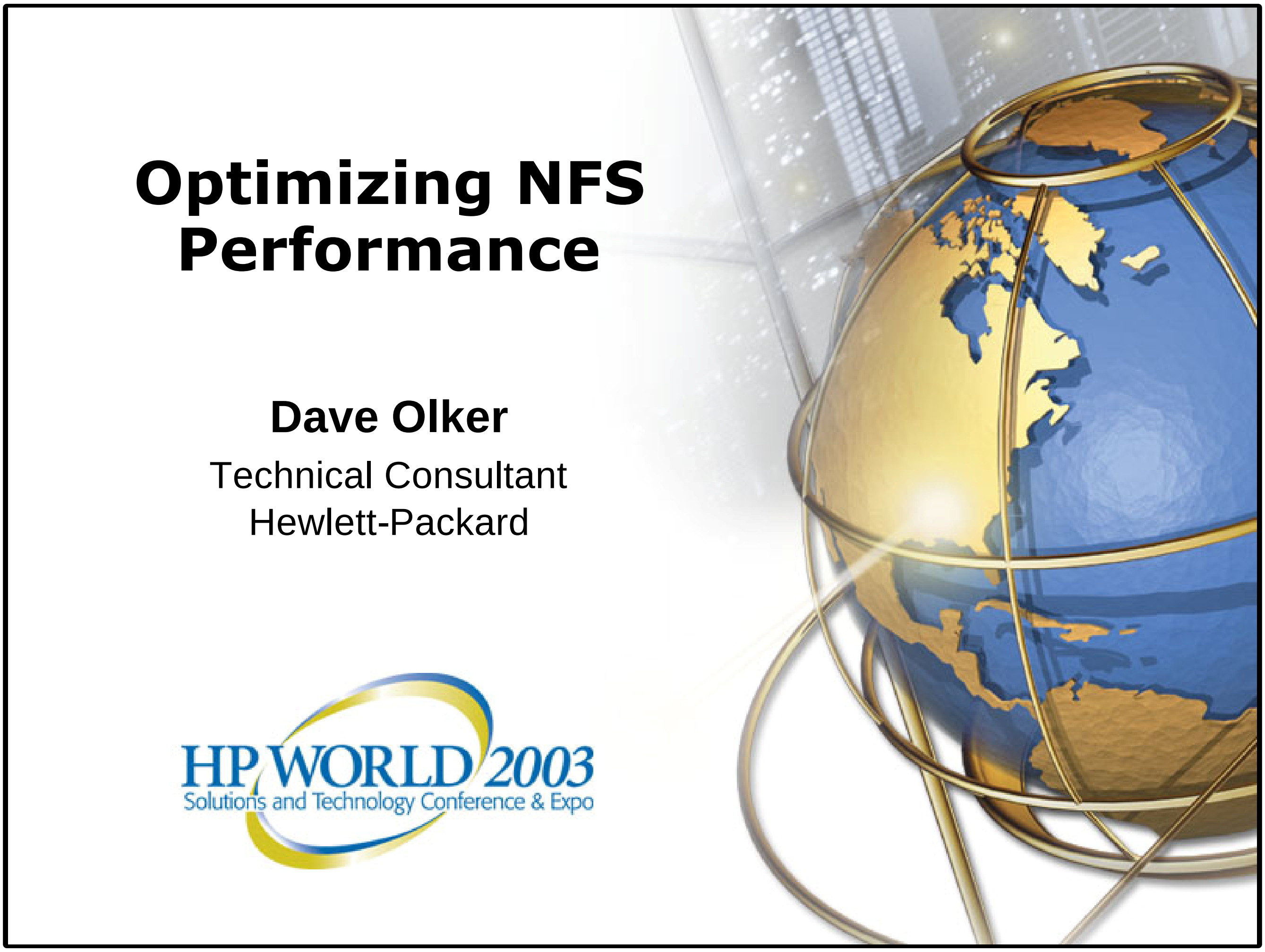


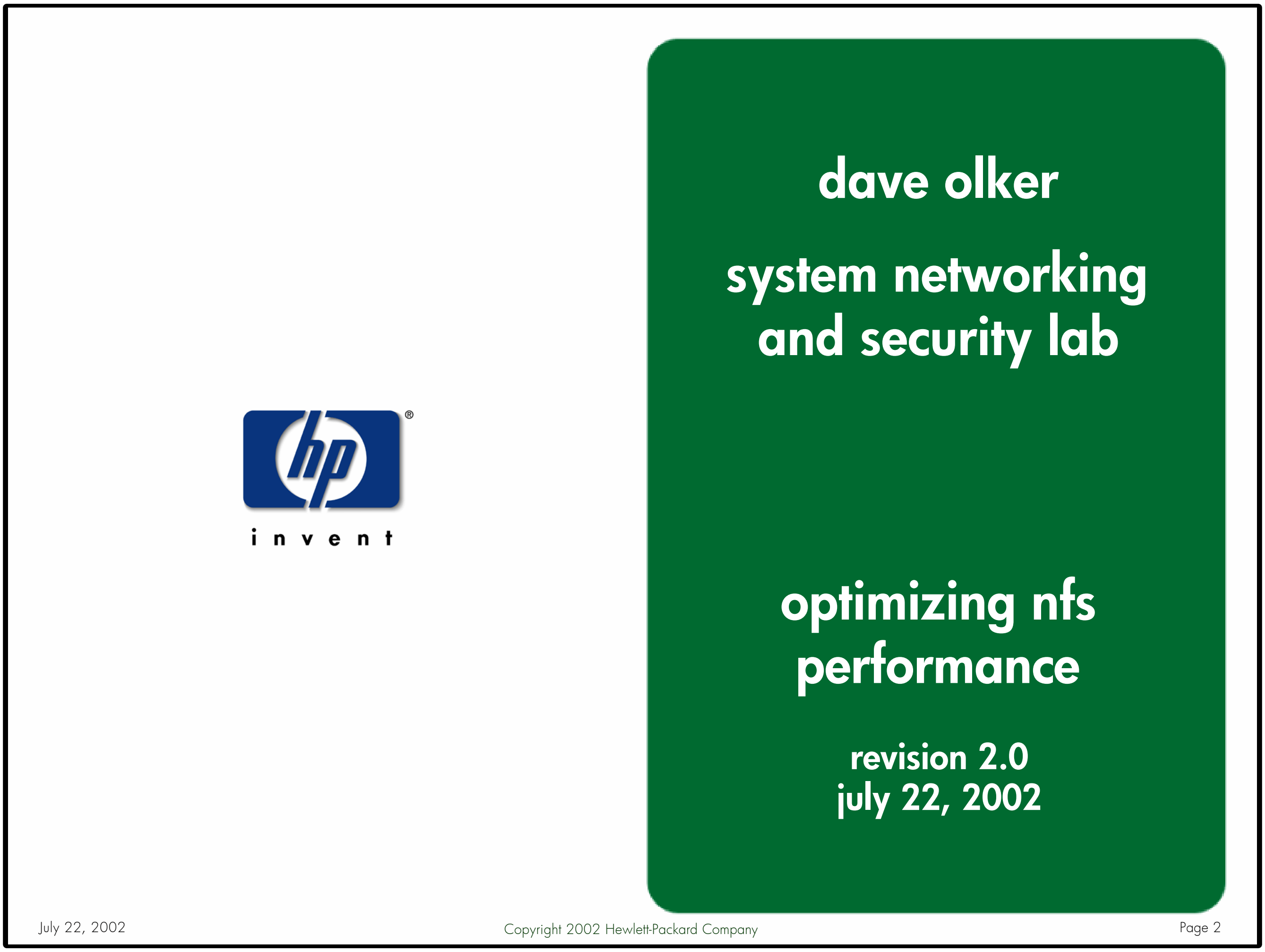
Optimizing NFS Performance

Dave Olker

Technical Consultant
Hewlett-Packard

HP WORLD 2003
Solutions and Technology Conference & Expo

Notes:



dave olker

system networking
and security lab

optimizing nfs performance

revision 2.0
july 22, 2002

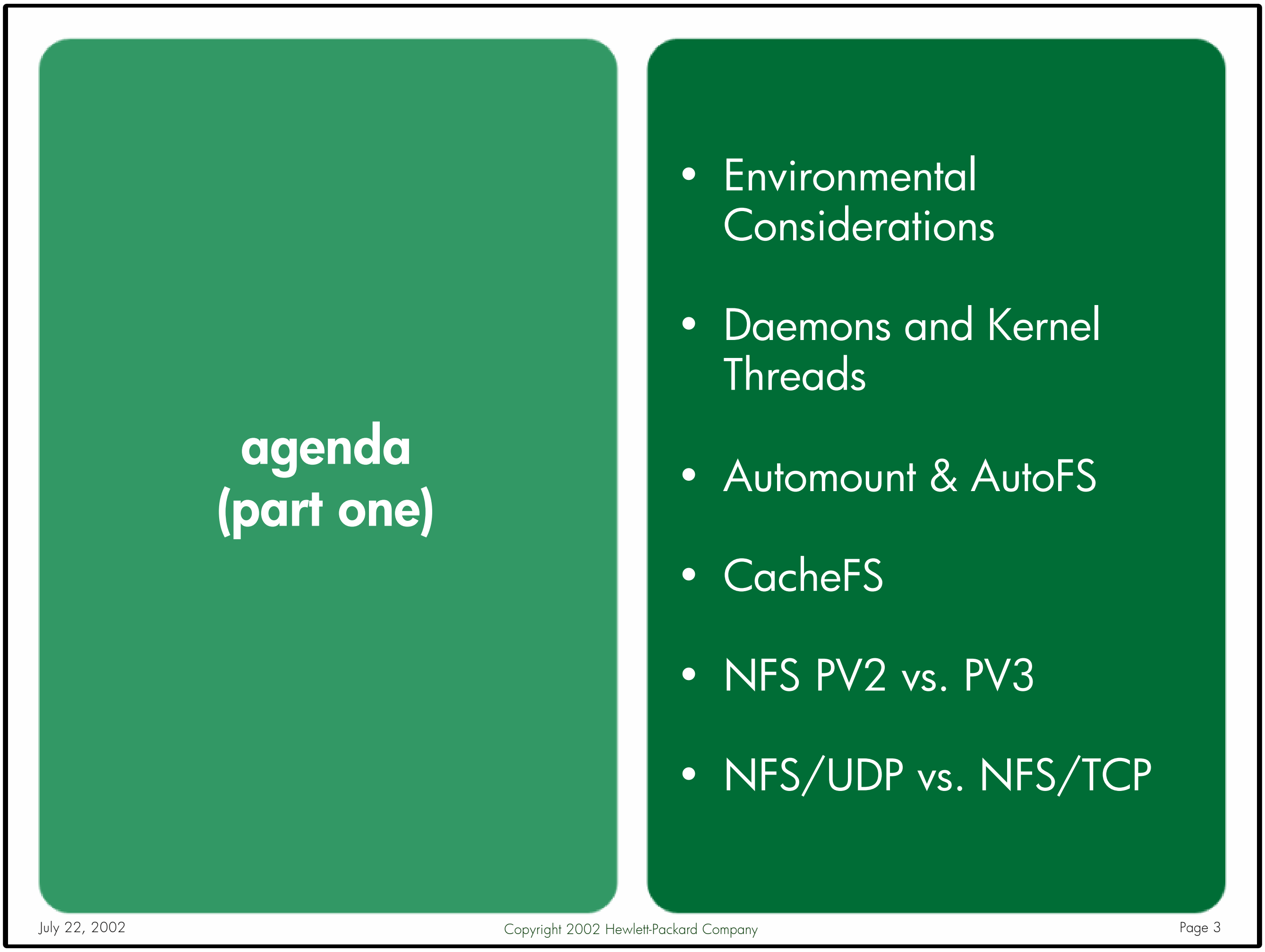
July 22, 2002

Page 2

Notes: Networked File System (NFS) has been the industry standard protocol for remote file access on the UNIX operating system platform for many years. It has become a critical component of every flavor of UNIX, as well as many non-UNIX based operating systems. NFS is also a central component of HP's NAS offerings. However, there is very little information available describing how to properly tune NFS clients and servers for optimal performance.

HP recently released HP-UX version 11i to support our new SuperDome hardware platform. Many changes were made to our NFS offering in this version of HP-UX; however, some of these differences are not currently described in any customer-viewable documentation. Also, there are no books available that specifically describe HP's NFS implementation.

