Networks and File Servers: A Performance Tuning Guide

Sun Microsystems, Inc.
December, 1990

Table of Contents

List of Figures

Chapter 1. Performance in Distributed Computing Environments

Several factors contribute to client and server performance in a distributed computing environment. This guide will expand your understanding of these factors, suggest methodologies for evaluating your configuration and current performance bottlenecks, and help you improve your overall network computing performance. Special attention will be paid to accelerating server performance with the Sun Prestoserve NFS solution. Sun documentation and technical reference materials complement this guide and will complete your understanding of Sun products and their optimum use.

Compared with traditional computing models, the client–server computing model offers several advantages:

- Predictable interactive performance for desktop applications
- Simplified administration and more control of data
- The ability to easily share expensive resources such as backup devices and printers.
- The ability to expand as computing requirements grow

File servers are an integral part of the client–server model. They efficiently manage large amounts of stored data and eliminate data redundancy on the network. On–line documentation, image and file libraries, and application binaries need not be stored on every client workstation. High–availability features like disk–mirroring can be implemented selectively and economically but still benefit all of the workstations that require uninterrupted access to important resources.

In addition to the many benefits, a network of clients and servers poses a challenge for optimizing performance. Although one NFS client–server pair that is not hindered in any way will typically operate at 85 to 90 percent of its maximum level of performance, both hindrances and unforeseen interactions between systems are common. Typical NFS networks could perform much better than they do. This paper reviews each of the major components of a distributed environment: NFS, the Sun Prestoserve accelerator (for improving NFS write performance), network links and hardware, and servers. Suggestions are presented for measuring performance, eliminating bottlenecks, troubleshooting failures, and fine tuning configurations for optimized behavior and throughput. Organized by topic, you can read the paper in the presented order, or turn directly to the section addressing your particular area of interest or trouble.

Chapter 2. The Network File System (NFS)

As performance leaps forward and prices drop, the number of client workstations continues to explode at most sites. Servers are being configured with more clients, and powerful clients more severely load the servers. Limiting the number of clients per server can prevent a total overload of a server and ensure adequate performance for every client workstation. The following Network File System (NFS) description examines areas in which NFS performance is constrained.

2.1. An Overview of NFS

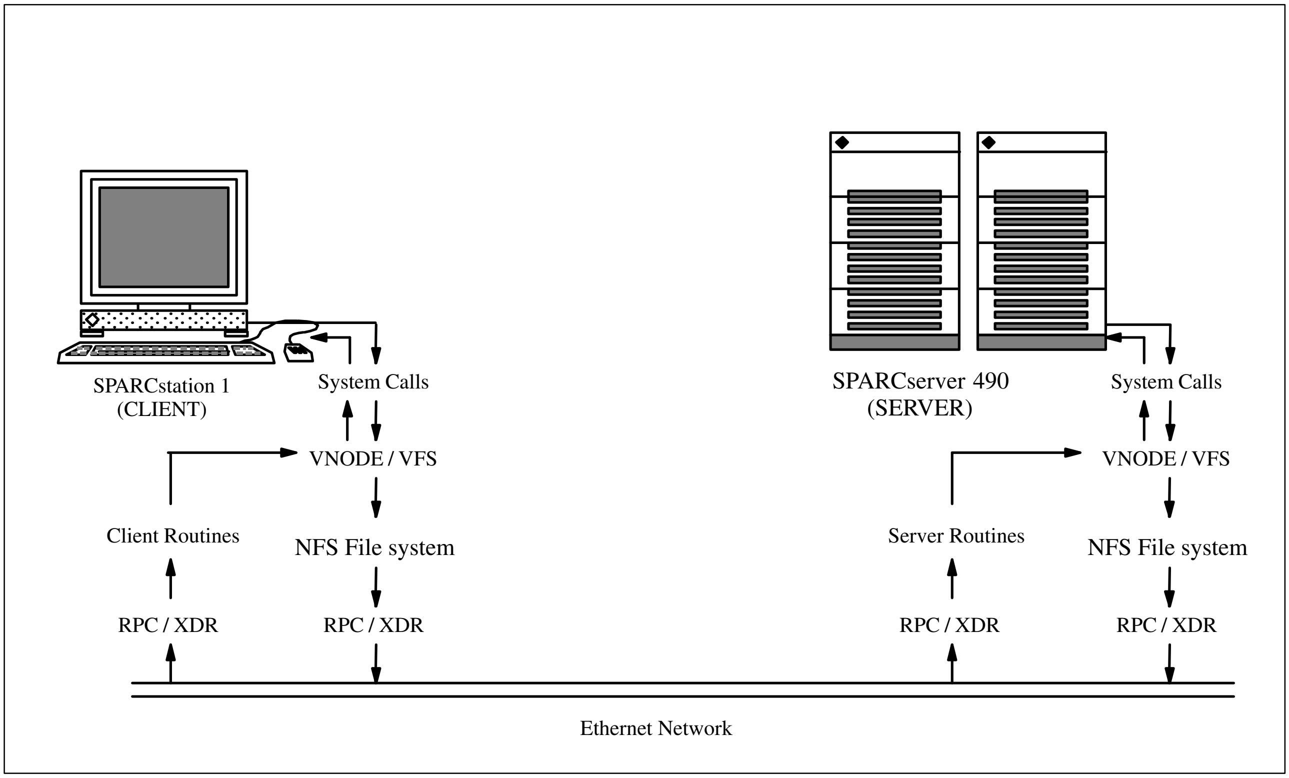
The most popular distributed file system is NFS. NFS provides transparent access to remote files on a network. All NFS files — regardless of actual location, host operating system, host machine type, or network characteristics — appear to be on a user's local disk. NFS files can be accessed using local file system commands. To deliver this powerful capability, computer vendors have adopted the NFS design for heterogeneous, network file management. NFS includes facilities for network–wide file and directory identification and access, remote request execution, and for managing server crash recovery.

The UNIX file system is composed of files and directories. Each file or directory has a corresponding *index node (inode)* which contains information such as location, size, ownership, permissions, and access time. Inodes are assigned unique numbers within a given file system only; an inode in one file system can have the same number as an inode in any other file system. This scheme is not well suited for distributed computing environments in which remote file systems are mounted dynamically. Sun's *virtual file system(VFS)* introduces the *vnode (virtual node)* concept to solve this problem. Vnodes, a generalized implementation of inodes, are unique across all file systems.

To allow remote files to be accessed just as if they were local, NFS includes a Remote Procedure Call (RPC) facility. Using the RPC facility, NFS translates local commands into requests for the remote host system. A local *caller* process communicates with a remote *server* process. The procedure call is executed remotely, but with the local caller's address space constraints. Because the caller and the server are two separate processes, they need not reside on the same physical machine. Remote procedure calls are synchronous — the client application is blocked or suspended until the server has completed the call and returned the results. The RPC mechanism is implemented as a library of procedures. These procedures utilize a standard for portable data representation, the eXternal Data Representation (XDR), to ensure consistent data transmission and decoding.

Figure 1 illustrates the flow of a request from a client workstation to a server. The request passes through the RPC and XDR layers, UDP/IP protocols,and is transmitted over the Ethernet. At the server, the request passes through the RPC/XDR layers and arrives at the NFS server. The server uses vnodes to access the correct VFS and to service the request. The same path is retraced in reverse to return the results. To the requesting workstation, there is no difference between a local and a remote VFS. The actual location of file system data is transparent.

Figure 1. NFS Environment



To minimize problems caused by system crashes, the NFS interface defines servers to be *stateless*. Servers do not remember anything about the state of clients on the network. Each procedure call passes all of the information necessary to complete the call, and does not depend on any past request. Handling a server crash is very easy: the client resends NFS requests until it receives a response or reaches a time–out limit. No other crash recovery is necessary for either the client or the server and data is never lost. To a client, servers that have crashed and recovered and slow servers are indistinguishable. Because of this stateless scheme, NFS servers must store any modified data to disk before returning results to the client.

Client workstations can mount any remote file system easily using the *mount* command. The *df* UNIX system utility lists all mounted file systems and reports the amount of free disk space on the mounted file systems.

2.2. Factors Affecting NFS Performance

Many factors affect NFS performance. Some of these can be *tuned* to yield better performance. Before tuning, the bottlenecks for your particular installation must be identified and the causes traced. Common bottlenecks and their causes are reviewed in this section. Later chapters discuss detailed troubleshooting tips and tuning instructions.

NFS Retransmissions: If the server cannot respond to a client's request, the request times out on the client system. The client will transmit a specified number of retries before it quits. Every retransmission imposes system and network overhead and increases network traffic. Excessive retransmissions are a common cause of network performance problems. There are several causes of excessive NFS retransmissions:

- Overloaded servers that cannot complete requests within the client's RPC timeout period
- Unreliable Ethernet interfaces dropping packets under burst conditions
- Physical network errors caused by unreliable hardware such as non–standard Ethernet cables.

