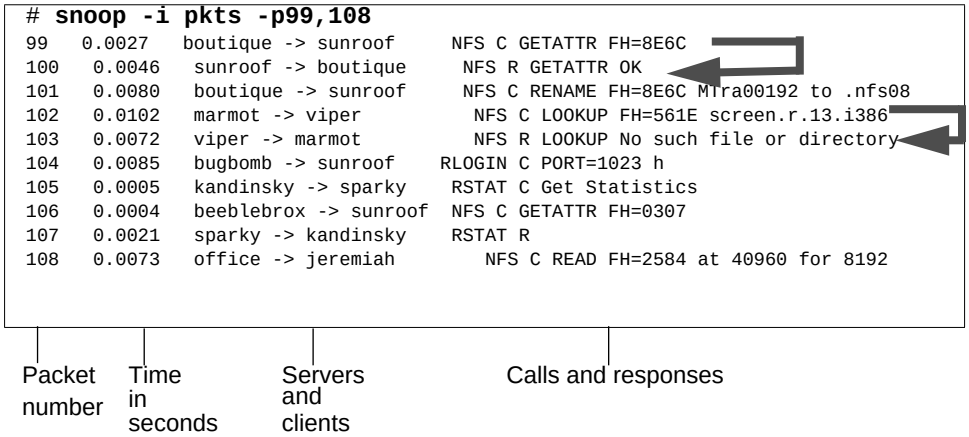
### A.2.1.1 Snoop Examples

Following are some examples of how to use snoop.

#### ▼ To look at selected packets in a capture file, pkts

The statistics show which client is making a read request, and the left-hand column shows the time in seconds with a resolution of about 4 microseconds. Calls and corresponding responses are shown by the arrows, which do not appear on the actual screen.

When a read or write request is made, be sure the server doesn't time-out. If it does, the client has to send again, and the client's IP code will break up the write block into smaller UDP blocks. The default write time is .07 seconds. The time-out factor is a tunable parameter in the mount command.




`-i pkts` Displays packets previously captured in the file, `pkts`.

`-p99,108` Selects packets 99 through 108 to be displayed from a capture file. The first number 99, is the first packet to be captured. The last number, 108, is the last packet to be captured. The first packet in a capture file is packet 1.

### ▼ To obtain more detailed information on a packet

```
# snoop -i pkts -v 101
#
```

The command `snoop -i pkts -v 101` obtains more detailed information on packet 101.

| | |
|---|---|
| `-i pkts` | Displays packets previously captured in the file, `pkts`. |
| `-v` | Verbose mode. Prints packet headers in detail for packet 101. Use this option only when you need information on selected packets. |

The output of the `snoop -i pkts -v 101` command follows showing the Ethernet, IP, UDP, RPC, and NFS headers captured in `snoop`.

```
ETHER:  ----- Ether Header -----
         ETHER:
         ETHER:  Packet 101 arrived at 16:09:53.59
         ETHER:  Packet size = 210 bytes
         ETHER:  Destination = 8:0:20:1:3d:94, Sun
        ETHER:  Source      = 8:0:69:1:5f:e,  Silicon Graphics
         ETHER:  Ethertype = 0800 (IP)
         ETHER:
         IP:   ----- IP Header -----
         IP:
         IP:   Version = 4, header length = 20 bytes
         IP:   Type of service = 00
         IP:         ..0. .... = routine
         IP:         ...0 .... = normal delay
         IP:         .... 0... = normal throughput
         IP:         .... .0.. = normal reliability
         IP:   Total length = 196 bytes
         IP:   Identification 19846
         IP:   Flags = 0X
         IP:   .0.. .... = may fragment
         IP:   ..0. .... = more fragments
         IP:   Fragment offset = 0 bytes
         IP:   Time to live = 255 seconds/hops
         IP:   Protocol = 17 (UDP)
         IP:   Header checksum = 18DC
         IP:   Source address = 129.144.40.222, boutique
         IP:   Destination address = 129.144.40.200, sunroof
         IP:
         UDP:  ----- UDP Header -----
         UDP:
         UDP:  Source port = 1023
         UDP:  Destination port = 2049 (Sun RPC)
         UDP:  Length = 176
         UDP:  Checksum = 0
         UDP:
         RPC:  ----- SUN RPC Header -----
         RPC:
         RPC:  Transaction id = 665905
         RPC:  Type = 0 (Call)
         RPC:  RPC version = 2
        RPC:  Program = 100003 (NFS), version = 2, procedure = 1
         RPC:  Credentials: Flavor = 1 (Unix), len = 32 bytes
```

```
RPC:      Time = 06-Mar-90 07:26:58
          RPC:      Hostname = boutique
          RPC:      Uid = 0, Gid = 1
          RPC:      Groups = 1
          RPC:   Verifier    : Flavor = 0 (None), len = 0 bytes
          RPC:
          NFS:   ----- SUN NFS -----
          NFS:
          NFS:   Proc = 11 (Rename)
          NFS:   File handle = 000016430000000100080000305A1C47
          NFS:                 597A00000008000020463l4AFC450000
          NFS:   File name = MTra00192
          NFS:   File handle = 000016430000000100080000305A1C47
          NFS:                 597A00000008000020463l4AFC450000
          NFS:   File name = .nfs08
          NFS:
```