The target audience for this presentation is systems, network, and storage administrators who want to learn how to optimally configure their HP-UX clients and servers for NFS performance. Some familiarity with the basic concepts of NFS and HP-UX internals are assumed.



agenda (part one)

- Environmental Considerations
- Daemons and Kernel Threads
- Automount & AutoFS
- CacheFS
- NFS PV2 vs. PV3
- NFS/UDP vs. NFS/TCP

July 22, 2002

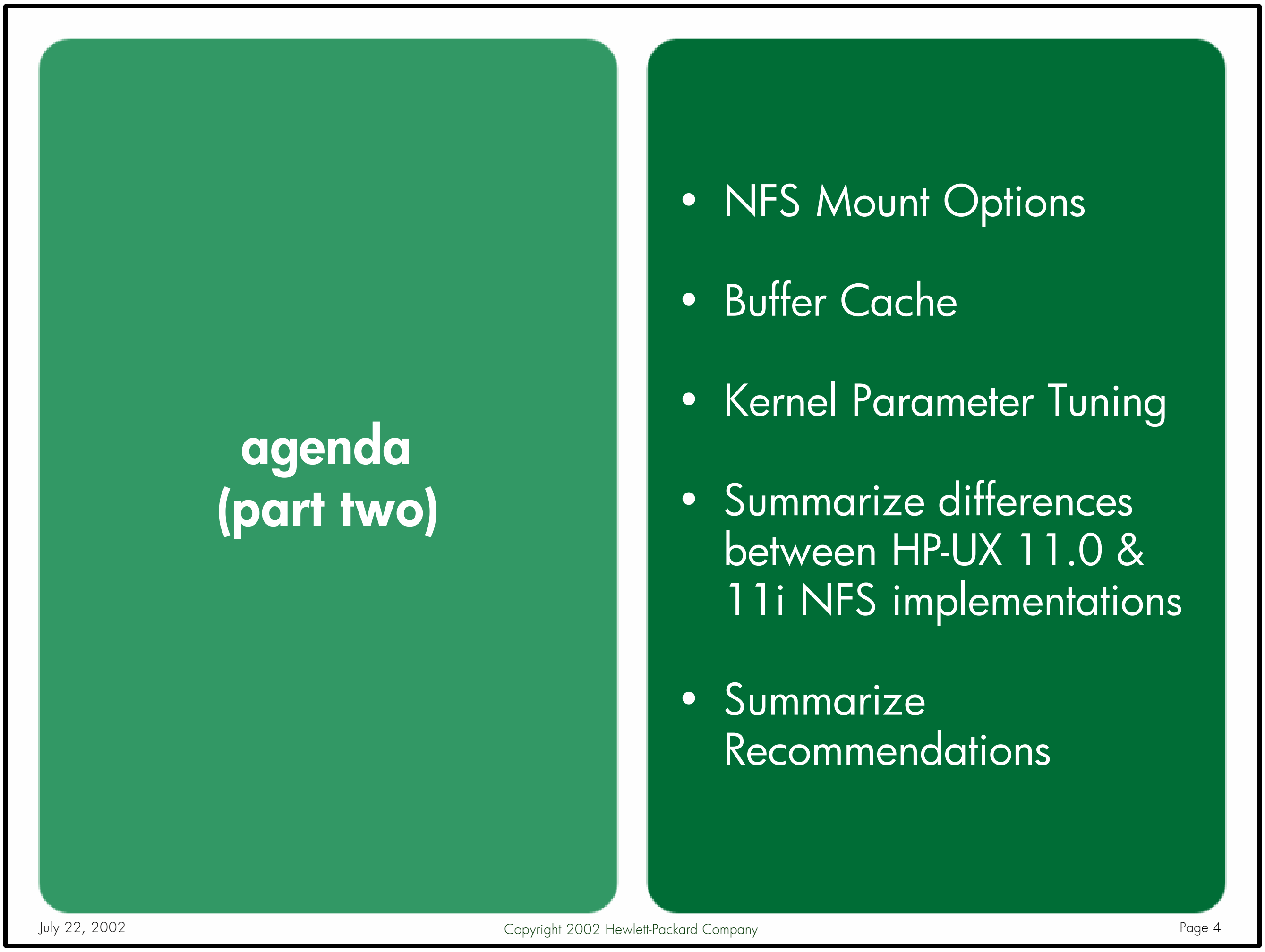
Page 3

Notes: An important step in tuning HP-UX systems for NFS performance is to evaluate the entire environment in which the client and server systems reside. The underlying network, local filesystems, and operating system patch levels can heavily influence throughput results.

There are many client and server daemons and threads involved in implementing the ONC+ protocol suite (including the optional AutoFS and CacheFS products). The behavior of these daemons and threads, as well as their impact on NFS performance, is discussed.

Many differences exist between the versions of the NFS protocol itself (HP-UX currently supports versions 2 and 3). These protocol differences can dramatically affect performance.

The choice of underlying network protocols, UDP/IP and TCP/IP, can also have a significant impact on how NFS behaves and performs.



agenda (part two)

- NFS Mount Options
- Buffer Cache
- Kernel Parameter Tuning
- Summarize differences between HP-UX 11.0 & 11i NFS implementations
- Summarize Recommendations

July 22, 2002

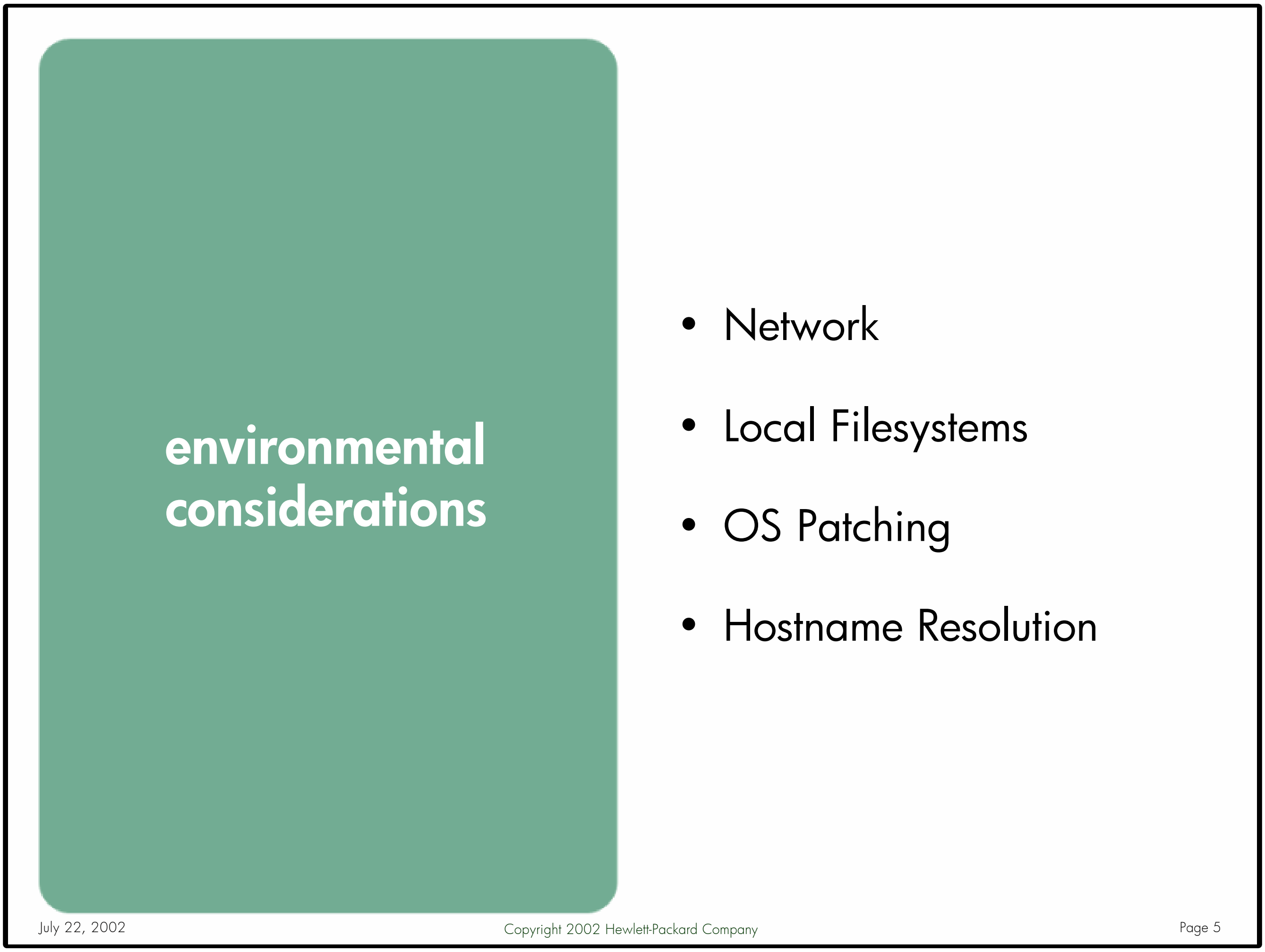
Page 4

Notes: There are many NFS-specific mount options available. Some of these options can have a positive impact on performance, while others can have a dramatically negative effect. It is important to know which options to use and which to avoid.

The buffer cache memory subsystem is heavily used by NFS. Sizing this resource properly on the client and server is critical. In addition to sizing considerations, it is also important to understand the differences between static and dynamic buffer cache allocation mechanisms and recognize which one to use in your environment.

Since much of NFS runs in the kernel, it should come as no surprise that there are many kernel parameters that can influence the behavior and performance of NFS. Tuning these parameters can be time consuming, but is well worth the effort.

Some of the NFS differences between HP-UX 11.0 and 11i are discussed during the presentation. A complete summary of these differences, as well as a high-level summary of all the tuning recommendations, is provided at the end.



environmental considerations

- Network
- Local Filesystems
- OS Patching
- Hostname Resolution

July 22, 2002

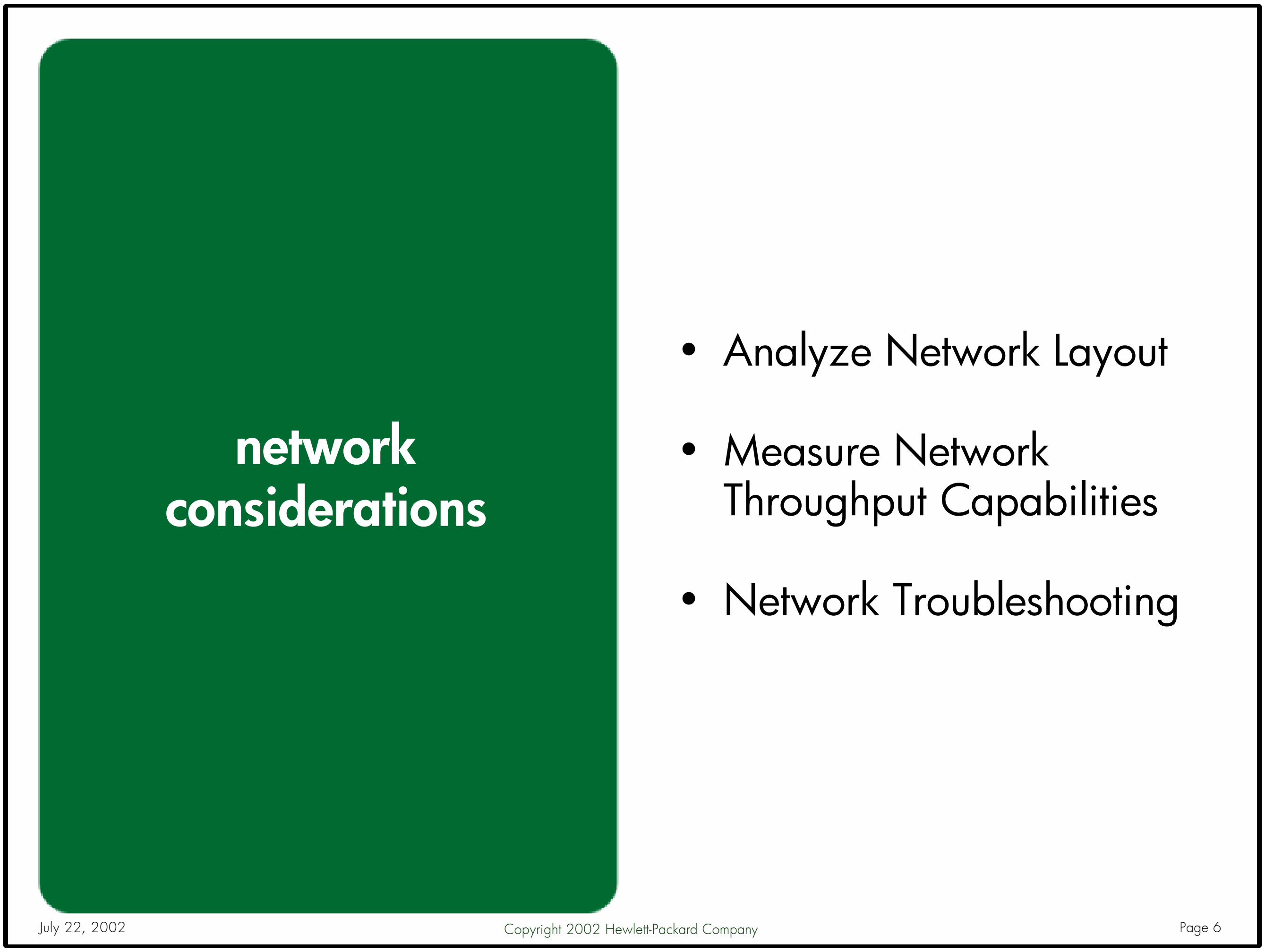
Page 5

Notes: NFS is essentially a network-based application that runs on top of an operating system, such as HP-UX. Like most applications, NFS competes for resources (such as disk, network, memory, and kernel tables) with the other processes on the system. When investigating any NFS performance issue, it is important to perform a “sanity check” of the overall environment in which the clients and servers reside.

NFS functionality is heavily dependent upon the behavior of many subsystems (i.e. disk, network, memory) and is therefore susceptible to performance problems in them. In other words, if the performance of the network is slow, NFS throughput will most likely suffer. Similarly, if local filesystem read and write throughput on the NFS server is slow, NFS read and write throughput to this server will likely be slow.

NFS continues to evolve over time. Consequently, it is a good idea to keep the HP-UX operating system on your NFS systems up-to-date with available patches.