A tool is available for analyzing retransmissions on your network. The *nfsstat* utility (Figure 2) measures the overall NFS retransmission rate. The *time–out* statistic reports the number of calls that timed out. The *retrans* statistic reports the number of times an RPC request is retransmitted at the request of the caller. In the example shown in Figure2, the retransmission rate is reported for a diskless client named *sunflower*. For this workstation, the number of retransmissions *per call* is very low — only 0.42% (457 *time–outs out of* 109440 *calls*).[1] Retransmission rates less than 1% may be caused by temporary failures such as a server rebooting. Retransmission rates that exceed 5–10% indicate a problem. Before planning any corrective action, it is necessary to investigate the causes of high retransmissions in your environment. Refer to the chapters on "Improving Network Performance" and "Improving Server Performance" for more details.

Figure 2. *nfsstat* Utility

sunflower% nfsstat

Client rpc:

calls	badcalls	retrans	badxid	time–out	wait	newcred
109440	356	101	0	457	0	0

Client nfs:

calls	badcalls	nclget	nclsleep			
109430	356	109432	0			
null	getattr	setattr	root	lookup	readlink	read
0 0%	48098 43%	1430 1%	0 0%	12293 11%	3987 3%	21062 19%
wrcache	write	create	remove	rename	link	symlink
0 0%	16317 14%	1933 1%	1581 1%	739 0%	357 0%	22 0%
mkdir	rmdir	readdir	fsstat			
3 0%	1 0%	1588 1%	19 0%			

Operating System Kernel: NFS performance is implementation dependent.In SunOS, NFS is built on a kernel–based design of the RPC/XDR protocol layers.SunOS 4.1 includes additional kernel–level enhancements. An NFS request cache allows a server to reject duplicate requests from clients and considerably improves the performance of heavily loaded servers. A dynamic back–off feature reduces the number of client retransmissions. These SunOS 4.1

1. The results from the *nfsstat* utility will overstate the *retrans* statistic in SunOS 4.1 and above due to the dynamic retransmission feature. The *badxid* statistic will be understated due to duplicate request cache algorithm embedded in SunOS 4.1 and above.

enhancements may deliver noticeable NFS performance improvements at your site. The number of NFS daemons also affect NFS performance. This issue is covered in detail in the introduction section of the chapter on “Improving Server Performance.”

Mounting File Systems: Mounting NFS file systems through gateways imposes performance penalties on the clients, on the gateway doing the routing, and on the remote system. If this mount strategy must be used, adjust the time–out value and retransmission count parameters as well as the packet size to avoid overloading the gateway with NFS packets. See chapter on “Improving Server Performance” for more details. In multi–server networks, avoid cross–mounting file systems containing system executables (ex: /usr/bin) between servers. If both servers have equal time–out values and retransmission count values, it is possible for the servers to initiate a deadly embrace — both servers can be waiting on each other.

Network Problems: If Ethernet traffic is heavy, clients experience longer delays waiting for a free slot for their NFS requests. If you suspect that performance problems may be due to heavy Ethernet traffic, there are several network characteristics to check. A high collision/defer rate is one indication of an over–loaded Ethernet. After checking for problem hosts on the Ethernet, the network should be segmented (partitioned) if the network–wide collision rate often exceeds 10%. Another indication of an over–loaded Ethernet is an averagel usage level of greater than 30%. This will result in an increased time to send requests to the server, increasing perceived server response time. The tool *traffic* is available for measuring network use.

The *netstat* utility returns information that is useful for analyzing network problems. Figure 3 is a sample output of this utility. *netstat* displays cumulative statistics for packets transferred, errors, collisions, the network addresses for the interface, and the maximum transmission unit (mtu). The data is collected starting at the time the system was last rebooted. *ipkts* and *ierrs* refer to input packets and error messages received by the client. Input errors can be due to poor network media, bad Ethernet boards, or under–configured server system software. *opkts* and *oerrs* refer to output packets from the client. Network error rate can be defined as the

$$\text{network–wide collision rate} = \frac{\sum_{host=1}^{all} \text{collisions}_{host}}{\sum_{host=1}^{all} \text{transmissions}_{host}} \neq \sum_{host=1}^{all} \frac{\text{collisions}_{host}}{\text{transmissions}_{host}}$$

For the diskless client *sunflower,* the collision rate is 0.64% (1338 *collisions /* 208800 output packets). The collision statistic includes only the collisions that occurred while the local host was transmitting a packet.[2] A collision rate for one host that’s much higher than the network–wide collision rate indicates a problem with that host, perhaps a bad transceiver, transceiver cable, or connection.

In networks with fast and powerful desktops, collisions are more often related to particular individual hosts than to overall poor network media. In such cases a high collision rate on one host doesn’t imply a high network error rate. If the collision

2. The results from *netstat –i* may understate the collision rate for *leN* type Ethernet interfaces, which count only the first zero, one, or two collisions of each packet transmitted.

rate reported by one client workstation is high, investigate that client workstation. Also examine the network topology and nearby cabling hardware if several clients in one area have abnormally high collision rates.

Network performance issues are covered in more detail in the chapter on "Improving Network Performance."

Figure 3. *netstat* Utility

sunflower% netstat

Name	Mtu	Net/Dest	Address	Ipkts	Ierrs	Opkts	Oerrs	Collis	Queue
le0	1500	mtnview–en	sunflower	287072	70	208800	0	1338	0
lo0	1536	loopback	localhost	2087	0	2087	0	0	0

sunflower%

<u>Client Bottlenecks</u>: The amount of memory installed on a diskless client affects network performance. With limited memory, a diskless client generates more frequent requests for pages of a file on the server. Increased paging over the network degrades network and client performance. With more memory, a client can cache more pages and reduce total NFS traffic on the network.

Certain applications — CAD or image processing, for example — are inherently memory and CPU intensive. If a client with minimal memory runs applications of this type, both the server and the network carry increased loads. The network experiences more traffic (most of which is client paging activity) and the server spends more time accessing the disk. The situation is worsened for write–intensive applications. In general, performance is optimal for applications that minimize disk writes and paging on both the server and client sides.

Adding a local disk for unshared local files, like *swap* and *tmp* files, contributes to better NFS performance. As disks become faster,smaller, quieter, and more economical, this becomes a more attractive option.Today, most clients store data files on a server to simplify administration and backup. Dataless clients — those configured either with only root and swap filesystems local, or with /usr and swap filesystems local and the *tmpfs* filesystem active — can realize substantial performance improvements without sacrificing the benefits of a centralized administration scheme.

There is no simple equation for the performance gains delivered when more memory and disks are added to a client workstation. Some situations will gain from expansion and others will not. In local area networks with several diskless clients, the cost of adding more memory and local disks to every workstation should be compared to the cost of alternative NFS tuning methods. Enhancing a key server, changing the network, or adding another server are viable option selected at many sites. Adding a single Sun Prestoserve NFS accelerator to a file server can also be less expensive and more beneficial than expanding storage on every client workstation.

<u>Server Bottlenecks</u>: The most significant NFS file server bottleneck is the server disk subsystem. NFS performance varies directly with the rate at which NFS calls are written to disk. Therefore, the mix of read, modify, and write operations is significant. If the majority of server operations are data modifications or writes, a bottleneck is likely. A bottleneck is unlikely if most server operations are read or query only. Since the speed of the disk is substantially less than the speed at which requests are processed (memory speed), disks are the gating factor for server performance.In comparison, disk seek and rotation times force clients to spend most of their time waiting for disk operations to complete.

If the CPU is used for activities other than client NFS service (i.e. time–sharing applications), the number of diskless clients that can be effectively supported decreases. Ideally, these servers should not perform heavy computations or I/O–intensive tasks in addition to terminal session management or network management functions. Tasks such as database operations and modem pool management will degrade network performance if the server is also supporting diskless clients.

2.3. Where To Start

This section has covered the key factors affecting network computing performance: the network file system, client bottlenecks, network problems, and server bottlenecks. As you reviewed the topics, you may have recognized aspects of your own network installation. If so, you can pursue a specific area by turning directly to the relevant chapter on servers or networks. If the optimal tuning path is still unclear for you, review each of the following chapters for more details about the factors that affect performance and associated guidelines for tuning. The NFS performance measurements supplied and explained in the "Measuring Network Performance" chapter will also contribute to your overall understanding of network performance and will aid your tuning efforts.

Chapter 3. Improving NFS Write Performance: Sun Prestoserve Accelerator

Sun Prestoserve combines hardware and software technology to improve NFS performance on heavily loaded SPARCservers. Equipped with Sun Prestoserve, a SPARCserver 470 or SPARCserver 490 can improve NFS responsiveness by up to 75%. This chapter includes a review of the typical disk usage of NFS, the Sun Prestoserve solution for accelerating NFS write operations, and guidelines for evaluating where to use the Sun Prestoserve on your network.