## ▼ To view NFS packets

```
# snoop -i pkts rpc nfs and sunroof and boutique
1   0.0000   boutique -> sunroof    NFS C GETATTR FH=8E6C
2   0.0046    sunroof -> boutique   NFS R GETATTR OK
3   0.0080  boutique -> sunroof   NFS C RENAME FH=8E6C MTra00192 to .nfs08
```

This example views the NFS packets between the systems `sunroof` and `boutique`. Calls and corresponding responses are shown by arrows which do not appear on the actual screen.

`-i pkts` Displays packets previously captured in the file, `pkts`.

`rpc nfs` Displays packets for an RPC call or reply packet for the NFS protocol. The option following `nfs` is the name of an RPC protocol from `/etc/rpc` or a program number.

`and` Performs a logical and operation between two boolean values. For example, `sunroof boutique` is the same as `sunroof and boutique`.

### ▼ To save packets to a new capture file

```
# snoop -i pkts -o pkts.nfs rpc nfs sunroof boutique
#
```

`-i pkts` Displays packets previously captured in the file, `pkts`.

`-o pkts.nfs` Saves the displayed packets in the output file, `pkts.nfs`.

`rpc nfs` Displays packets for an RPC call or reply packet for the NFS protocol. The option following `nfs` is the name of an RPC protocol from `/etc/rpc` or a program number.

See the `snoop man` page for additional details on options used with the `snoop` command and additional information about using `snoop`.

## A.3 LADDIS

SPEC SFS 1 (0.97 LADDIS) is a vendor-neutral NFS file server benchmark incorporated by SPEC as a member of the System-level File Server (SFS) Benchmark Suite. Two reference points are considered when reporting 097.LADDIS.

- NFS operation throughput
  The peak number of NFS operations the target server can complete in a given number of milliseconds. The larger the number of operations an NFS server can support, the more users it can serve.

- Response time
  The average time needed for an NFS client to receive a reply from a target server in response to an NFS request. The response time of an NFS server is the client's perception of how fast the server is.

LADDIS is designed so that its workload can be incrementally increased until the target server performance falls below a certain level. That point is defined as an average response time exceeding 50ms. This restriction is applied when deriving SPECnfs_A93, the maximum throughput in NFS operations per second for which the response time does not exceed 50 ms.

As long as throughput continues to increase with workload, the throughput figure at 50 ms is reported. In many cases, throughput will start to fall off at a response time below the 50 ms limit. In these cases, the tables in this chapter report the response time at the point of maximum throughput.

The SPEC SFS 1 (0.97 LADDIS) benchmark is a synthetic NFS workload based on an application abstraction, an NFS operation mix, and an NFS operation request rate. The workload generated by the benchmark emulates an intensive software development environment at the NFS protocol level. The LADDIS benchmark makes direct RPC calls to the server, eliminating any variation in NFS client implementation. This makes it easier to control the operation mix and workload, especially for comparing results between vendors. However, this also hides the benefits of special client implementations, such as the cache file system client implemented in the Solaris 2.3 and 2.4 software environments.

Table A-3 summarizes the percentage mix of NFS operations. These percentages indicate the relative number of calls made to each operation.