Most NFS components rely in some way on hostname resolution (i.e. rpc.mountd, rpc.lockd, rpc.statd, automount, etc.). It is therefore important to verify that your hostname resolution servers respond quickly and provide accurate information.



network considerations

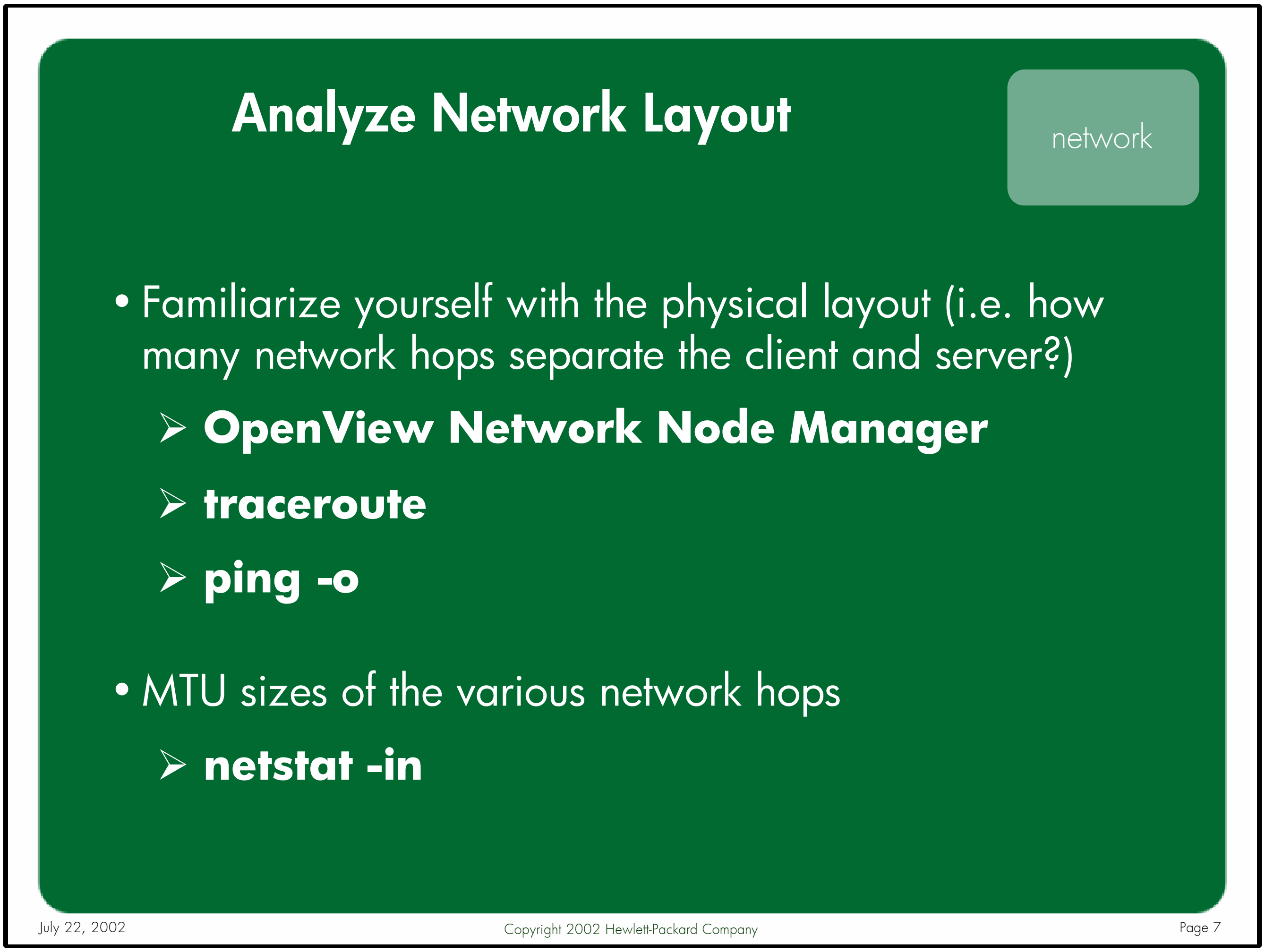
- Analyze Network Layout
- Measure Network Throughput Capabilities
- Network Troubleshooting

July 22, 2002

Notes: Since NFS is an acronym for "**Network** File System", it should come as no surprise that NFS performance is heavily dependent upon the latency and bandwidth of the underlying network. Before embarking on a detailed investigation into a specific area of NFS, it always makes sense to first verify that the underlying network is performing as expected.

This section will not be describing the myriad of available networking topologies and interface cards. NFS runs on most any networking link, and it typically performs faster on faster links. There is a wealth of information about the latest and greatest networking technologies (such as Gigabit Ethernet, APA, etc.) available from HP's IT Resource Center: **http://itrc.hp.com** and HP's online documentation repository: **http://docs.hp.com**.

What will be described is a recommended methodology and set of tools available for understanding the physical layout of your network, measuring its throughput, and performing routine network troubleshooting.



Analyze Network Layout

network

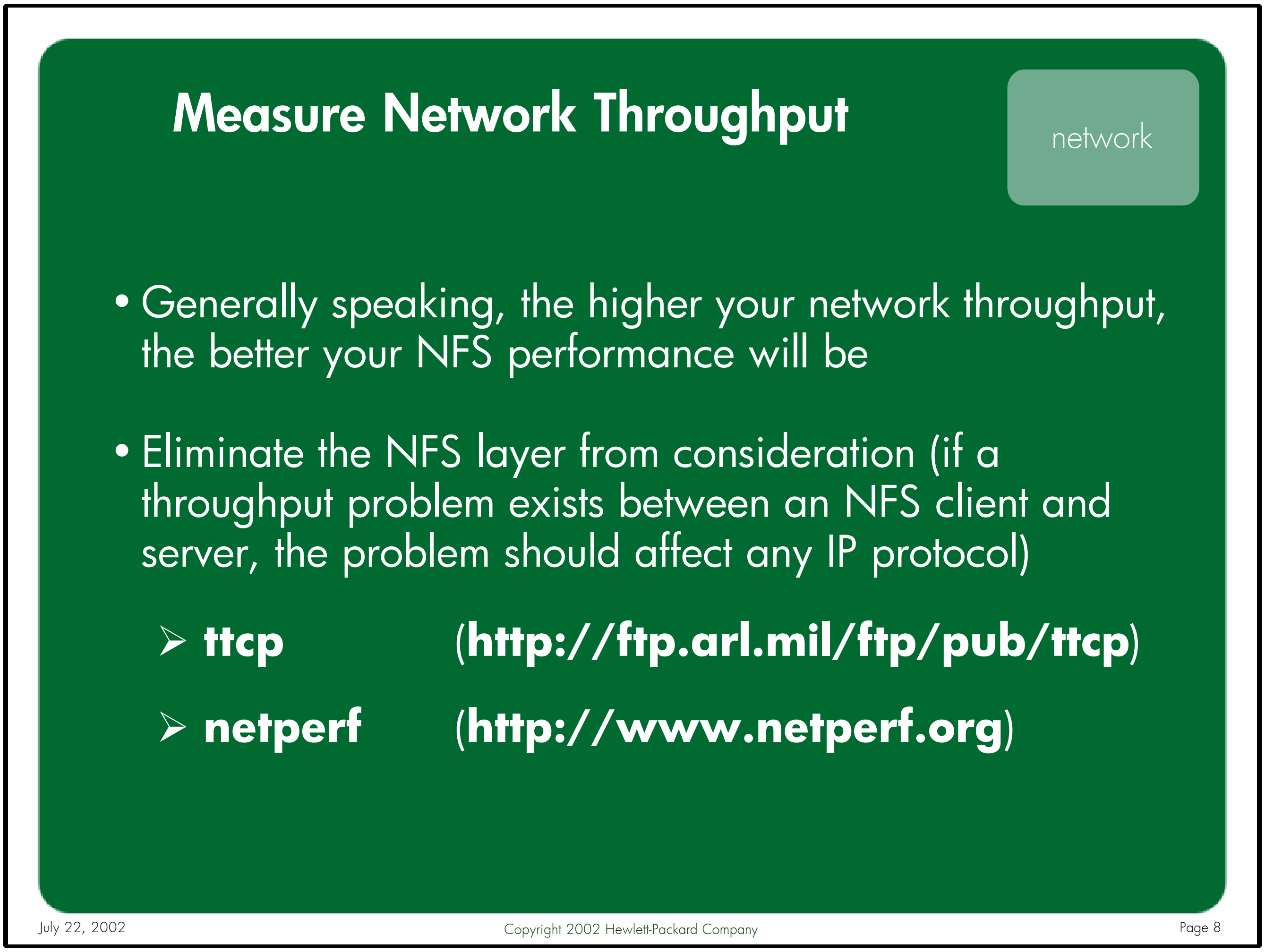
- Familiarize yourself with the physical layout (i.e. how many network hops separate the client and server?)
  - ➢ **OpenView Network Node Manager**
  - ➢ **traceroute**
  - ➢ **ping -o**
- MTU sizes of the various network hops
  - ➢ **netstat -in**

July 22, 2002

Page 7

Notes: An important early step in troubleshooting any NFS performance issue is to learn as much as possible about the physical layout of the underlying network topology. How many network hops (i.e. bridges, hubs, routers, switches, etc.) exist between the client and the server? What is the speed of each link separating these systems? Do packets sent from the client to the server take the same route through the network as the packets sent from the server back to the client?

While the network administrator should be the best source of knowledge about the layout and capabilities of the underlying network, even they are not always up-to-date on the current state of the network. In many large corporate environments, the physical network is constantly evolving as new equipment replaces old, new backbone technologies replace antiquated ones, new systems are added to existing networks, new subnets are created, etc. Whenever there is any uncertainty as to the physical layout of the network separating the NFS clients and servers, a network layout analysis should be performed.



Measure Network Throughput

network

- Generally speaking, the higher your network throughput, the better your NFS performance will be
- Eliminate the NFS layer from consideration (if a throughput problem exists between an NFS client and server, the problem should affect any IP protocol)
  - ➢ **ttcp** (**http://ftp.arl.mil/ftp/pub/ttcp**)
  - ➢ **netperf** (**http://www.netperf.org**)

July 22, 2002

Page 8

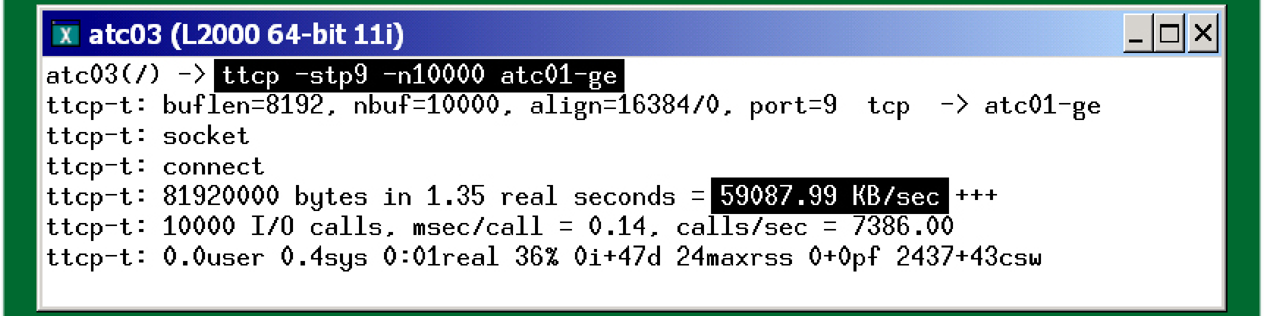
Notes: Once the layout of the physical network is understood, the next step in validating your network is to measure the throughput of the connection separating the client and server. Generally speaking, the faster your network throughput, the better your NFS performance will be.

When testing the performance of the network for NFS purposes, it is essential to eliminate the NFS layer from consideration by simply testing the network transport layer using non-NFS protocols. If a throughput problem exists between an NFS client and server, the problem would affect any network-based application, not just NFS. Similarly, when attempting to characterize network throughput, it is important to eliminate any impact of the local filesystems. It is therefore necessary to select measurement tools that are not dependent upon NFS or other filesystem resources.

Several tools exist to help system and network administrators measure the throughput of their network connections. ttcp and netperf are two of the most prominent tools.

ttcp

network



```
atc03(/) -> ttcp -stp9 -n10000 atc01-ge
ttcp-t: buflen=8192, nbuf=10000, align=16384/0, port=9  tcp  -> atc01-ge
ttcp-t: socket
ttcp-t: connect
ttcp-t: 81920000 bytes in 1.35 real seconds = 59087.99 KB/sec +++
ttcp-t: 10000 I/O calls, msec/call = 0.14, calls/sec = 7386.00
ttcp-t: 0.0user 0.4sys 0:01real 36% 0i+47d 24maxrss 0+0pf 2437+43csw
```

The above ttcp output shows this NFS server can send 80MB of TCP/IP data to the client's discard port (9) in 1.35 seconds (approximately ~59MB/sec).

Notes: ttcp is a simple, lightweight program that measures the throughput of any network connection without relying on the filesystem layer. It can generate either UDP or TCP traffic, and it can be configured to send small or large sized packets to simulate the behavior of different applications.

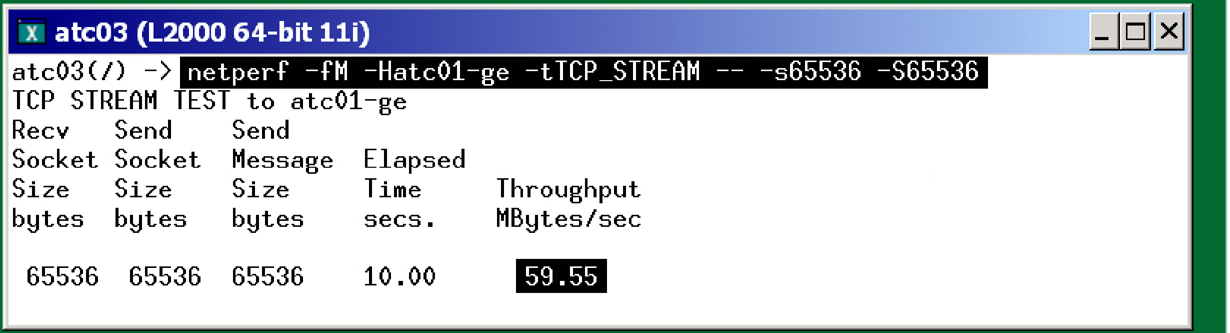
Mike Muuss (the author of the "ping" command) and Terry Slattery, of the US Army Ballistics Research Lab (BRL), developed the ttcp (Test TCP Connection) program in the early 1980's. The program now resides in the public domain and is freely available to download from **http://ftp.arl.mil/ftp/pub/ttcp**. Also available from this site is a man page for the ttcp program, which documents the many available command-line arguments.