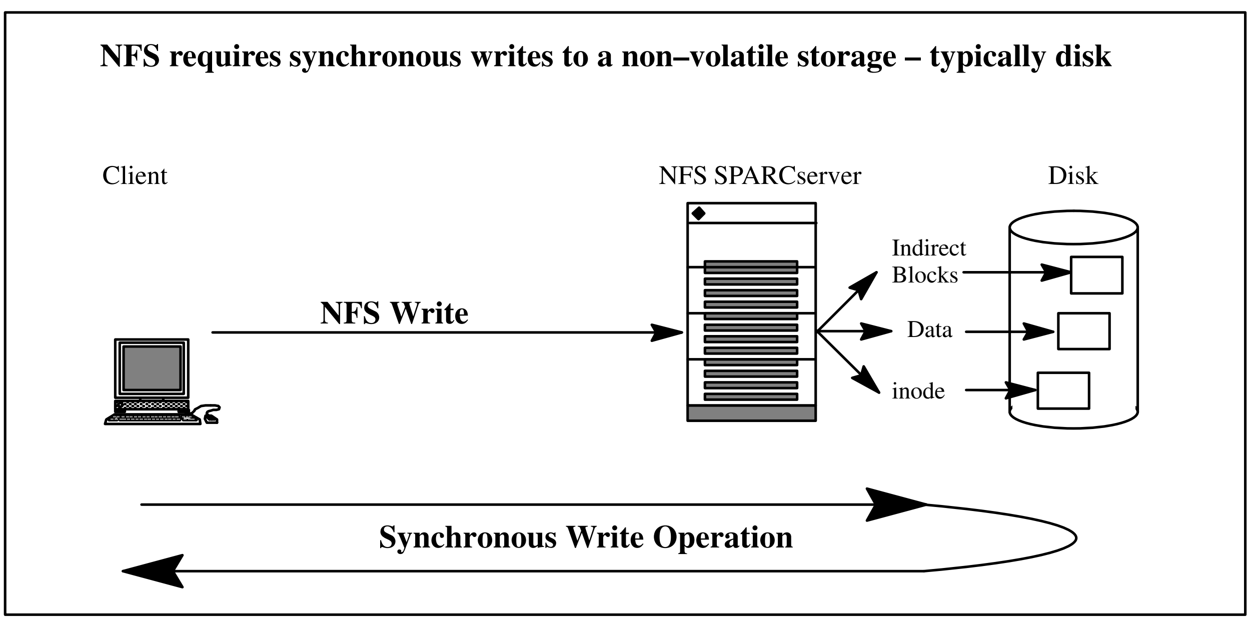
3.1. NFS Writes

A fundamental attribute of NFS is crash worthiness — no client data loss will result if a server fails during a file transfer This is a key strength of NFS, but it requires that all NFS transactions be written to disk on the server before returning results to the client. The data must be written synchronously, meaning that the server must wait for the write to complete before proceeding with the acknowledgement. Synchronous writes do not take full advantage of the sophisticated write optimization capabilities of Sun's IPI controllers.

The total overhead for NFS writes is further complicated. NFS transfers a file as multiple packets of information. To properly reconstruct the file, the packets must be accompanied with file definition information (inodes and indirect blocks). Therefore each NFS packet transfer results in multiple disk writes — the data itself, inode information, and indirect blocks are each separate write operations. As described above, these write operations can degrade performance; they are synchronous disk operations that do not take advantage of controller optimization technology

Sun Prestoserve accelerates NFS write performance without sacrificing the crash immunity of the NFS design.

Figure 4. NFS Write Operation

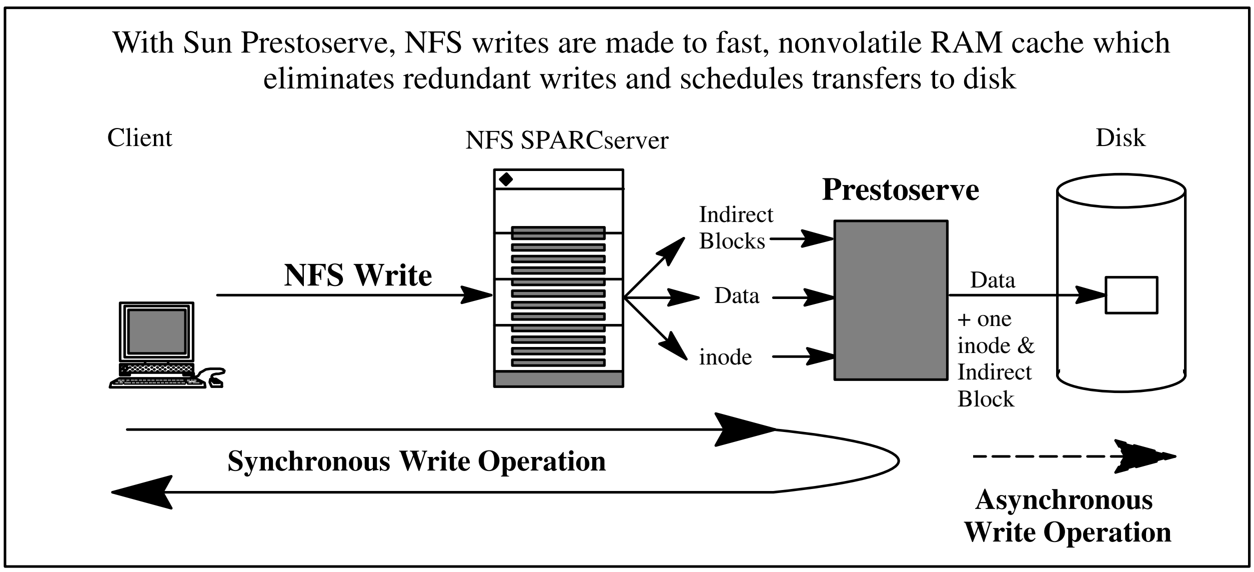


To summarize:

- NFS transfers must be recorded on non–volatile storage such as disks (for crash immunity)
- Each NFS write results in 2 to 3 writes to disk (inodes and indirect block information must be updated in addition to actual file data)
- NFS disk write calls are synchronous (the server must wait for the write to complete before returning the call)
- Synchronous writes don't utilize disk controller performance optimization technology.

By improving NFS write performance, Sun Prestoserve allows file servers to support more clients and offer faster response time. Sun Prestoserve accelerates NFS performance by caching critical file system data in nonvolatile memory, and efficiently scheduling disk writes in a manner tuned for NFS requirements.

Figure 5. How Sun Prestoserve Helps



3.2. The Sun Prestoserve Solution

Sun Prestoserve improves server performance in three ways. First, Sun Prestoserve reduces the number of writes to disk. When each data packet is received, Sun Prestoserve updates a cache. When the data is written from the cache to disk, only one copy of the inode and indirect block is written for each file. The redundant writes for inode and indirect block information are eliminated for each file. For example, the transfer of a 1–megabyte file requires 372 disk writes without Sun Prestoserve. The same transfer requires only 130 writes with Sun Prestoserve because all but one of the inode and indirect block writes are eliminated. This represents a 65% reduction in the number of required writes to disk (see Figure 6).

With Sun Prestoserve, NFS writes occur at memory speeds rather than disk speeds since they are buffered in the Sun Prestoserve nonvolatile cache. Clients executing write commands get almost–immediate response from the server.

Because the cache is a nonvolatile storage media, the realized performance improvement is made without sacrificing the data integrity advantage inherent in the NFS stateless protocol. If the file server should crash, the data on the Sun Prestoserve is maintained with a battery backup system.

Finally, multiple disk block writes can be scheduled efficiently from the Sun Prestoserve cache. Seek time is minimized and the rotational characteristics of the disks are taken into account. Data is flushed from the cache asynchronously in large 64–kilobyte blocks. This scheme allows the disk controller optimization technology to benefit NFS operations.

Figure 6. Number of Required Disk Writes

Example: NFS write of a 1–MByte file
Number of required disk writes

	Without Sun Prestoserve	**With Sun Prestoserve**
Data	128	128
inode	128	1
indirect blocks	116	1
Total	**372**	**130**

3.3. Situations That Benefit From Sun Prestoserve

The actual benefits delivered by Sun Prestoserve depend upon a server's:

- NFS write mix (percent of total NFS operations that append or modify data)
- volume of NFS calls (total number of NFS calls per second).

Servers that will benefit from Sun Prestoserve have three characteristics. First, at least 5 percent (and more reliably at least 15 percent) of all NFS operations are write or data modify operations. Second, the volume of NFS calls per second is significant. Finally, servers that will benefit from Sun Prestoserve are those that are heavily loaded with clients. These characteristics are described in more detail in the paragraphs that follow. Not all servers will benefit from Sun Prestoserve. Review the guidelines at the end of this chapter to identify unlikely candidates. The performance data presented in "Measuring Network Performance" chapter includes some benchmark results for servers equipped with the Sun Prestoserve compared with servers running without the accelerator.

An NFS Write Mix of 5 to 15 Percent

The greatest performance gains can be realized if NFS write or modify operations are a significant percentage of the total NFS operations performed. Noticeable improvements are possible if this percentage is greater than 5 percent, and ideally more than 15 percent. This percentage is referred to as the *NFS write mix* and is a function of the client types — diskless, dataless, or diskfull — and the application mix for those clients.