*Table A-3* NFS Operations Mix By Call

| NFS Operation | Percent Mix |
|---|---|
| Lookup | 34 |
| Read | 22 |
| Write | 15 |
| GetAttr | 13 |
| ReadLink | 8 |
| ReadDir | 3 |
| Create | 2 |
| Remove | 1 |
| Statfs | 1 |
| SetAttr | 1 |

The LADDIS benchmark for NFS file systems uses an operation mix that is 15 percent write operations. If your NFS clients generate only one to two percent write operations, LADDIS underestimates your performance. The greater the similarity between the operation mixes, the more reliable SPECnfs_A93 will be as a reference.

To run the benchmark requires the server being benchmarked, at least two NFS clients (the NFS load generators), and one or more isolated networks. The ability to support multiple networks is important because a single network may saturate before the server maximum performance point is reached. One client is designated as the LADDIS Prime Load Generator. The prime load generator controls the execution of the LADDIS load generating code on all load generating clients. It typically controls the benchmark, also. In this capacity, it is responsible for collecting throughput and response time data at each of the workload points and for generating results.

To improve response time, you can configure your NFS server with the NVRAM-NVSIMM Prestoserve NFS accelerator. NVSIMMs provide storage directly in the high-speed memory subsystem. Using NVSIMMs results in considerably lower latency and reduces the number of disks required to attain a given level of performance.

Since there are extra data copies in and out of the NVSIMM, the ultimate peak throughput is reduced. Because NFS loads rarely sustain peak throughput, the better response time using the NVSIMMs is preferable. For information on the Prestoserve NFS accelerator, see Section 4.6, “Prestoserve NFS Accelerator.”

### *A.3.1 Interpreting LADDIS Results*

The two factors to consider when analyzing LADDIS results are throughput and response time. Although response time increases with throughput, a well-designed NFS server delivers a range of throughput results while maintaining a more or less constant response time.

Figure A-1 illustrates an idealized LADDIS baseline result. At point A, there is little load on the system. Response time is limited by the hardware characteristics of the I/O subsystem. At point D, the LADDIS benchmark reports the SPECnfs_A93 single figure of merit. As you can see, between points B and C, workload increases about five times without impacting client performance.

The closer a NFS server LADDIS benchmark graph of average NFS response time versus NFS throughput resembles Figure A-1, the better the NFS server can meet the average performance requirements and handle burst activity.

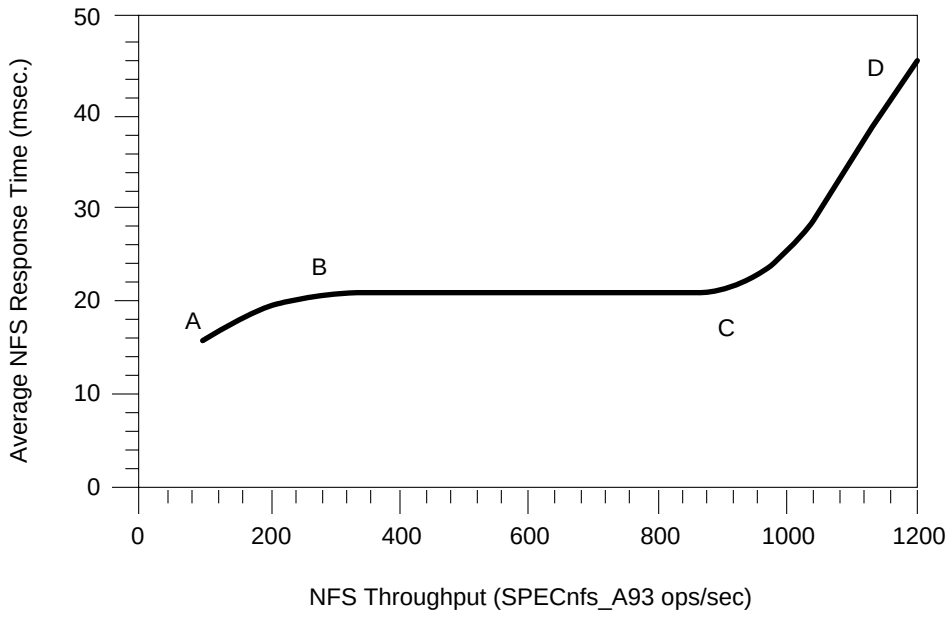



*Figure A-1* Idealized LADDIS Baseline Result, Case 1

Figure A-2 shows a second baseline case of average NFS response time vs. NFS throughput. Point C in Figure A-1 shows a response time of 20 milliseconds. However, point C in Figure A-2 shows a response time of 35 milliseconds.