It is important to measure the network throughput going in both directions (i.e. client to server, server to client) to make sure that the bi-directional performance is comparable.

The ttcp results should not be misinterpreted as potential NFS throughput values. They merely illustrate the capabilities of the specific network link being tested.

netperf

network



```
atc03(/) -> netperf -fM -Hatc01-ge -tTCP_STREAM -- -s65536 -S65536
TCP STREAM TEST to atc01-ge
Recv   Send    Send
Socket Socket  Message  Elapsed
Size   Size    Size     Time     Throughput
bytes  bytes   bytes    secs.    MBytes/sec

 65536  65536  65536    10.00      59.55
```

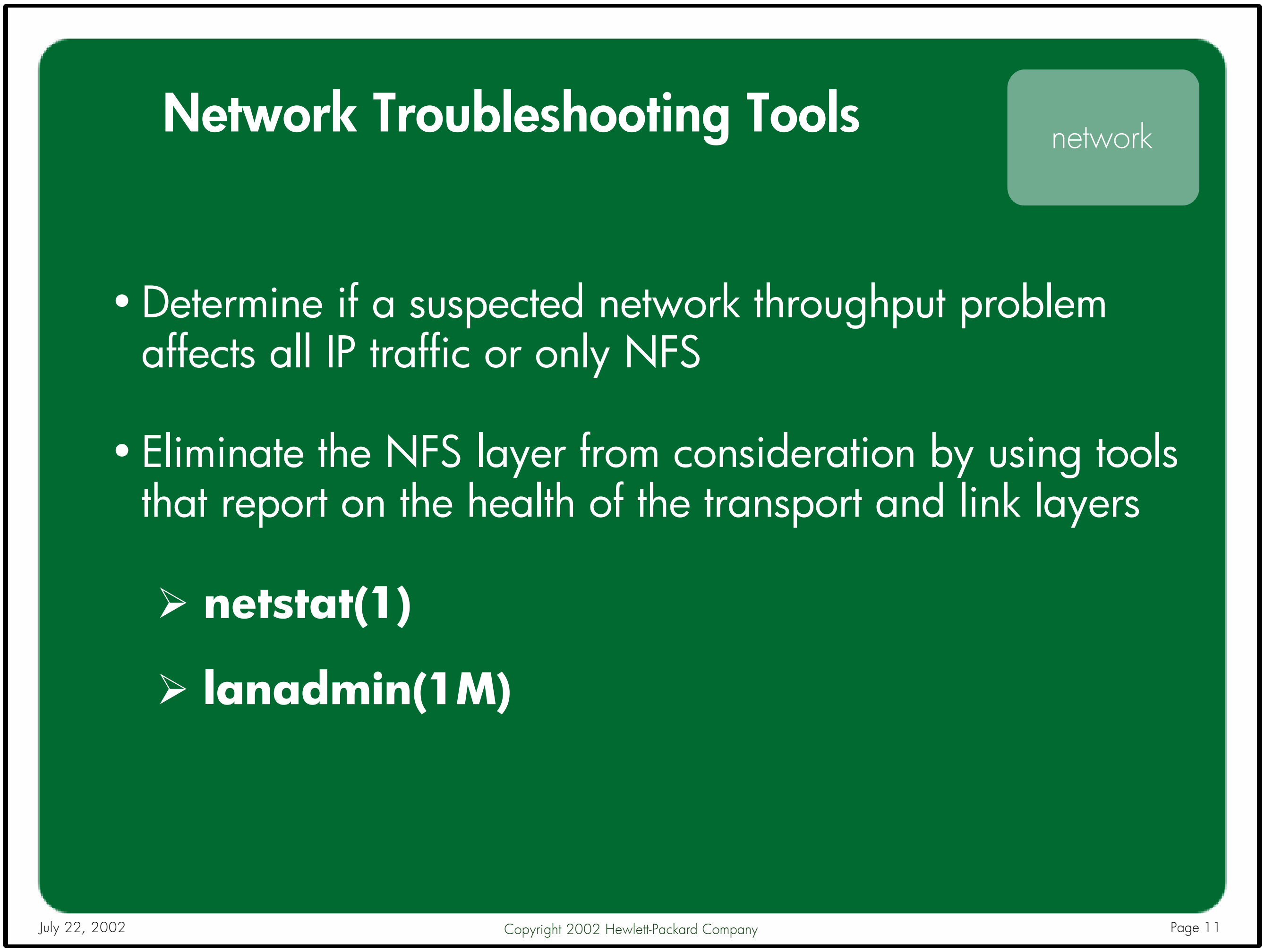
The above netperf output shows this NFS server was able to send TCP/IP data to the client at a sustained rate of ~59.5MB/sec during the 10 second test.

Notes: Netperf is a benchmark utility that can measure the performance of many different types of networks. Like ttcp, netperf measures throughput without relying on any filesystem resources. The environments currently measurable by netperf include:

- TCP and UDP via BSD Sockets
- DLPI
- UNIX Domain Sockets

Rick Jones, of HP's Infrastructure Solutions and Partners group, developed netperf back in 1993, and he has continued to add new features and capabilities to the program as new networking technologies became available. The best source of information is the official netperf website: **http://www.netperf.org**.

As with ttcp, any netperf results should not be interpreted as potential NFS throughput values.



Network Troubleshooting Tools

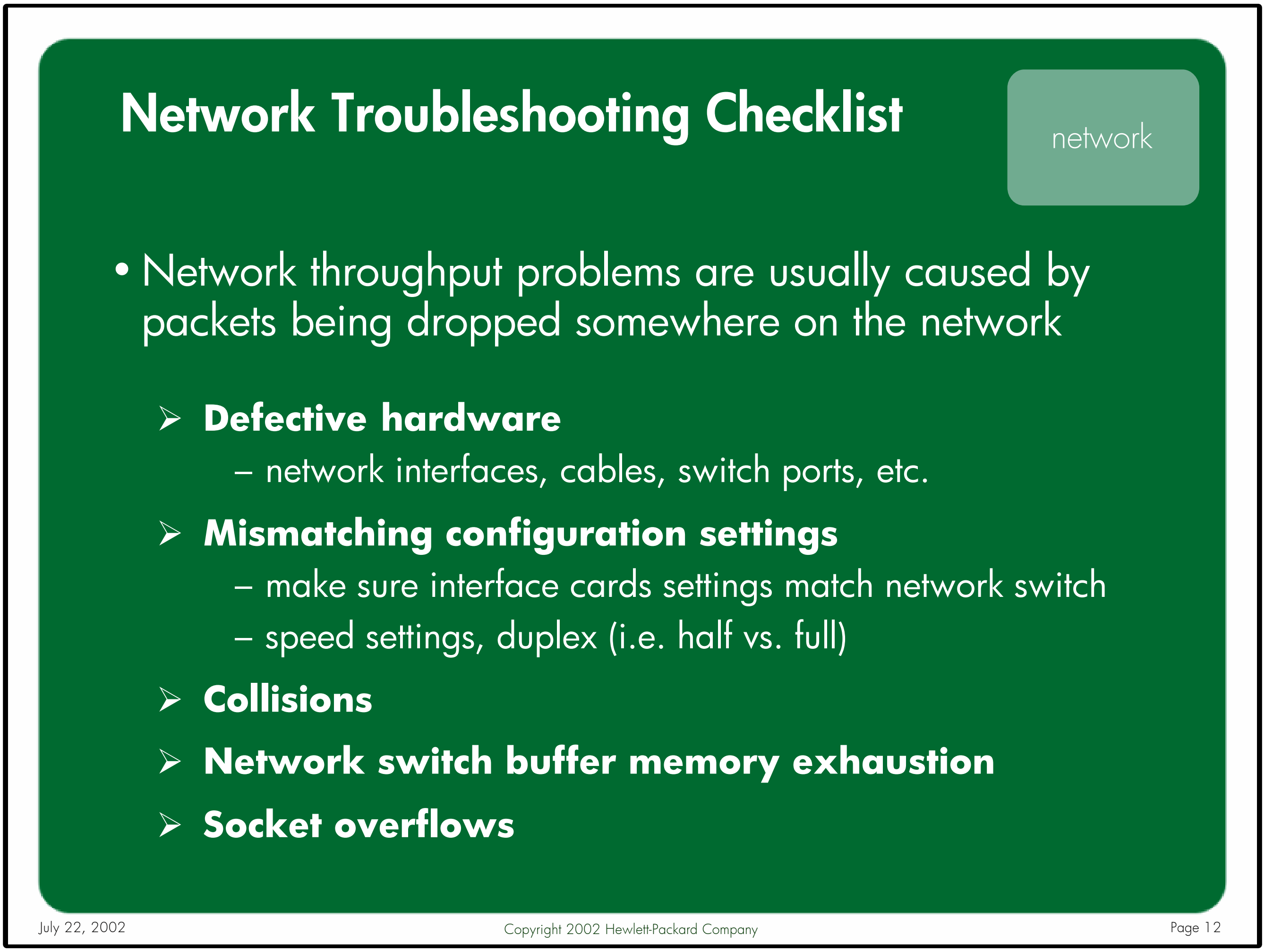
network

- Determine if a suspected network throughput problem affects all IP traffic or only NFS
- Eliminate the NFS layer from consideration by using tools that report on the health of the transport and link layers
  - ➢ **netstat(1)**
  - ➢ **lanadmin(1M)**

July 22, 2002

Page 11

Notes: The goal for this phase of the investigation is to determine if a network throughput problem is affecting all IP traffic or only NFS. HP provides a number of tools to help detect and analyze network problems. Two frequently used tools for troubleshooting the network layer are netstat(1) and lanadmin(1M).



Network Troubleshooting Checklist

network

- Network throughput problems are usually caused by packets being dropped somewhere on the network
  - **Defective hardware**
    - network interfaces, cables, switch ports, etc.
  - **Mismatching configuration settings**
    - make sure interface cards settings match network switch
    - speed settings, duplex (i.e. half vs. full)
  - **Collisions**
  - **Network switch buffer memory exhaustion**
  - **Socket overflows**

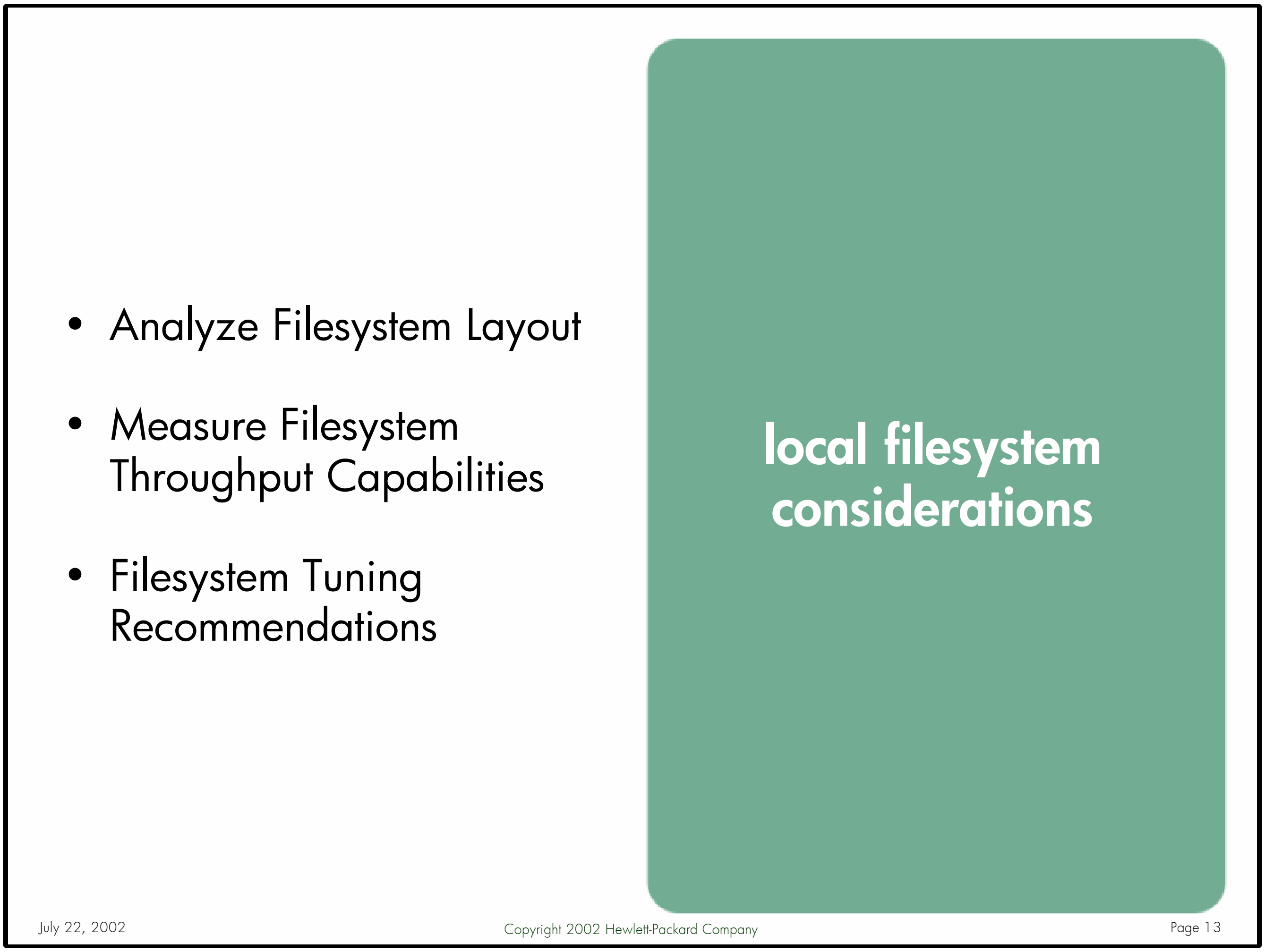
July 22, 2002

Page 12

Notes: In most cases, network throughput problems are caused by packets being dropped somewhere on the network. Some of the more common reasons why packets or requests are dropped include:

- Defective hardware (i.e. network interfaces, cables, switch ports, etc.)
- Mismatching configuration settings between interface cards and network switch equipment, such as the speed and duplex settings (i.e. half vs. full)
- Collisions (this is typically not a problem with switched networks)
- Switch buffer memory exhaustion
- Socket overflows

In some cases, the only tools that can detect these types of problems are external analyzers and reporting tools specific to your network hardware.



local filesystem considerations

- Analyze Filesystem Layout
- Measure Filesystem Throughput Capabilities
- Filesystem Tuning Recommendations

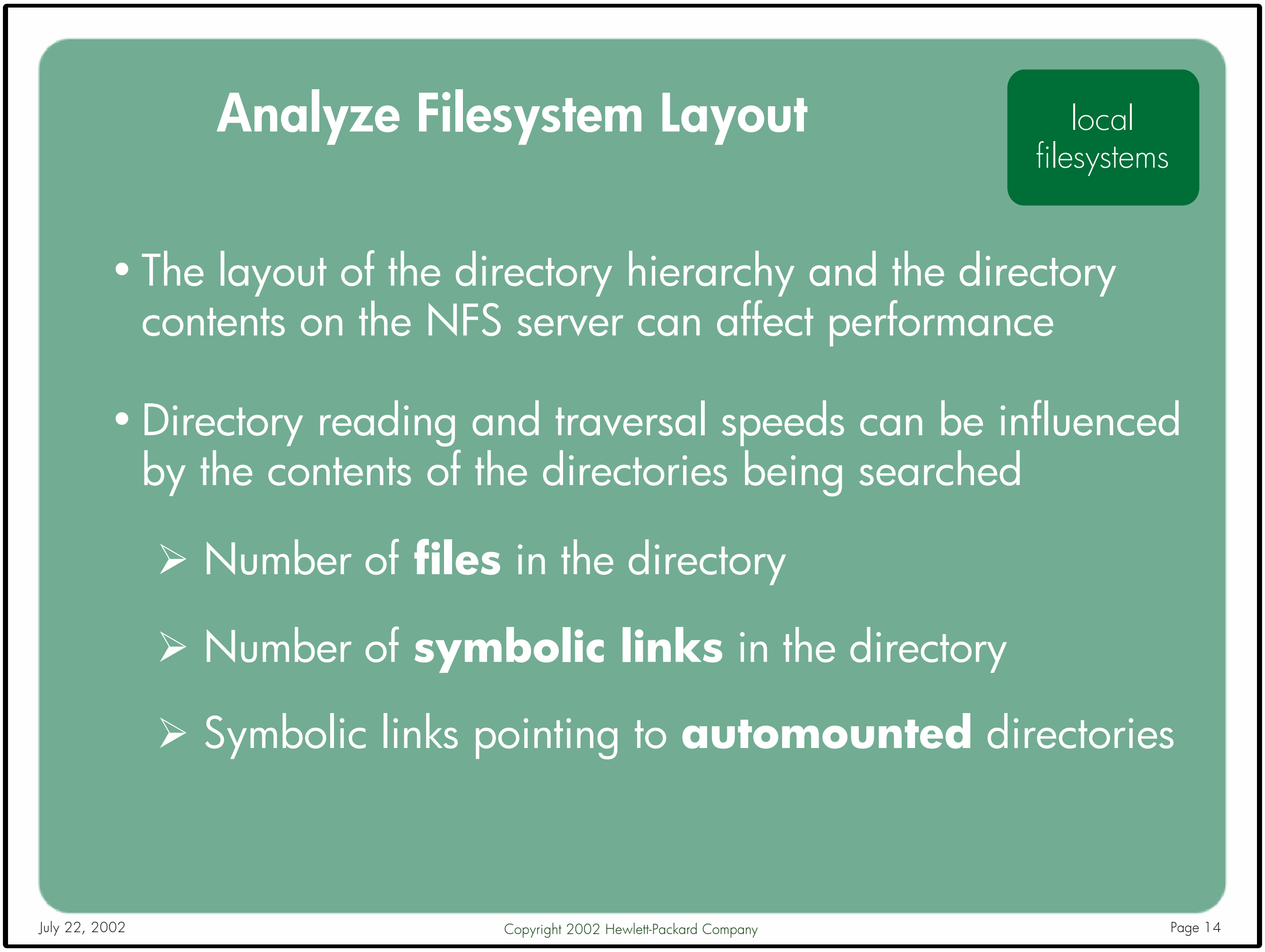
July 22, 2002

Page 13

Notes: The performance of the local filesystems (both on the NFS client and server) can have just as big an impact to overall NFS performance as the underlying network. Again, this shouldn't be a surprise since NFS is an acronym for "Network **File System**". Therefore, when faced with an NFS performance issue, it is a good idea to perform some "sanity checks" on the underlying filesystems involved before beginning a detailed investigation into a specific area of NFS.

This section will not be describing the numerous available disk and filesystem technologies (such as disk striping, RAID, disk arrays, etc.). NFS servers are allowed to export any local filesystems, and NFS typically performs better with a faster underlying disk subsystem. There is a wealth of information about the latest and greatest filesystem technologies available from HP's IT Resource Center: **http://itrc.hp.com** and HP's documentation repository: **http://docs.hp.com**.

What will be described is a recommended methodology and set of tools available for understanding the physical layout of your filesystems, measuring their throughput, and performing routine filesystem maintenance and troubleshooting.



Analyze Filesystem Layout

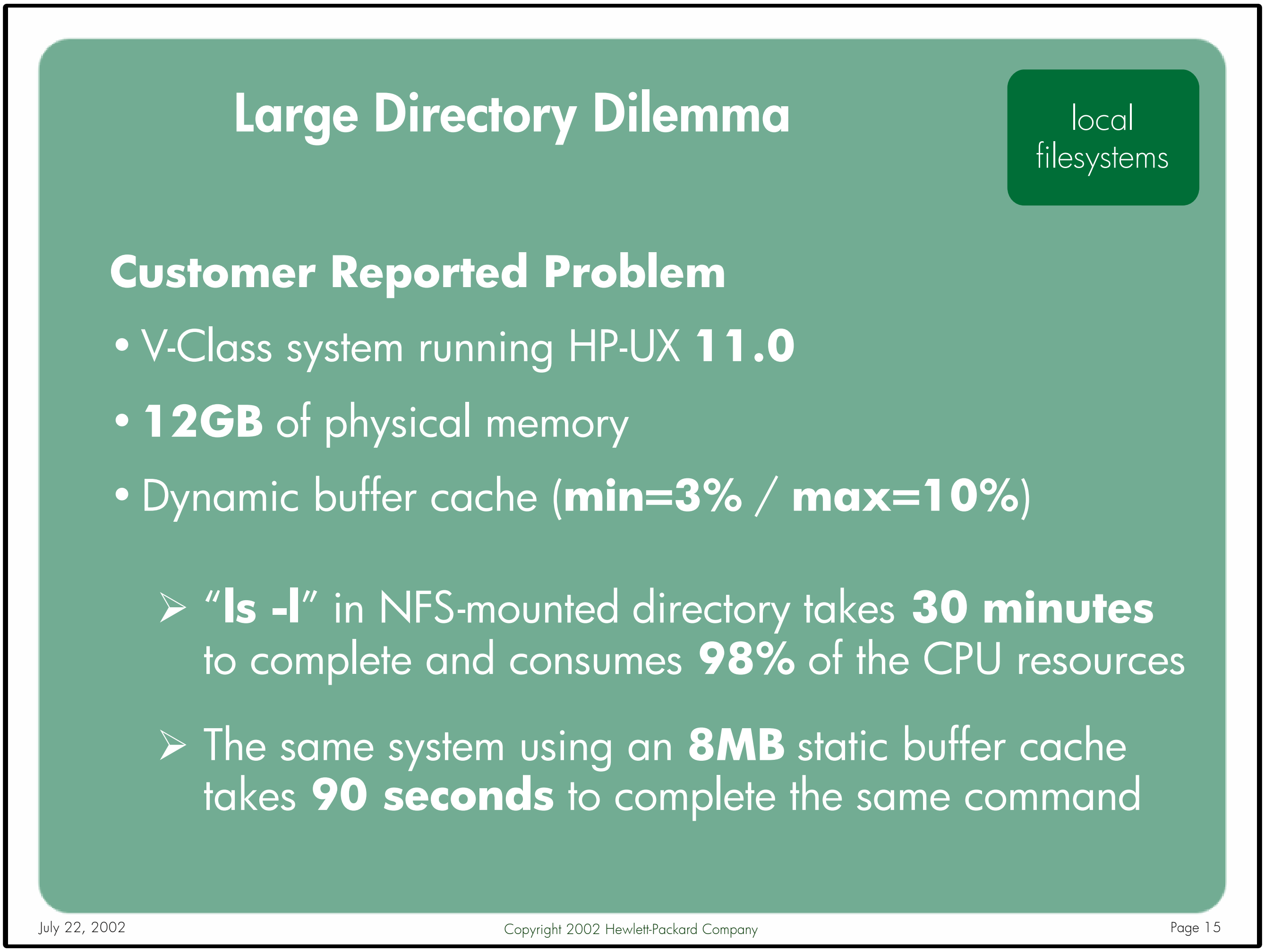
local filesystems

- The layout of the directory hierarchy and the directory contents on the NFS server can affect performance
- Directory reading and traversal speeds can be influenced by the contents of the directories being searched
  - ➢ Number of **files** in the directory
  - ➢ Number of **symbolic links** in the directory
  - ➢ Symbolic links pointing to **automounted** directories

Notes: When retrieving the contents of an NFS mounted directory, the client will issue either a combination of READDIR and LOOKUP calls or a series of READDIRPLUS requests. The larger the directory, the more calls needed to retrieve the contents. Symbolic links require still more calls because a READLINK is sent to retrieve the contents of the link. Symbolic links that point to automount managed directories are rarely a good idea and should be avoided.

Whenever possible, you should try to "flatten out" the directory hierarchy of your NFS filesystems to achieve a reasonable balance between the number of directories and the number of files in those directories. The goal is to minimize the number of NFS calls required by the client to retrieve the contents of any given directory, while at the same time to arrange the files in such a way as to reduce the number of directories needing to be searched by a user for a typical operation.

Some applications have restrictions on where data files can reside. Be sure to check with your application provider if you have any concerns about supported directory configurations before re-distributing the files used by an application.



Large Directory Dilemma

local filesystems

Customer Reported Problem

- V-Class system running HP-UX **11.0**
- **12GB** of physical memory
- Dynamic buffer cache (**min=3%** / **max=10%**)

➢ "**ls -l**" in NFS-mounted directory takes **30 minutes** to complete and consumes **98%** of the CPU resources

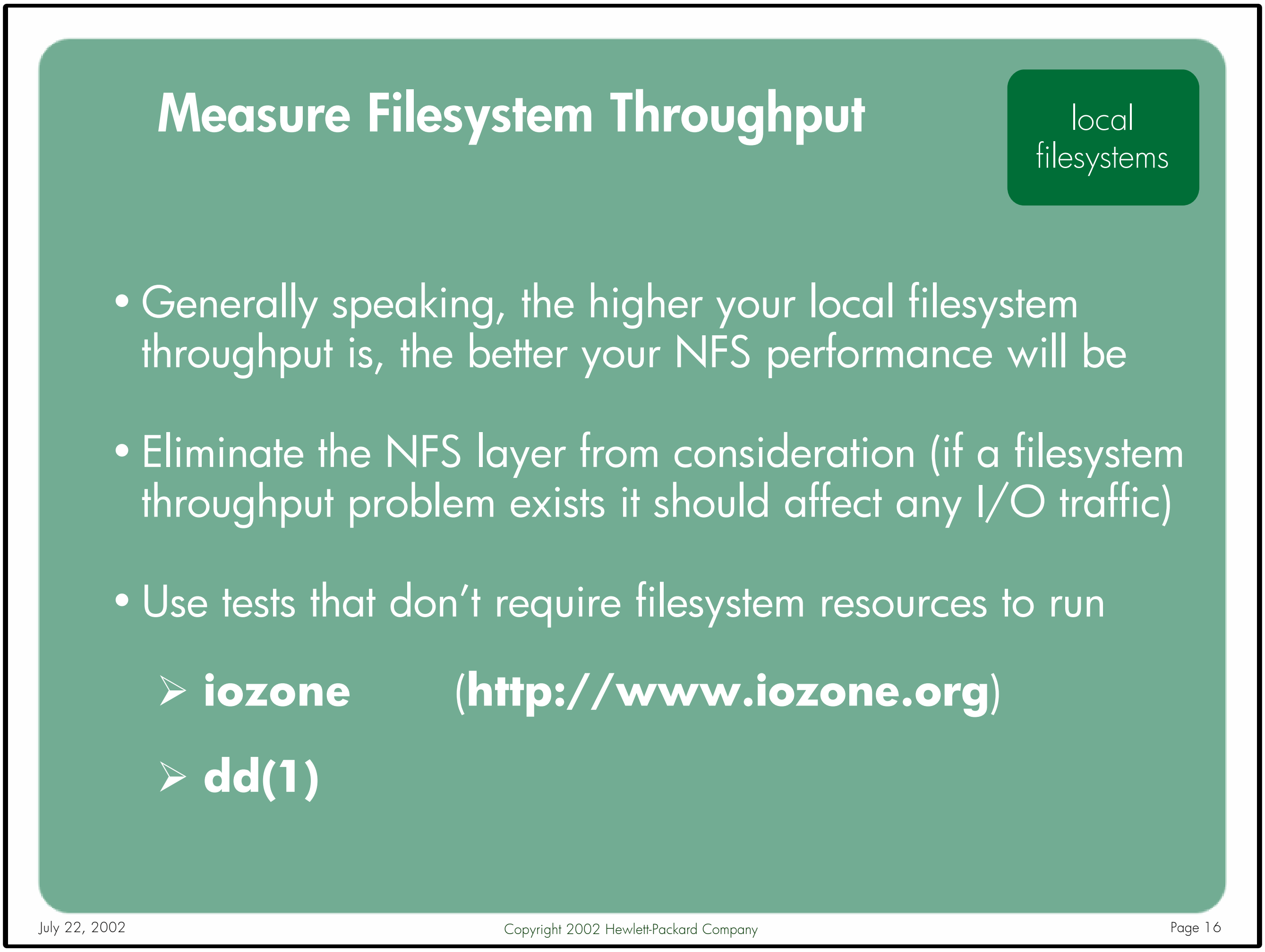
➢ The same system using an **8MB** static buffer cache takes **90 seconds** to complete the same command

July 22, 2002 Copyright 2002 Hewlett-Packard Company Page 15

Notes: Question: Why would a simple "ls -l" command issued in a large NFS-mounted directory take a long time to complete and consume huge amounts of system resources when a large buffer cache is configured on the client?