A large number of diskless clients will result in more NFS write operations on the server. Diskless clients rely exclusively on a networked file server for all operating system disk functions and file access. The volume of NFS calls per client is the highest with diskless clients. Dataless clients, those with *swap* and *tmp* files on a local disk, perform fewer NFS calls than do diskless clients, but still access all application and data files over the network. Diskfull clients perform the fewest number of NFS operations.

Some types of applications traditionally require more frequent saves to disk and generate higher percentages of NFS writes. These include CASE, CAD, and EPUBS applications.

If you suspect a high NFS write mix as the cause of low performance, use *nfsstat* to analyze server behavior. *nfsstat* will return the total number of NFS calls as well as percentages for individual operation types. Add the percentages for *write, create,* and *remove* calls to get the total NFS write mix. Average the results from several measurement periods and try to perform all measurements during peak load conditions. Before installing a new server, use *nfsstat* on another similarly–loaded server to determine if Sun Prestoserve should be installed with the new server.

Remember that a high write mix is not a stand–alone requirement for Sun Prestoserve. Without a heavy load on the server, a high write mix may not be a problem. A single client on a SPARCserver 490 will not easily overload the server, even when performing a high percentage of NFS writes. But, if speed of writes is important, the benefits of the Sun Prestoserve will be attractive even for a server without a heavy overall load.

A Heavily–loaded File Server

An over–worked NFS server will result in a frequent messages to the clients of that server: "NFS server not responding; still trying." High numbers of clients — especially diskless clients — will burden a file server. Heavily–loaded file servers are the most improved by Sun Prestoserve.

Exceptions

Sun Prestoserve will not improve NFS performance for all file servers. The accelerator is based on a cache for all synchronous writes to disk. Servers that do not often perform synchronous writes will see little if any improvement. Neither will lightly loaded servers benefit substantially from Sun Prestoserve.

To identify servers that are unlikely to benefit from Sun Prestoserve, look for:

- ♦ Frequent asynchronous I/O (e.g., lots of database applications performing raw I/O)
- ♦ Read servers (mostly reads and few writes operations).

Chapter 4. Improving Network Performance

4.1. Identifying Network Problems

To an NFS client, a slow server and a bad network appear the same. Whichever the source of the problem, NFS requests are not acknowledged in a timely manner, and retransmissions are frequent. If the server is heavily loaded, it cannot complete a client request within the client's RPC timeout period, and the client will retransmit the request. Severe loads may also impair the server's ability to receive packets from the network, forcing the client to retransmit when it does not receive a reply from the server. Finally, network loading will increase the perceived response time of the server, as network congestion adds to the delay a packet encounters when being transmitted from both the client and server

It is also possible for the network itself to be unreliable: An NFS server has additional bandwidth to handle requests, but the requests do not reach the server because network hardware such as bridges or routers consume one or packets composing the request. The link between client and server becomes ineffective.[3]

Systems administrators can differentiate network–related problems from server problems using *nfsstat*.[4] *nfsstat* displays a summary of server or client statistics. The *–c* option shows only client statistics (see Figure 7).

Figure 7. *nfsstat* Utility

```
sunflower% nfsstat –c

Client rpc:
calls       badcalls    retrans     badxid      time–out    wait        newcred     timers
109373      411         670         131         1074         0          0           666

Client nfs:
calls       badcalls    nclget      nclsleep
108963      1           108963       0
null        getattr     setattr     root         lookup     readlink    read
0  0%       24508 22%   285  0%      0  0%      27023 24%   37771 34%   6348  5%
wrcache     write       create      remove      rename      link        symlink
0  0%       1210 1%     262  0%     10  0%      5  0%       0  0%       0  0%
mkdir       rmdir       readdir     fsstat
3  0%       1  0%       11014 10%   523  0%
```

The *badxid* value indicates the number of times that a duplicate acknowledgement was received for a single NFS request. When a request is retransmitted, the same transaction id (*xid*) is used. In the event of a slow server, a client will initiate

3. UDP is connectionless and NFS is stateless, so if any of the Ethernet packets composing an NFS transaction are lost in transit the entire transaction must be retransmitted. If your network loses packets, you can compensate by manually reducing the NFS transaction size. If the client and server are separated by an IP router, SunOS 4.1 will change the NFS transaction size dynamically if conditions on the network warrant it.

4. The results from the *nfsstat* utility will overstate the *retrans* statistic in SunOS 4.1 and above due to the dynamic retransmission feature. The *badxid* statistic will be understated due to duplicate request cache algorithm embedded in SunOS 4.1 and above.

retransmissions even though the first transmission is not lost, but only delayed. The server will finally acknowledge the first transmission and each of the retransmissions as well, resulting in the duplicate acknowledgements for a single request. The *retrans* count indicates the total number of RPCs that were retransmitted when no acknowledgement was received within the time–out period.

If *badxid* and *retrans* are close in value, then the client is outpacing the server Increasing the server response time will improve network performance. If *badxid* is zero or significantly less than the retransmission count, then packets are being lost and the network itself or the server's network interface is to blame. This chapter will help you resolve these types of network problems.

4.2. Network Connection Problems

The heavier the traffic on an Ethernet link, the longer the delays imposed on client transmissions. Applications, such as *rwho*, can significantly increase Ethernet traffic by frequently using network broadcasting facilities. Checking for heavy traffic is straightforward, and can help you identify components that may be slowing your network.

The *netstat* utility counts the number of collisions and other errors registered by each network interface (see Figure 8). A network–wide collision rate that exceeds 5 percent indicates a possible problem, and one that exceeds 10 percent is a certain indicator of a problem. The problem could be electrical — more likely the network is overloaded.[5] Saturation can result from actual or perceived contention for the network. One or two machines dominating the network can create heavy traffic and increase collisions. A bad local Ethernet card can create perceived contention problems, but this will most likely be localized with only some of the users experiencing poor network response. If caused by heavy trafic, overall traffic and collisions can be reduced by partitioning the network. Network utilization is measured on a logarithmic scale; the network's efficiency will improve exponentially as you reduce traffic.

Figure 8. *netstat* Utility

```
sunflower% netstat –i

Name    Mtu     Net/Dest     Address     Ipkts     Ierrs   Opkts    Oerrs   Collis   Queue
le0     1500    mtnview–en   sunflower   1362112   85      1715200  93      53812    0
lo0     1536    loopback     localhost   89725     0       89725    0       0        0

sunflower%
```

Other sources of perceived contention problems are poor electrical characteristics of the network termination hardware, transceivers, or drop cables. *netstat* will indicate the input and output packet error rates(reported by the *Ierrs* and *Oerrs*

---

5. The collision rate can be calculated by dividing the number of collision counts (*Collis*) by the total number of out packets (*Opkts*).

statistics). [6] If input error rate exceeds 0.025 percent, the host may be dropping packets due to insufficient buffer space or CPU overloading, or poor network electrical characteristics may be damaging packets. Increasing Ethernet driver buffer space is discussed in Appendix A. Electrical problems to investigate include:

- Loose cable or transceiver connections. Older transceiver connectors that rely on the slide lock rather than friction fit may not seat fully into machine–mounted backpanel connectors such as those on Sun systems. Compensate for this problem by removing one of the two washers from under each screw post on the male (pins) end of the transceiver cable. This common problem is indicated by input, output, or collision rates on a particular host being dramatically higher than the corresponding network–wide error rate.

- Improper grounding. The shield of each thick Ethernet cable should be grounded in exactly one place. Terminators or barrel connectors that inadvertently contact metal ceiling supports or pipes can result in a lot of electrical noise in the network since they allow ground loop currents to flow Grounding problems can also occur if a single long drop cable is built out of a mixture of 802.3 and Type II transceiver cables.

- Missing termination, particularly when Cheapernet (thinwire) cabling is used. The most common cause of improper termination is a thinnet cable attached directly to a BNC connector. A thinnet cable should be connected via a T–connector with a 50–ohm terminator on the opposite side of the T. The unterminated network sometimes works, although certain pairs of machines may not talk to each other and packets may be randomly dropped.

- Signal reflection with thinnet cabling. This can be caused when a thinnet cable is attached between the BNC Ethernet connector on a host and a T–connector on the thinnet backbone. To avoid signal reflection, the host's transceiver must be attached to the backbone with as short a signal path as possible; in the case of thicknet Ethernet, the transceivers are attached in–line or with "vampire taps" onto the backbone. With thinnet, the transceiver is on the CPU (or Ethernet) board, and therefore the backbone must be brought all the way to the machine to ensure a proper electrical connection. The problem is proportional to the *total* length of all thinnet "drop cables" — so what appears to work on only one or two systems will shut down the network if done on every system.

- Bad or over–extended network segments. Bad segments are quite common with thinnet. The conductor and shield can be stressed when a machine is moved, or if the cleaning crew runs a vacuum cleaner over the cable. Check thinnet cable insulation for any breaks or marks that indicate rough treatment or stress.

- Inappropriately set transceivers. Sun-3 and Sun-4 systems will work equally well whether SQE (also called "heartbeat" or "Type II") is enabled or disabled.[7] But some other systems have more stringent requirements. Most notably, repeaters require that SQE be disabled on the transceivers they are connected to. Connecting a repeater to a transceiver that has SQE enabled will cause significant connectivity problems throughout the network.