Compared to case 1 (Figure A-1), a case 2 client (Figure A-2) can experience an extra 15 second delay for each 1000 NFS operations transacted on an NFS server throughput rate of 1000 NFSops. For example, when executing a long listing of the `ls` command (`ls -l`) on a NFS mounted file system can easily request over 1000 NFS operations. Both cases report the same LADDIS result at point D.

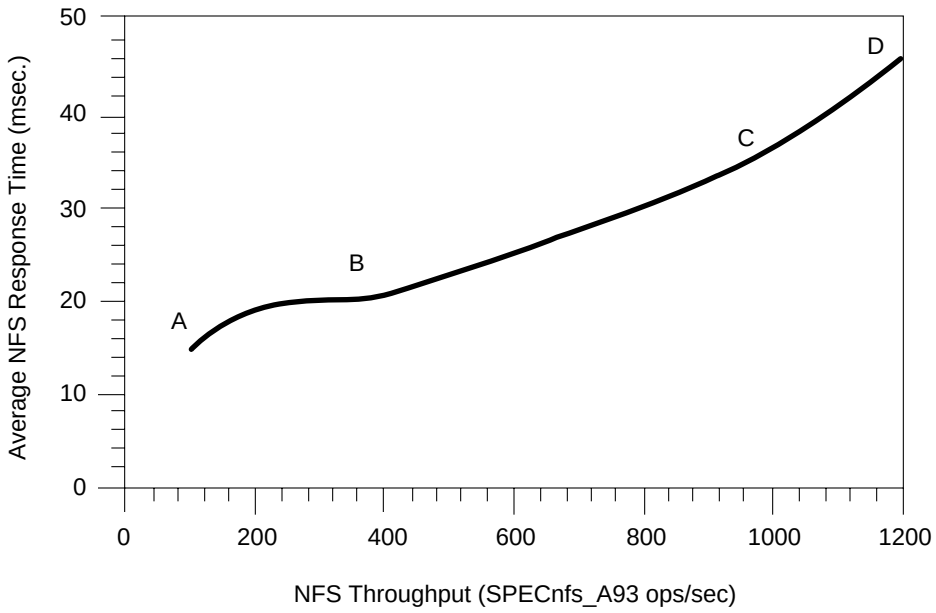



*Figure A-2* Idealized LADDIS Baseline Result, Case 2

# *Glossary*

**Attribute-intensive**

Referring to applications or environments where primarily small files (one- to two-hundred bytes) are accessed. Software development is an example of an attribute-intensive environment.

**Bandwidth**

The difference in frequency between the highest and lowest frequency in a band. This is the peak that cannot be exceeded.

**Biod**

Asynchronous-block I/O daemon.

**Capacity planning**

Planning for and configuring in such a way as to allow for future use of a server or system.

**Client demand**

The request for services generated by server clients. *See also* load.

**Configuration**

The number and relationships of internal and subsystem components (for instance, disk drives and network interfaces).

**Configuration estimation**

Estimating the server or system configuration that will produce the best performance for the current and anticipated tasks to be performed by the server.

**Data-intensive**

Referring to applications or environments, where primarily large files are accessed. A *large* file might be defined as a file that takes one or more seconds to transfer (roughly 512 Kbytes to 1 Mbyte). CAD or CAE are examples of data-intensive environments.

**Fully-active client**

An NFS client that is performing NFS operations, such as reading in a file.

**IP**

Internet protocol

**Latency**

*See* operation latency.

**Load**

The request for services generated by server clients.

**Load estimation**

A component task of configuration estimation. Calculation of the normal work load likely to occur on the server with current and near-term projection of frequency and types of (in this case) NFS tasks. These statistics are used in making decisions about server configuration.

**Operation latency**

The elapsed time between issuing a request and receiving the response.

**Throughput**

The number of NFS operations (for instance, `write` or `getattr`) performed in a time period, usually one second.

**Tuning**

A task or tasks executed with the purpose of improving performance.

**UDP**

User Datagram Protocol. A transport protocol in the Internet suite of protocols. UDP, like TCP, uses IP for delivery. However, unlike TCP, UDP provides for exchange of datagrams without acknowledgments or guaranteed delivery.

# *Index*

## Symbols



## Numerics



## A



## B



## C

## D



## E



## F

## O



## P



## R

## S

## T



## U



## V



## W



## Z