Answer: When reading an NFS-mounted directory, an rnode structure must be allocated for every entry in the directory, and only a fixed number of rnodes are available on the client (sized by nrnode, which in turn is sized by ncsize or ninode). Once all free rnodes are used, the system must begin reusing existing rnodes. In order to reuse an rnode, the system must first invalidate all pages associated with the rnode from the buffer cache. This requires the client to perform a serial search of the entire buffer cache looking for pages to invalidate. When traversing a directory containing many thousands of entries, the system will need to search buffer cache many times. With a large buffer cache configured, these serial searches can consume high amounts of CPU resources.

Fortunately, the buffer cache subsystem has be redesigned in HP-UX 11i to eliminate this problem. Refer to page 108 for more information.



Measure Filesystem Throughput

local filesystems

- Generally speaking, the higher your local filesystem throughput is, the better your NFS performance will be
- Eliminate the NFS layer from consideration (if a filesystem throughput problem exists it should affect any I/O traffic)
- Use tests that don't require filesystem resources to run
  - ➢ **iozone** (**http://www.iozone.org**)
  - ➢ **dd(1)**

July 22, 2002

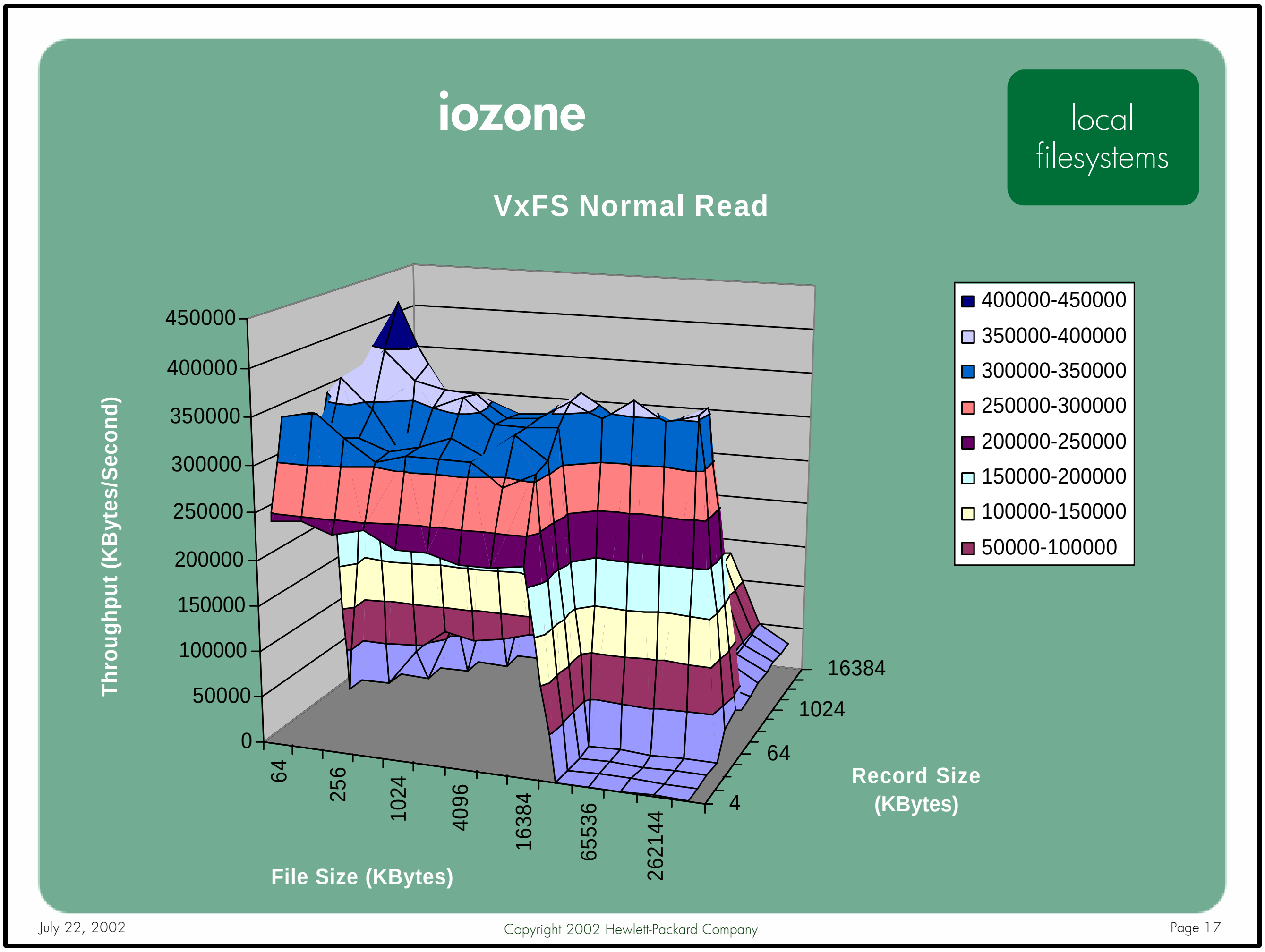
Page 16

Notes: Once the layout of the server's filesystems has been analyzed and optimized where possible, the next step in validating your disk subsystems is to measure the throughput of the client's and server's local filesystems. Generally speaking, the faster your underlying disk subsystems, the better your NFS performance will be.

An approach similar to the network testing described earlier should be used, where NFS is removed from consideration. If a filesystem throughput problem exists on the NFS client or server, the problem should affect any I/O traffic, not just NFS.

The testing utilities should allow the administrator to isolate one type of filesystem I/O at a time (i.e. test read throughput then write throughput) before mixing I/O types. Also, the tests themselves should not require any filesystem resources to run as this could affect the throughput results (i.e. don't test filesystem write performance with a test that requires reading from a filesystem).

Two of the available tools that meet these requirements are iozone and dd(1).



iozone

local filesystems

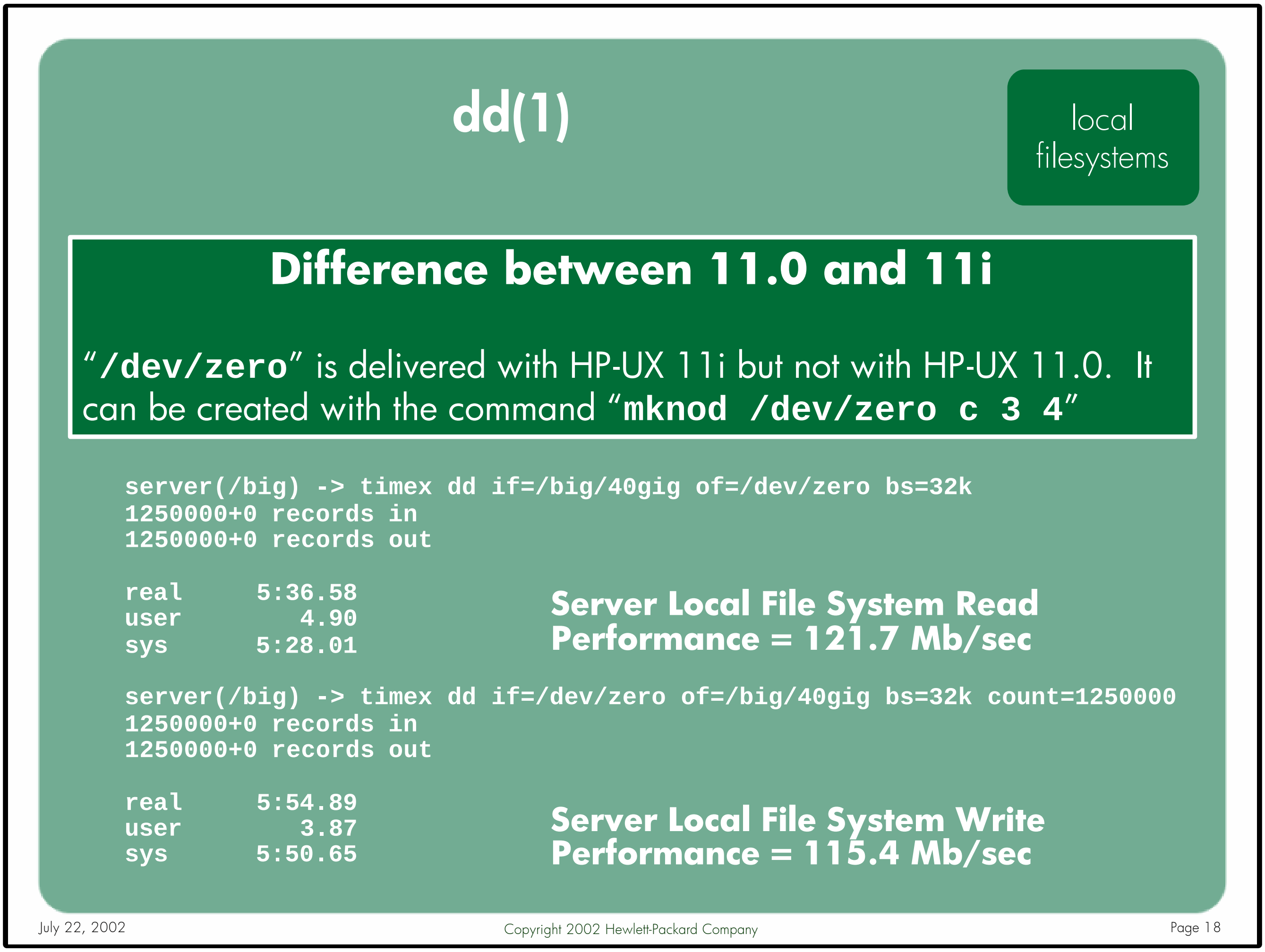
July 22, 2002

Page 17

Notes: One of the better filesystem performance benchmark utilities available is iozone. It can perform a wide variety of I/O operations, including: read, re-read, write, re-write, random read and write, etc. It also has many advanced capabilities, such as: testing with different record lengths, locking the target file during reads and writes, unmounting/re-mounting filesystems between tests, etc. It can even generate test results in Microsoft Excel™ format to simplify graphing the throughput numbers.

Don Capps, of the HP's Technical Computing Division, is one of the authors of iozone. The best source of information about iozone is the dedicated website: **http://www.iozone.org**. This site holds the latest source code, which is available at no cost, as well as program documentation and sample output graphs. The users guide includes a detailed explanation of the available command-line options, and instructions for generating Excel ™ charts using the data you collect.

It is important to measure the local filesystem capabilities of both the client and the server since both are usually involved in NFS activities. As with the networking tools described earlier, iozone results do not directly correlate to potential NFS throughput numbers. They merely reflect the capabilities of the local filesystem.



dd(1)

local filesystems

Difference between 11.0 and 11i

"**/dev/zero**" is delivered with HP-UX 11i but not with HP-UX 11.0. It can be created with the command "**mknod /dev/zero c 3 4**"

```
server(/big) -> timex dd if=/big/40gig of=/dev/zero bs=32k
1250000+0 records in
1250000+0 records out

real     5:36.58
user        4.90
sys      5:28.01
```

Server Local File System Read Performance = 121.7 Mb/sec

```
server(/big) -> timex dd if=/dev/zero of=/big/40gig bs=32k count=1250000
1250000+0 records in
1250000+0 records out

real     5:54.89
user        3.87
sys      5:50.65
```