---

6. Input and output packet error rates can be calculated by dividing the number of errors (*Ierrs* or *Oerrs*) by the total number of packets (*Ipkts* or *Opkts*, respectively).

7. If SQE is enabled, the red indicator labelled "CP" on older Cabletron equipment such as MT-800 and ST-500 will flash whenever a packet is transmitted, whether or not there is a collision. If you do not have newer equipment that has two separate indicators labelled "col" and "sqe", disable SQE so the red indicator flashes only for collisions.

If you are unsure about the electrical integrity of your networking hardware. consult with your local Sun sales ofice for assistance from Sun's Networking Services or one of Sun's network installation partners. These experts can identify the sources of many performance problems and eliminate cable–chasing headaches for you.

4.3. Other Hardware on the Network

Using *netstat*, you can analyze transmissions on your network. The previous sections help you identify situations that prevent packets from reaching their destinations. In addition to heavy traffic and low–level hardware problems, other hardware on the network can introduce transmission problems. Network partitioning hardware — bridges or routers — can drop packets forcing many retransmissions and resulting in degraded performance.Bridges also impose delays when they examine packet headers for Ethernet addresses. During such examinations, a bridge's network interface may not keep up with traffic, dropping one or more packet fragments completely.

Dropped packets can noticeably lower performance. Consider a typical NFS write: an 8–KByte block is being sent to a server. The block is divided into 6 packets by the UDP layer. Each Ethernet packet can contain a maximum of 1514 bytes; an 8–KByte block is sent as five full–sized 1514 byte packets and a shorter fragment containing the remainder of the data. These packets are reassembled on the server side. If a single packet is lost, the server cannot attempt the write operation and the client retransmits the request after it times out. If the bridge regularly drops packets, clients communicating through the the bridge will suffer from very high retransmission rates.

To compensate for bandwidth–limited network hardware, reduce the packet size specifications. The packet granularity variables — *rsize* (read) and *wsize* (write) — can be set as the command line options when using the mount utility, or by changing entries in the the */etc/fstab* file. Try reducing the appropriate variables (depending on the direction of data passing through the bridge) to 2048. If the bridge or other device passes data in both directions, reduce both *rsize* and *wsize*:

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

A similar problem can occur with the Ethernet interfaces of some servers. Many older Ethernet interfaces cannot reliably receive a burst of more than three or four packets. If dropped packets occur frequently for one server, adjust the write packet size variables for the clients using that server. The read packet size can be left as the default value, while *wsize* can be set explicitly:

```
farserver:/home  /home/farserver  nfs rw,wsize=2048  0 0
```

To test the reliability of your specified packet sizes, or to quantitatively determine the largest reliable size for your network, exercise the *spray* utility while varying the packet sizes. *Spray* is similar to NFS in that it works over UDP/IP and does not use any flow control.

```
% /usr/etc/spray thud –l 2048 –c 100 –d 2
sending 100 packets of lnth 2048 to thud ...
        in 1.0 seconds elapsed time,
        6 packets (6.00%) dropped by thud
Sent:   100 packets/sec, 200.0K bytes/sec
Rcvd:   94 packets/sec, 188.0K bytes/sec
```

In the previous example, the –d2 option for *spray* imposes a delay between each packet. With the delays, *spray* will test the reliable transmission of large packets — not streams of large packets. If used without a delay, the remote Ethernet driver can run out of buffer space and flush packets. Use *spray* to generate, but not count, packets. It will not report the reason for dropped packets; you will have to deductively determine if packets are dropped locally or on the physical media by taking into account the total network behavior and other problem symptoms.

4.4. Ethernet Segment Partitioning

Powerful systems do not guarantee optimal network–computing performance. A properly configured network is crucial. If implemented correctly, Ethernet segment partitioning can increase network performance on LANs with current traffic exceeding 40 percent of Ethernet's serial bit rate (see Figure 9). However, consider the impact carefully; performance will be lower when communicating with servers that become isolated from the new segment. Clients will have to mount and access file systems on remote servers by communicating over the backbone to another remote segment.

As an alternative to a backbone and segmented LAN scheme, a bridge can connect a server and its associated diskless clients to a larger network (see Figure 10). The server and diskless clients will be isolated and unencumbered by the total traffic on the larger network. They can still communicate with the entire network through the bridge.

Figure 9: Ethernet Segment Partitioning

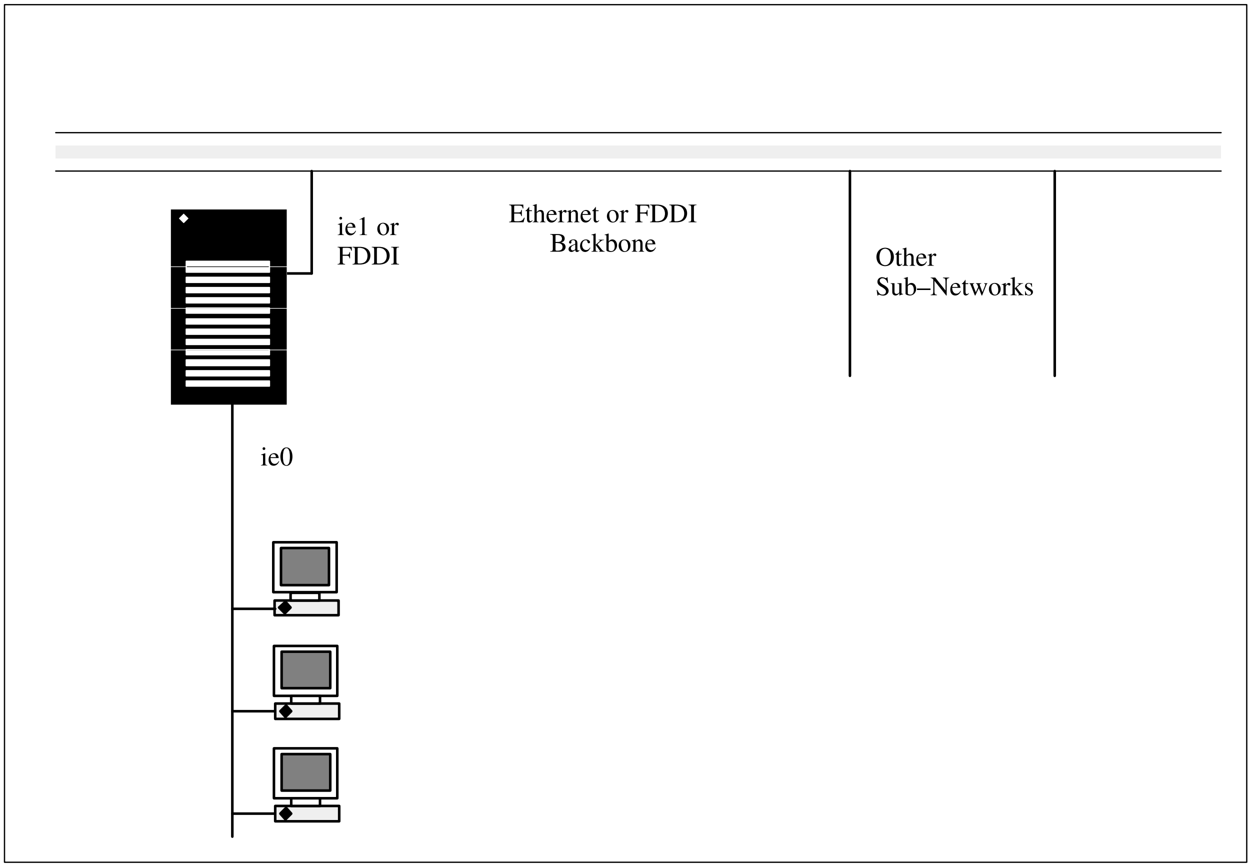
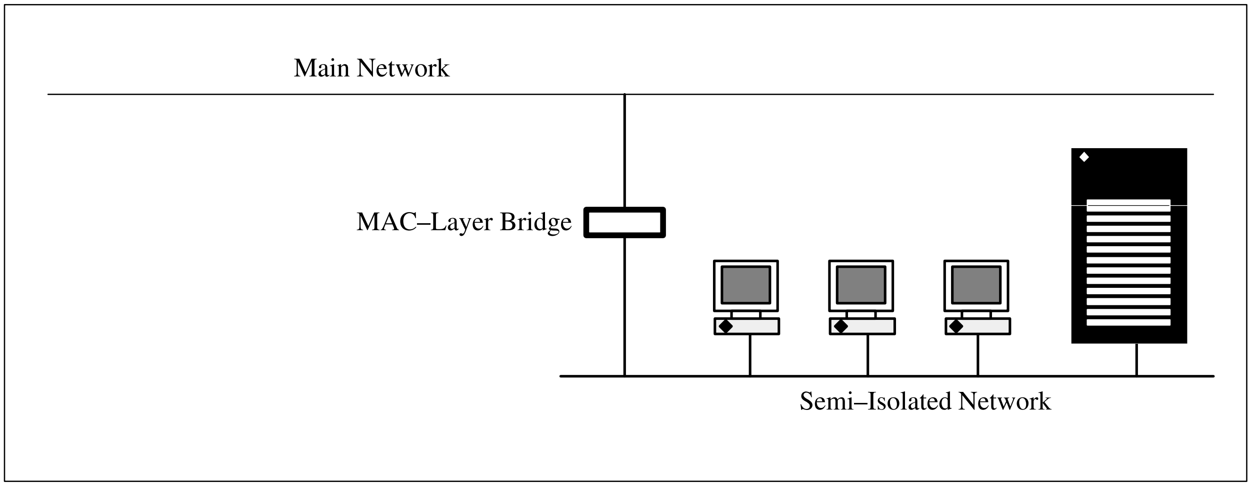


Figure 10: Ethernet Segment Partitioning Using Bridges



4.5. Network Information System (NIS) Configuration

NIS[8] activity does not generate excessive traffic on a network. Overhead associated with NIS activity can be minimized if NIS slave servers are used appropriately. For example, a file server can also be an NIS slave server. This reduces instances of NIS clients binding to servers on the other side of the network. For sub–networked environments connected by routers, an NIS slave must exist on the local cable — NIS binding uses network broadcasting and broadcasts are not forwarded by routers.