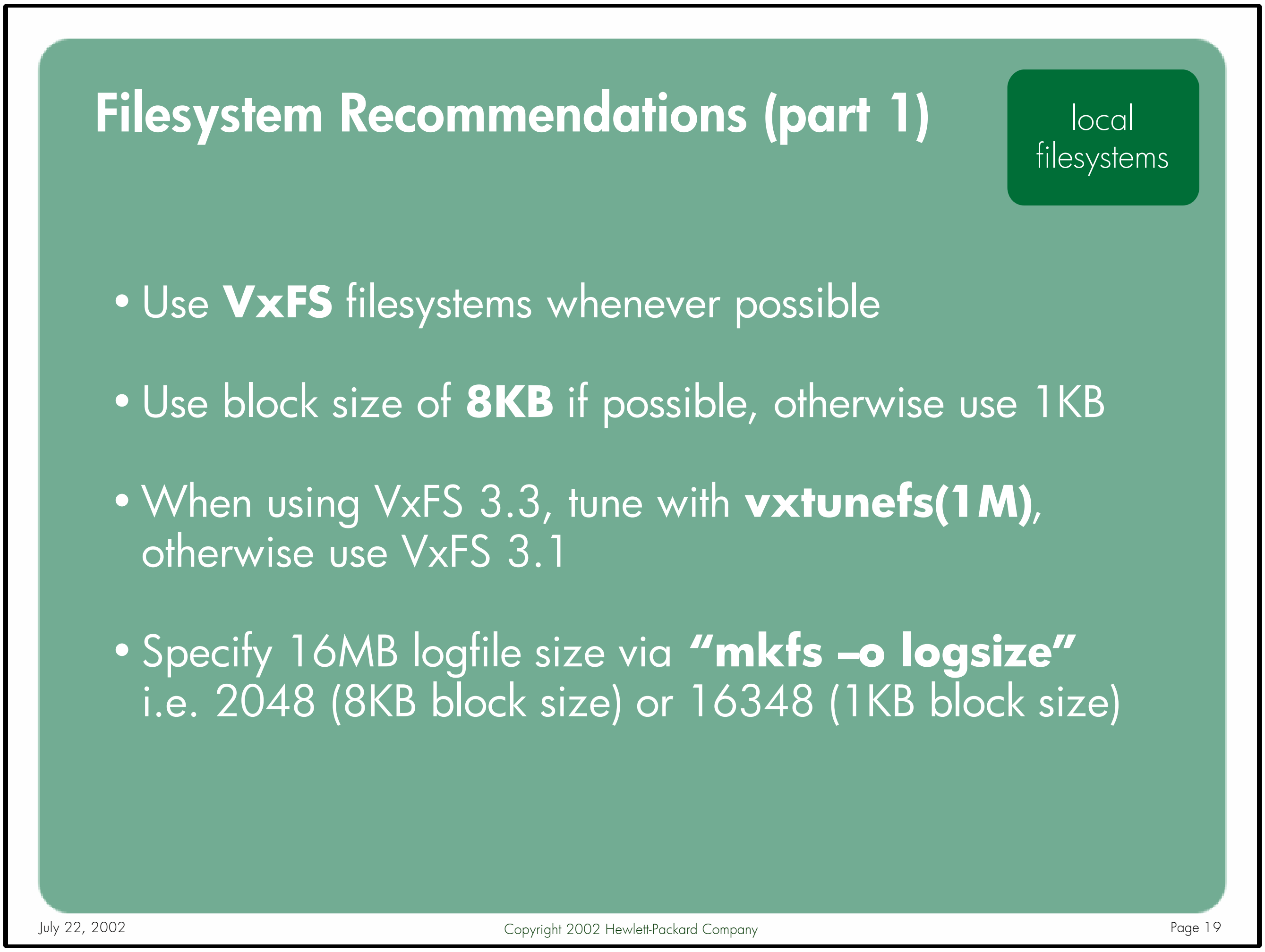
Server Local File System Write Performance = 115.4 Mb/sec

July 22, 2002 Page 18

Notes: The dd(1) command was originally developed to facilitate the duplication of disk drives by transferring their contents to magnetic tape. However, since it has the ability to read from any input device and write to any output device, it is ideal for generating disk I/O requests to specific filesystems. Also, since it allows the user to control the record size used for the I/O transactions, dd can be used to simulate a wide variety of applications.

When testing filesystem throughput, dd can be directed to read from or write to the special file `/dev/zero`, thereby avoiding any physical disk read and write requests. On 11.0 systems `/dev/zero` does not exist, but can be built via the mknod(1M) command. Once built, reads from `/dev/zero` always returns a buffer full of zeroes. The file is of infinite length, so the reading application needs to specify the desired amount of data. Writes to `/dev/zero` are always successful, but the written data is discarded.

NOTE: Avoid using the "ibs" and "obs" options to specify input and output block sizes. When these options are specified, dd performs internal data conversions that can dramatically impact the throughput numbers. Use the "bs" option instead.



Filesystem Recommendations (part 1)

local filesystems

- Use **VxFS** filesystems whenever possible
- Use block size of **8KB** if possible, otherwise use 1KB
- When using VxFS 3.3, tune with **vxtunefs(1M)**, otherwise use VxFS 3.1
- Specify 16MB logfile size via **"mkfs –o logsize"** i.e. 2048 (8KB block size) or 16348 (1KB block size)

July 22, 2002

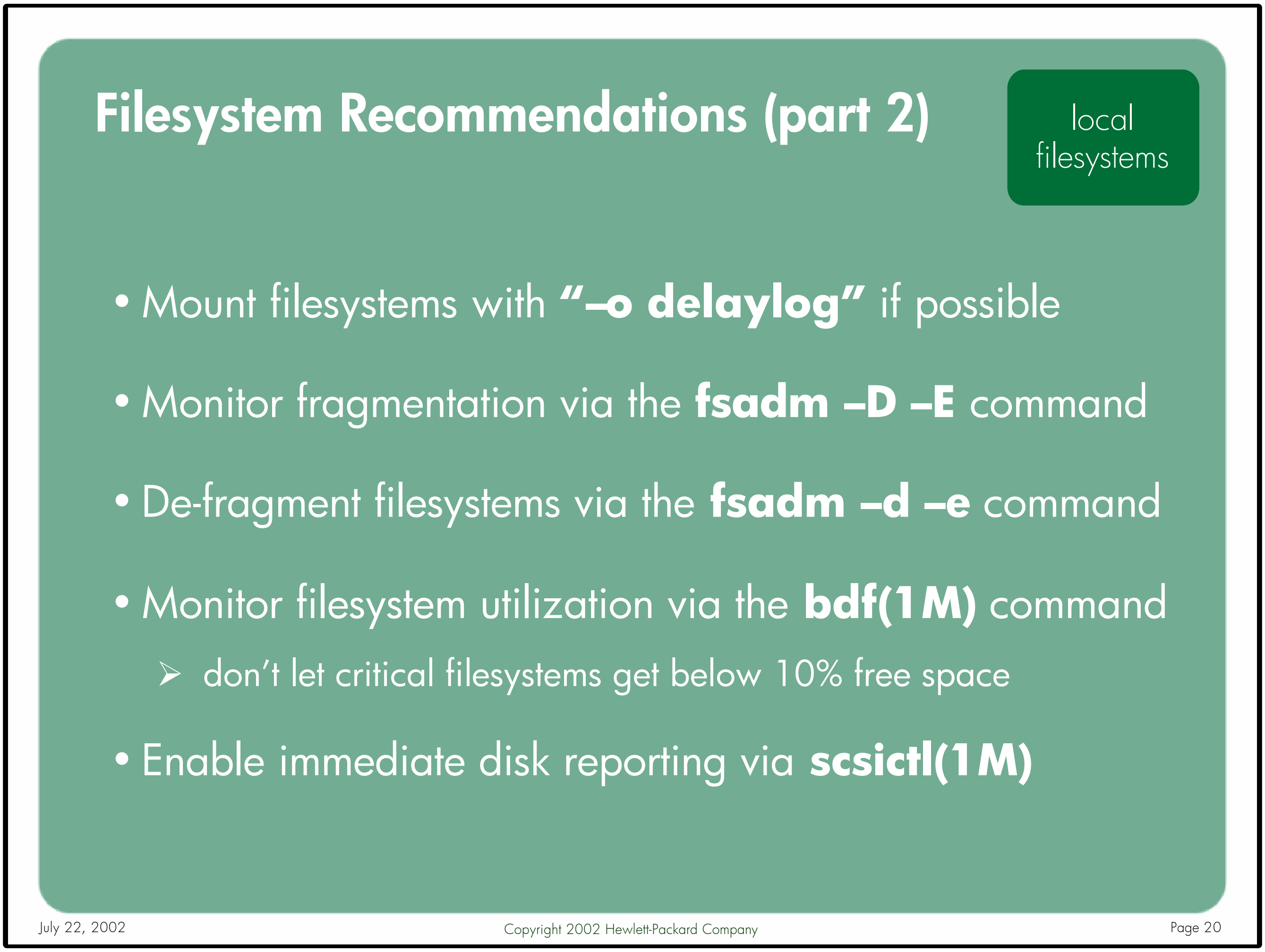
Page 19

Notes: VxFS filesystems are recommended over HFS filesystems on both NFS clients and servers. VxFS filesystems have a huge advantage over their HFS counterparts because of the way VxFS interacts with buffer cache. VxFS filesystems track their buffer cache usage on a file-by-file basis, which allows the kernel to invalidate VxFS files from the cache much faster than HFS.

An 8KB block size should be used if possible. While this might result in wasted space if the filesystem is comprised mainly of very small files, the 8KB block size allows VxFS to perform "block" reads when servicing NFS read requests.

If you wish to use VxFS 3.3 in your environment, you should familiarize yourself with the vxtunefs(1M) command. In many situations, an un-tuned VxFS 3.3 filesystem will perform much worse than a similar VxFS 3.1 filesystem.

When building the VxFS filesystems on your NFS server, the maximum intent log size (16MB) should be specified, as this allows the largest number of requests to be pending in the logfile before any intent log maintenance is required by the kernel.



Filesystem Recommendations (part 2)

local filesystems

- Mount filesystems with **"–o delaylog"** if possible
- Monitor fragmentation via the **fsadm –D –E** command
- De-fragment filesystems via the **fsadm –d –e** command
- Monitor filesystem utilization via the **bdf(1M)** command
  - ➢ don't let critical filesystems get below 10% free space
- Enable immediate disk reporting via **scsictl(1M)**

July 22, 2002

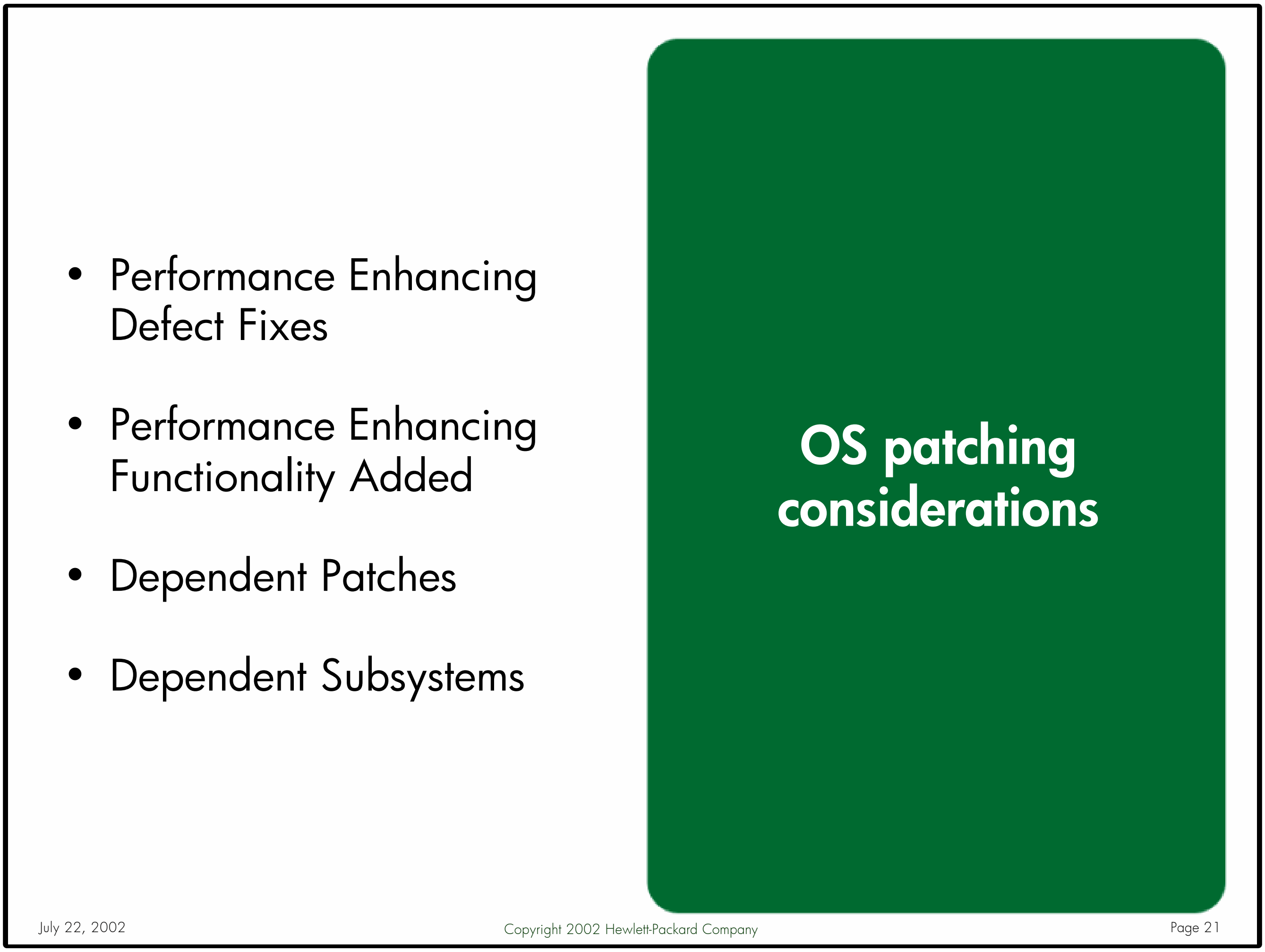
Page 20

Notes: The "–o delaylog" mount option should be specified if your environment allows it. In delaylog mode, some system calls return before the intent log is written. This improves the performance of the system, but some changes are not guaranteed until a short time later when the intent log is written.

VxFS filesystems can get fragmented with heavy usage over time. Performance can be severely impacted on very fragmented volumes. It is therefore important to monitor the fragmentation level of your critical exported filesystems and periodically de-fragment them via the fsadm(1M) command.

It is also important to monitor the utilization of critical filesystems. Care should be taken to ensure that these filesystem resources are not exhausted.

For those SCSI devices where immediate disk reporting is available (such as the SC10 disks in an FC-60 array), use the scsictl(1M) command to enable this feature.