8. The Network Information Service (NIS) is primarily a point–to–point service. The ypbind daemon will broadcast a request for service when initialized and at any time that the response from the currently bound server becomes unacceptably slow. Aside from these isolated broadcasts, the majority of NIS transactions are directed to a specific server.

Chapter 5. Improving Server Performance

After finding and correcting problems associated with your network, you can focus your attention on the network servers. If *nfsstat* indicates that an NFS server is slow, the most likely explanations include:

- The server is unable to pull packets from the Ethernet due to an overloaded Ethernet driver or due to interrupt handling constraints. If the server is interrupt bound or the Ethernet device driver is continually running out of buffer space, then the server will not be able to handle all incoming traffic. As discussed in the chapter on "Improving Network Performance", the results of *netstat –i* include the input error rate for each Ethernet interface, and input operation can be stress–tested using the spray utility. Excessively high input error rates (greater than 0.025% during normal operation, or greater than 10% during a spray test) indicate that packets are not being extracted from the cable.
- The server cannot schedule *nfsd* daemons quickly enough. There are two instances when this occurs. A server with high CPU utilization will schedule *nfsd* daemons more slowly since other processes are competing for the same CPU. In other cases, large backlogs of NFS requests overflow the UDP socket structure and incoming requests are dropped. If the UDP socket structures fill up, *netstat –s* will report the number of overflows. In this case, adding more *nfsd* server daemons will reduce the number of overflows provided the server is not CPU–bound. Increase the number of NFS daemons (*nfsd*) from the default value of 8 to 12 in */etc/rc.local*.
- A server that is accepting all incoming requests but still performing poorly may be disk–bound. A high percentage of CPU idle time is an indication that the server is bound by disk bandwidth. Some combination of faster disks, more disks, additional disk controllers, or a more equitable distribution of NFS requests over all disks and controllers is required. Server performance decreases by file system effects such as a small *inode* cache, a very fragmented file system, or a commonly used hop that is a parent of several mount points on many machines (see chapter on the "Network File System").

This chapter explains each of these situations in more detail and provides recommendations for improving performance on affected servers. The last sections discuss other improvements that can be gained by adding multiple disks and multiple disk controllers.

5.1. Packet Reception

Packets can be dropped by an Ethernet interface if it is unable to interrupt the CPU or if there are no data buffers or descriptors available. To examine interrupt handling rates, use *spray* with a small packet size (to avoid buffer–filling effects). With this utility, you can find the maximum packets per second that the server can receive:

```
% spray bigserver –l 100 –c 1000
```

If the input error exceeds ten percent, increase the server's Ethernet buffer space as outlined in Appendix A. Input error rates are calculated from the results returned by the *netstat* utility Divide the total number of input errors (*Ierrs*) by the total number of input packets (*Ipkts*). Note that this threshold is significantly higher than that suggested for normal operation. Because spray stresses the server's network interface, it is expected to drop some small percentage of packets.

The type of Ethernet board installed can impact the overall Ethernet packet receive rate. A SunNet Ethernet/VME controller board is faster than the older Sun–2 Multibus–based Ethernet board. Only one additional Multibus Ethernet board can be added to a system as device ie1, although up to 3 additional SunNet Ethernet/VME controller boards can be installed as devices ie2, ie3, and ie4. Diskless nodes should be connected to the built–in (*ie0/le0*) interface first. Diskless nodes should not be connected to older Sun-2 Multibus–based Ethernet adapters, or even to SunNet Ethernet/VME boards if they are installed in a server that is slower than a 4/470 or 4/490.

5.2. CPU and Context Switching

As the Ethernet device driver receives packets, they are passed to the IP layer, to UDP, and eventually to a queue on a socket data structure. The *nfsd* daemons unqueue NFS requests from this socket; when a new request arrives, the next available daemon removes it from the socket and schedules a disk request. The *nfsd* daemons, running as separate processes, are suspended while disk requests are processed. Performance is optimized since there are multiple threads extracting packets from the socket. Each *nfsd* daemon is a separate execution thread that can be scheduled by the kernel. Running *netstat –s* will tell you the number of socket overflows for UDP.

A server can drop packets if the socket structure fills up with unread packets.This will happen when there are too few *nfsd* daemons, or more commonly when the server is CPU–bound and cannot schedule the *nfsd* processes quickly enough. At the other extreme, too many *nfsd* daemons can impose a context–switching penalty on the server, decreasing overall performance by increasing the time for scheduling a daemon.

Examine CPU use with *perfmeter* or *vmstat*. If the percentage of idle time is near zero, then the server is CPU–bound; a high percentage of idle time indicates that a busy server is disk bound. Comparing the percentages of system and user time indicates whether the CPU cycles are expended in the kernel or by user processes. Users logging into an NFS server almost always cause a noticeable performance degradation.

Changing the number of *nfsd* daemons can improve performance. Try this only if NFS requests are not being scheduled as quickly as they arrive and if the disk subsystems have adequate resources to handle the additional daemons.The number cannot be increased without limit. Context switching overhead will eventually outweigh performance improvements. This fine–tuning may yield minor improvements, but major performance problems can usually be traced to other bottlenecks or problems.

5.3. Disk Bandwidth, Load Balancing, and File System Effects

Disk and file system effects can lower NFS performance. Unbalanced disk loads,improperly configured file systems, or overloaded file systems can create bottlenecks. Disk and file system effects are lessened with the addition of disks or the redistribution of the files on the disks. Look for the following common problems.

Unbalanced Disk Loads

Ideally, incoming requests on NFS servers would reference all disks with equal frequency. Realistically, some disks are more popular than others, and efforts must be taken to balance the file systems. The effect of unbalanced streams of disk requests is exacerbated if a server has many slow disks instead of a few fast disks. On average, a server with six SMD disks and the same aggregate bandwidth as a server with two IPI disks will have a slower turnaround for any random disk request. If you cannot afford to upgrade to faster disks, at least try to fairly distribute popular files and file systems among the disks that you do have.

The explanation of this effect involves stochastic process modeling, but can be compared with events in a typical supermarket. If Market A has 2 checkout lines, with each checker processing 120 items per minute, while Market B has 6 lines with slower checkers that process only 40 items per minute, the aggregate checkout rates of the markets are equal. However, you will wait longer (on average) at Market B if you choose a line at random, because 5 times out of 6 you will choose the line that does not have the shortest total backlog. An NFS request destined for a particular file system can be considered a random selection; it cannot alter its destination if it finds that the request in front of it will take a long time to service. The fewer choices offered to the NFS request, the shorter its average delay in being scheduled by the disk controller.

***inode* Cache is Too Small**

The most common file system effect occurs when an NFS server is configured with a very small inode cache. NFS requests that can be satisfied from information in the inode cache — *lookup*, *getattr*, and *setattr* —comprise a large percentage of NFS requests handled by a server. The MAXUSERS parameter in the kernel configuration file governs the size of the memory–resident inode table. By default, this value is set to 8. To ensure an adequate cache size, increment MAXUSERS by one for each NFS client: a server with 32 clients should set MAXUSERS=40.

Poor Overall Conditions for File Systems

File systems can create performance problems when excessive fragmentation occurs or loading is unbalanced or excessive. A file system that is consistently at 90% or more of its capacity will perform poorly even when accessed locally.Block placement algorithms and hash tables designed to minimize seek and rotational latencies break down when a disk is overloaded. The increase in disk access time will be reflected as a corresponding increase in NFS request service times.

5.4. Multiple Disk Controllers

The seek and rotational latency optimization technology of a disk controller is not always beneficial to NFS. NFS writes are synchronous, and cannot be re–ordered for "elevator seeking" by the device driver, as they are when writes are done asynchronously. Read–ahead caches are ineffective since long sequential disk accesses using NFS may be received out of order or interspersed with other requests.

Multiple disk controllers allow parallel seeks to be performed, aiding performance in cases where incoming requests are distributed among the multiple disks. To improve the request distribution, *swap* space for the server and its clients should be divided equally among the controllers. Similarly, *root* file systems for diskless clients can be broken into multiple,smaller file systems, reducing contention for a single disk controller.

5.5. Data Architecture and Heavily–used Servers

A final area to consider is the data architecture of the network. Poor NFS client performance can be caused by undesirable disk naming schemes and the extensive use of symbolic links. The transparency achieved using symbolic links is more efficiently implemented with NFS. Consider the following example:

```
# mount fred:/home/projects /home/projects
# mount wilma:/home/source /home/source
# ln –s /home/source /home/projects/source
```

Each reference to */home/projects/source* affects both servers *fred* and *wilma*. *fred* must read the symbolic link and return a path name that the client will resolve to a file system mounted on another server. The inefficiency can be eliminated by mounting */home/source* on both */home/source* and */home/projects/source*, or by switching the mount point and symbolic link so that the link is resolved on the client machine.

At one customer site, we found that 30 percent of the NFS requests made to one server were symbolic link reads. Closer inspection revealed that this server exported */usr/local* to the entire network. But */usr/local* contained symbolic links to the *bin*, *lib*, *src*, and *man* directories scattered on other servers. Each reference to */usr/local* came through this server, even though the final target was another machine. Replacing the symbolic links with direct mounts of the target file systems reduced the load on this over–burdened server.

Chapter 6. Measuring Network Performance

6.1. Scope of Performance Measurements

Due to the complexity and variety of networks, no single benchmark can measure all aspects of performance in a distributed environment. A variety of benchmarks measure CPU and compiler efficiency. The server response time for processing NFS calls is one measure of client–server computing. The aggregate number of NFS calls processed by a server indicates network throughput levels and directly correlates to the number of clients that a server can support.

Some vendors and benchmark organizations quote NFS performance in terms of the number of NFS Operations Per Second (NFSops[2]) performed by a server, or the average response time experienced by clients of that server. Without any additional information about the clients or workload, these types of performance figures are of little value.

To accurately evaluate NFS server performance, gather the following data:

- Client and server configurations. When high–performance systems are configured as clients of a slower server, the clients can outpace the server and create many network problems.
- Workload details. A laboratory full of students compiling homework assignments will produce a radically different mix of NFS remote calls than a network used primarily to generate documentation, manage electronic mail services, or run CAD/CAM/EDA applications.
- Average and worst–case response times seen by client machines.

Observing response times is important. Network traffic, and NFS traffic in particular, assumes a non–uniform distribution over time. Rather than approximating a straight line, a graph of the number of NFS requests received per second over time will include bursts of activity. Similarly, the mixture of RPC calls will vary greatly during a typical day. While users read mail or news, RPC calls will produce one mix of activity; building and debugging a graphics application produces quite a different mix. The same engineer may do both during each working day. Observing an NFS server during a steady stream of requests will not indicate how the same server will respond during bursts of activity that impose three or four times the steady–state average rate of requests.

6.2. Recommended Benchmark

No benchmark can effectively replicate the diverse real–world user environment. But they can provide us with contextual information about the compute abilities of the network. Legato Systems, Inc. has developed a benchmark that objectively measures client–server response times and network throughput. The *nhfsstone* benchmark places an artificial load on the file server. The load simulates the NFS operation mixes observed on working systems in a variety of application environments. The *nfsstat* UNIX utility is used to measure the actual mixes for your server, and the results of this measurement are supplied to the benchmark. Workload is described by the NFS operations mix and the NFS operation request

[2] NFSops is the unit of measure used by the *nhfsstone* benchmark. *nhfsstone*, developed by Legato Systems, Inc., is mentioned here since it is well known and is a reasonable starting point when measuring NFS performance. Interested readers can obtain a free copy of the benchmark from Legato Systems by sending electronic mail to *request@legato.com* and entering “send unsupported nhfsstone” on the subject line. Some vendors have modified Legato's benchmark to produce better benchmark results on their machines. Sun recommends the use of Legato's standard benchmark program and default benchmark parameters.

rate. *nhfsstone* returns file server performance in terms of the NFS request response time and NFS request throughput. *nhfsstone* is a single–client synthetic–workload benchmark.

To use *nhfsstone*, the user supplies four characterization parameters:

- NFS operation mix (a file containing the results from *nfsstat*)
- NFS operation request rate (aggregate number of NFS calls per second)
- Target file systems (the remote file systems that will serve as targets for generated NFS requests)
- Test duration (the run time, or the total number of NFS requests to be generated).

The benchmark returns five statistics:

- Generated NFS operation mix
- Elapsed execution time
- Total number of NFS operation requests generated
- NFS operation request rate
- Average service time for NFS operation requests.

The following section contains some sample *nhfsstone* benchmark results for Sun's SPARCserver 490.

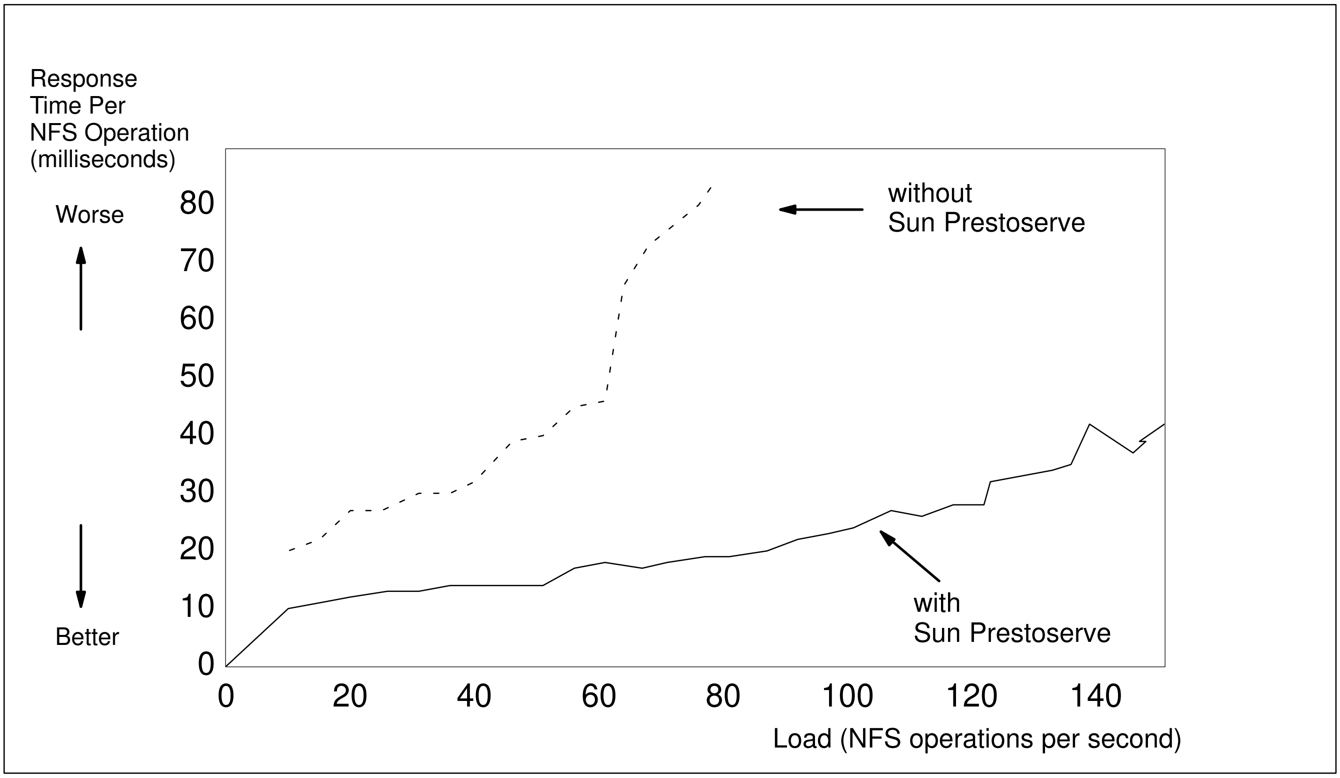
6.3. Sample Benchmark Results

Figure 11, on the following page, illustrates the results from the *nhfsstone* benchmark. The graph correlates the response time per NFS call (milliseconds) to a varying server load. The NFS mix and system configuration details are supplied in Appendix D. When the Sun Prestoserve accelerator is installed, the response time and NFS throughput on the SPARCserver 490 is substantially improved for the NFS operations mix used.

Useful approximations can be extracted from the graph. As an example, consider an NFS network with 15 diskless Sun–3 and SPARCstation 1+ clients. The measured activity is 50 NFS calls per second during normal working hours. Using the graph in Figure 11, we see that the average response time is approximately 40milliseconds per NFS operation for this activity level. With Sun Prestoserve installed in the SPARCserver 490, the average response time per NFS call will drop to 14 milliseconds. This implies that Sun Prestoserve decreases the SPARCserver 490 response time for this particular workload by 65 percent.

Even if the response time is not important, a Sun Prestoserve option can be used to support more clients on the same server. For a response time of 40milliseconds per NFS call, the SPARCserver 490 with Sun Prestoserve can sustain150 NFS operations per second — a 200 percent improvement in server throughput.This implies that the server can support many more clients at the same average response time.

Figure 11. Response Time versus Load on SPARCserver 490



Chapter 7. Conclusion

This paper has discussed the intricacies of network computing and offered some guidelines for configuring efficient networks. Following these guidelines will help you evaluate your particular computing environment, measure your workload,and select the best configurations and operating characteristics for your network. Before making any changes to your network, you should investigate and understand three critical characterizations of your current installation:

- The NFS operations mix for the various workloads your network experiences
- The highest aggregate rate of NFS requests presented to each server on the network
- The worst–case server response time that satisfies network users.

Applying the discussed troubleshooting tips could result in minimal to substantial performance improvements depending on the current state of optimization on your network. More important, the concepts reviewed in this paper will help you avoid some pitfalls. Some companies believe that the addition of more hardware and more powerful servers will eliminate all network bottlenecks — this is a fallacy. While it is much easier to add another server than it is to correct a poor network configuration, the new hardware will not eliminate problems with other network components or topology It is more important to understand the source of your performance problems. Then it will be obvious when more computing power is appropriate. Companies touting availability of several Ethernet ports or 50–MIP processors are offering features that may not contribute to the throughput in your particular computing environment — costly features when they remain largely unused.

Benchmark results by themselves are not indicative of what a customer can expect in their own computing environments. Response time and request handling values should be examined closely, as synthetic benchmark results can be colored by server loading, Ethernet bottlenecks, or an unrealistic RPC mix. Focusing purely on NFS performance may overlook some other problem with the computing system as a whole: NFS will only operate as efficiently as the rest of the networking protocol stack. As the top–layer application, it is impacted by performance problems in any underlying layer A system or network view of user requirements will often dictate the ideal configuration.

Future releases of NFS will make some currently–recommended network adjustments unnecessary. And we are hopeful that new and enhanced benchmarks will make it easier to benchmark NFS servers.

Appendix A. Increasing Ethernet Driver Buffer Space

In SunOS 4.0.x releases, the Ethernet driver parameters cannot be configured on–line. To adjust values, use *adb* and modify your copy of */vmunix* on the disk. Reboot the system to activate the new values. In SunOS 4.1, these parameters are in the files:

/sys/sunif/ie_conf.c and */sys/sunif/le_conf.c*

Before making any changes, create a backup copy of */vmunix*:

```
cp /vmunix /vmunix.old
```

There are three parameters of interest: *ie_rbds*, *ie_rfds*, and *ie_rbufs*. (For the Lance Ethernet device, the variables are prefixed with *le_*). The default values in SunOS 4.0.3 are:

Name	Default	Comment
ie_rbds	10	receive buffer descriptors (tiny)
ie_rfds	9	receive frame descriptors (tiny)
ie_rbufs	25	receive buffers (approx. 1500 bytes each)

The values of these variables must obey the relationship:

ie_rbufs > ie_rbds > ie_rfds

To start, try doubling the number of receive buffer descriptors, and adjust *ie_rfds* and *ie_rbufs* accordingly. To make these changes in SunOS 4.0.3, use *adb* on the kernel and reboot (note that the numbers entered are in hexadecimal, e.g., 14 hex is 20 decimal, 13 hex is 19, and 28 hex is 40):

```
adb –w –k /vmunix /dev/mem
ie_rbds?W 14ie_rfds?W 13ie_rbufs?W 28<control–D>
```

In SunOS 4.1, edit the configuration file (sys/sunif/Xe_conf.c) and rebuild the kernel. Also note that in SunOS 4.1, there are two sets of variables. The high values are used when multiple Ethernet interfaces are installed; the low values are for machines with only one interface. The default values are the same. Machines with multiple Ethernet interfaces may require hand–tuning.

Reboot and run the modified */vmunix* for a few days. If the Ethernet input error rate is not significantly lower with the modified values, the input errors are being caused by some other problem – probably electrical in nature.

Appendix B: SunNet Ethernet/VME Controller Board Configuration[9]

DIP switch settings:

SW0201	8	7	6	5	4	3	2	1
Base Addr	A23	A22	A21	A20	A19	A18	NA	24bit
300000	on	on	off	off	on	on	–	off
340000	on	on	off	off	on	off	–	off
380000	on	on	off	off	off	on	–	off
3c0000	on	on	off	off	off	off	–	off

Kernel configuration file entries:

```
device        ie2 at vme24d32 ? csr 0x31ff02 priority 3 vector ieintr 0x76
device        ie3 at vme24d32 ? csr 0x35ff02 priority 3 vector ieintr 0x77
device        ie4 at vme24d32 ? csr 0x39ff02 priority 3 vector ieintr 0x78
```

Add interface configuration (*ifconfig*) commands to */etc/rc.boot* and */etc/rc.local* for each interface added to the kernel.

9. Refer to Sun documents (813–1068–01 Rev 01 and 800–8027–02 Rev A) for more information.

Appendix C: Network Management – Boot Phase Problems

Often, problems occur when diskless clients attempt to boot from a server.Understanding the boot process for a diskless client makes client troubleshooting much easier (see also pages 70–73 in the SunOS 4.1 System and Network Administration Manual). Here is a summary of this process with potential problems mapped to possible causes:

Boot Phase Problems

Process Steps	***Possible Failures***	***Likely Causes***
1. Client powers on and finds its Ethernet address.		
2. Client issues RARP request in order to discover its IP address.	Wrong IP address returned from server.	(a) Bad data in NIS*ethers* map. (b) Wrong server responds to request.
	No IP address returned.	No *rarpd* running on server, or client broadcast not seen because of bad hardware or improperly configured network (for example, client not on the same cable as the server).
3. Client loads boot program via TFTP.	No response from server.	(a)*in.tftpd* disabled on server. (b) No hex entry for client IP address in server's */tftpboot*.
	No boot file found.	Symbolic link in */tftp–boot*points to nonexistent file.
4. Client executes boot program.	Garbled boot program.	Symbolic link in*/tftp–boot* points to wrong type of boot program (e.g., Sun–3 instead of Sun–4).
5. Client issues another RARP request to rediscover its IP address.		Same as in step 2.

Continued on next page.

Boot Phase Problems (continued)

Process Steps	***Possible Failures***	***Likely Causes***
Continued from previous page.		
6. Client issues WHOAMI request to discover its host name.	No response.	*bootparamd* not running on server.
	Wrong response.	Bad data in NIS *bootparams* map or in */etc/bootparams*.
7. Client issues *getfile* request to bootparams server to discover NFS host for *root* and *swap*.	No response.	*same as for step 6.*
8. *bootparam* server sends IP address of NFS server to client.		*same as for step 6.* Also can be bad IP addresses in NIS hosts map for NFS server or diskless client.
9. Client mounts root file system.	No response.	No *rpc.mountd* and/or *nfsd* running on NFS server.
	"Permission denied" or "NFS Write Error 13."	Root file system not exportable to diskless client or not exported.
10. Client opens *vmunix*.	"vn_open fails."	No *vmunix* on root file system or wrong *vmunix* present (i.e., wrong architecture).
	"Permission denied."	*root* exported but permissions are wrong.
11. Client loads *swap* file system.	"NFS Write Error 13" or "Permission Denied."	Same as for step 9. File doesn't exist.
	General confusion or reboot.	Bad data in *bootparams* map for *swap* file system.

For information about problems occurring after the boot phase and during normal NFS use, see pages 70 to 73 in the SunOS 4.1 System and Network Administration Manual. These pages include a good summary of the NFS error messages.

Appendix D. Hardware and Software Configuration

SPARCstation 370

25–MHz SPARC IU
25–MHz SPARC FPC + 25–MHz ACT8847 TI FPP
8–MBytes RAM
128–KBytes virtual–address write–through cache
Lance Ethernet Controller
SunOS 4.1

SPARCserver 490

33–MHz CY7C601 SPARC IU
33–MHz CY7C608 SPARC FPC + 33–MHz ACT8847 TI FPP
64–MBytes RAM
3–MBytes/sec. 1–Gbyte IPI Disk + ISP–80 IPI Controller
SunOS 4.1 PSR_A (Beta)

Nhfsstone

Default *nhfsstone* NFS Operations Mix

NFS Operation	*Nhfsstone*
null	0%
getattr	13%
setattr	1%
root	0%
lookup	34%
readlink	8%
read	22%
wrcache	0%
write	15%
create	2%
remove	1%
rename	0%
link	0%
symlink	0%
mkdir	0%
rmdir	0%
readdir	3%
fsstat	1%

Appendix E. References

Joseph Moran, Russel Sandberg, Don Coleman, Jonathan Kepecs, and Bob Lyon, *Breaking Through the NFS Performance Barrier,* Legato Systems, Inc. (1/90)

Bruce E. Keith, *Perspectives on NFS File Server Performance Characterization,* USENIX Summer Conference, June 11–15, 1990, Anaheim, California.