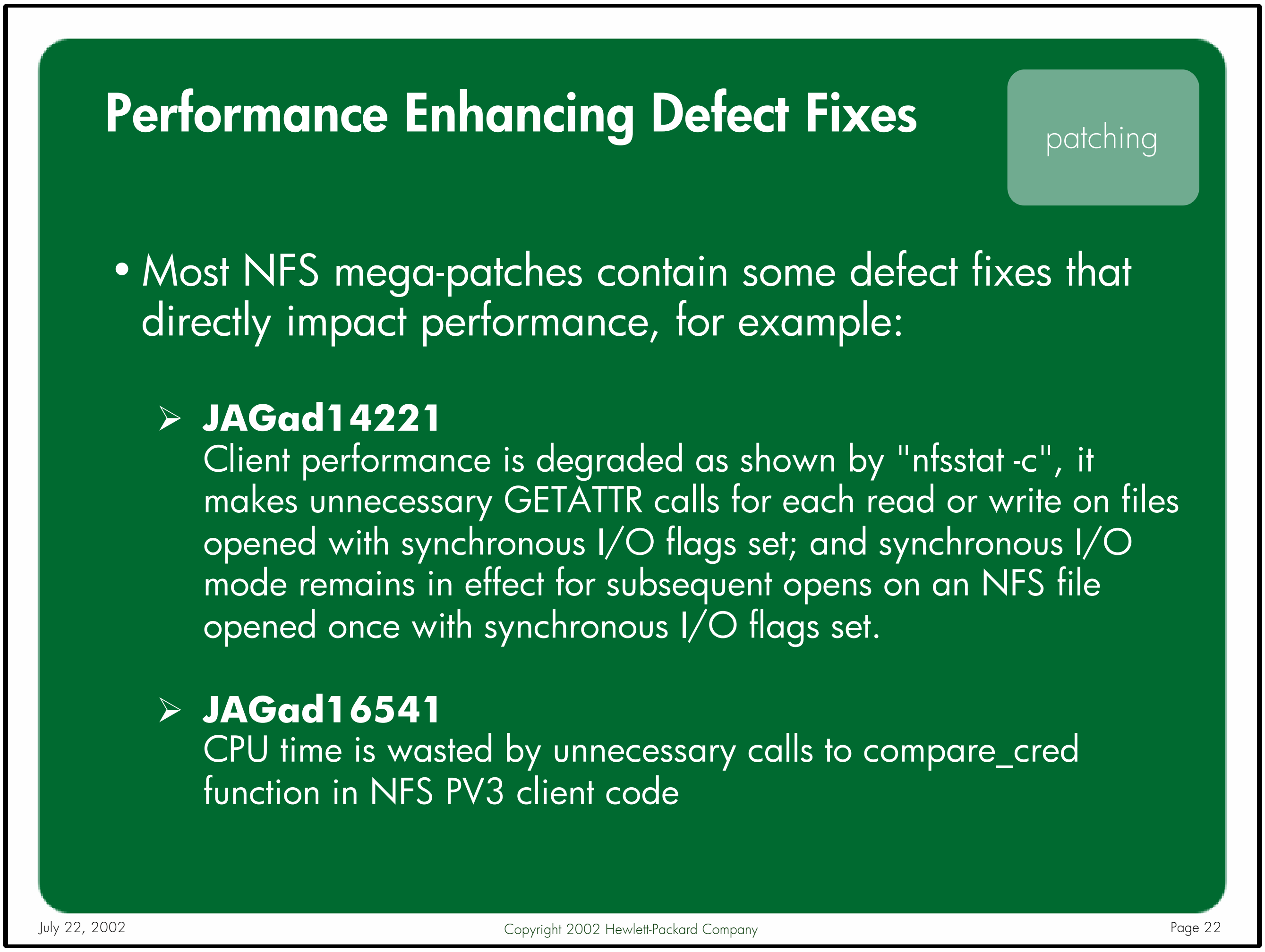
OS patching considerations

- Performance Enhancing Defect Fixes
- Performance Enhancing Functionality Added
- Dependent Patches
- Dependent Subsystems

July 22, 2002 Copyright 2002 Hewlett-Packard Company Page 21

Notes: NFS continues to evolve over time. The ONC+ offering that shipped on HP-UX 11i is a far superior product in terms of functionality, stability, and performance compared to the ONC+ product that shipped with the HP-UX 11.0 release. This does not imply that improvements in NFS are only made as new versions of the operating system are introduced – far from it. HP's NFS lab continually strives to improve its products on the existing supported OS releases by distributing patches containing defect fixes and functionality enhancements.

Since NFS has so many dependencies on other components of the system, it comes as no surprise that the list of dependent patches is lengthy. It is very important to keep the patch levels of all the underlying subsystems (i.e. network, buffer cache, transport, etc.) current since a defect in those subsystems can negatively impact NFS behavior and performance.

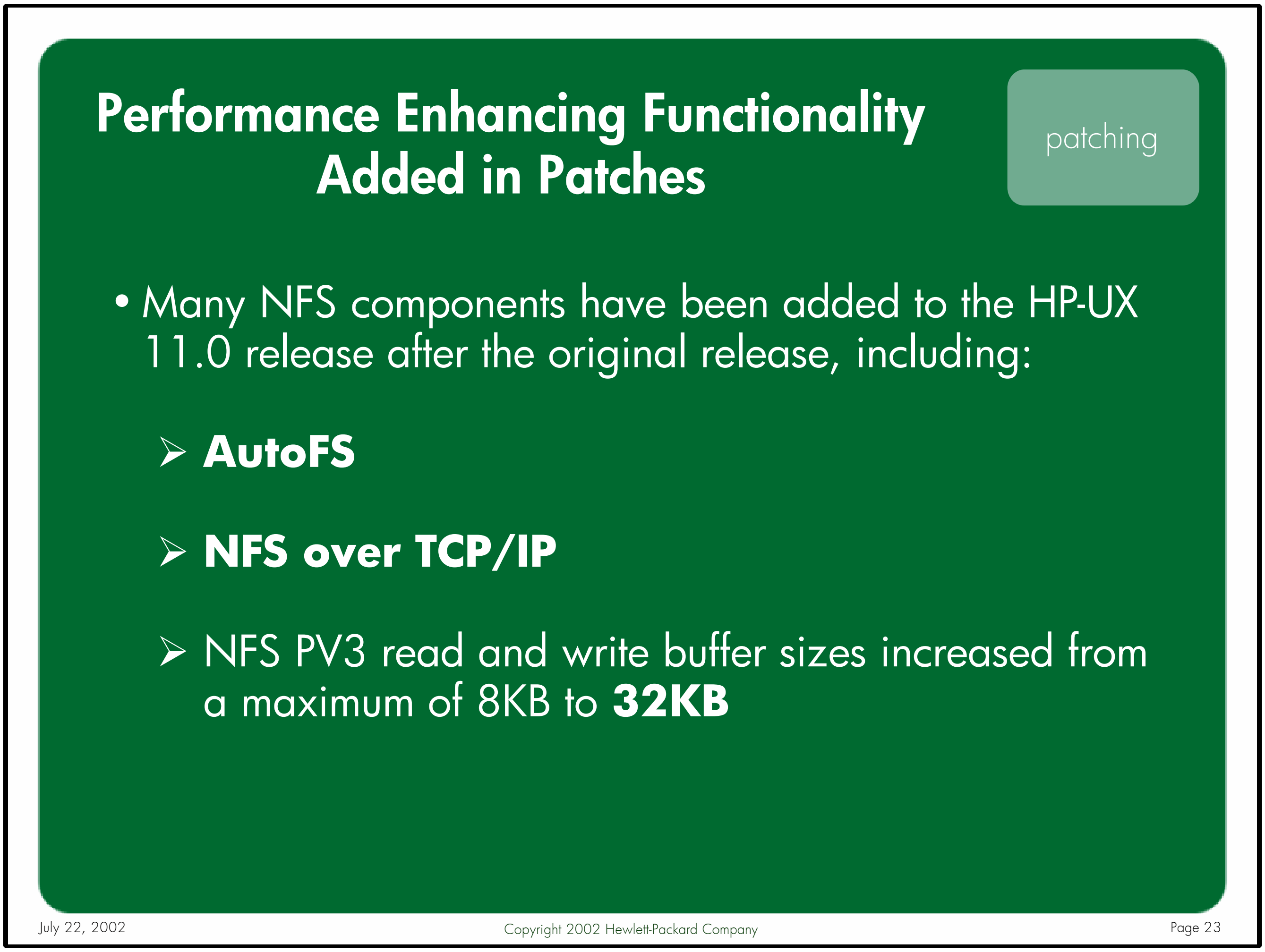


Performance Enhancing Defect Fixes

patching

- Most NFS mega-patches contain some defect fixes that directly impact performance, for example:
  - **JAGad14221**
    Client performance is degraded as shown by "nfsstat -c", it makes unnecessary GETATTR calls for each read or write on files opened with synchronous I/O flags set; and synchronous I/O mode remains in effect for subsequent opens on an NFS file opened once with synchronous I/O flags set.
  - **JAGad16541**
    CPU time is wasted by unnecessary calls to compare_cred function in NFS PV3 client code

Notes: The NFS lab releases mega-patches quarterly. These patches frequently include defect fixes that impact performance as well as behavior.



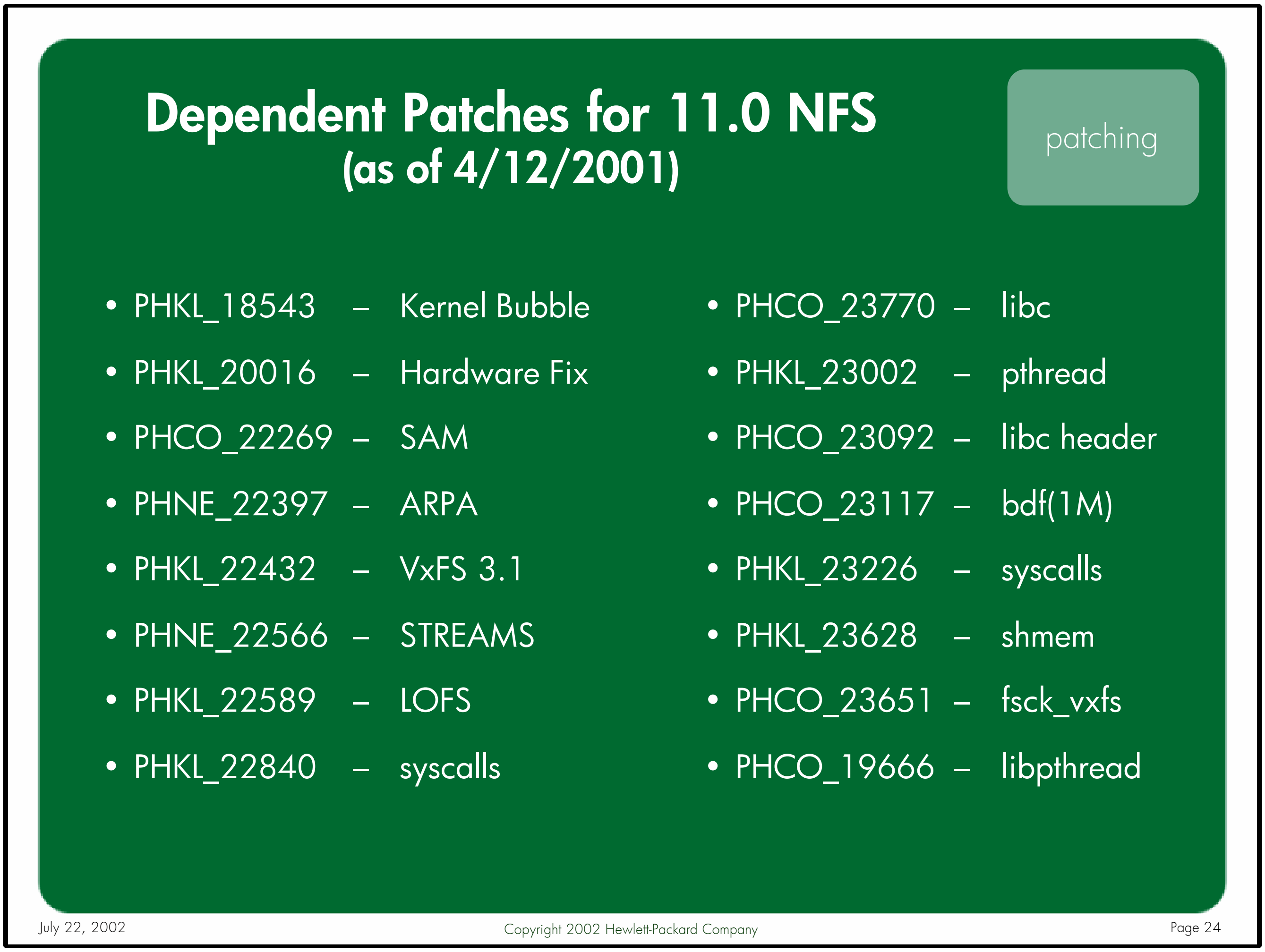
Performance Enhancing Functionality Added in Patches

patching

- Many NFS components have been added to the HP-UX 11.0 release after the original release, including:
  - **AutoFS**
  - **NFS over TCP/IP**
  - NFS PV3 read and write buffer sizes increased from a maximum of 8KB to **32KB**

Notes: In addition to defect fixes, on occasion the NFS lab includes new functionality in their mega-patches. Many of these enhancements directly impact the performance of our NFS implementation.

In March 2000, the NFS lab shipped patches that allowed NFS to run over the TCP/IP transport protocol. This allows NFS to behave more reliably and perform better over wide area networks. At the same time, the lab increased the read and write buffer sizes of the PV3 mount points from a maximum size of 8KB to the industry standard 32KB. This increased buffer size helps HP's implementation achieve much higher read and write throughput numbers than with our previous 8KB limitation.



Dependent Patches for 11.0 NFS (as of 4/12/2001)

patching

PHKL_18543	–	Kernel Bubble	PHCO_23770	–	libc
PHKL_20016	–	Hardware Fix	PHKL_23002	–	pthread
PHCO_22269	–	SAM	PHCO_23092	–	libc header
PHNE_22397	–	ARPA	PHCO_23117	–	bdf(1M)
PHKL_22432	–	VxFS 3.1	PHKL_23226	–	syscalls
PHNE_22566	–	STREAMS	PHKL_23628	–	shmem
PHKL_22589	–	LOFS	PHCO_23651	–	fsck_vxfs
PHKL_22840	–	syscalls	PHCO_19666	–	libpthread

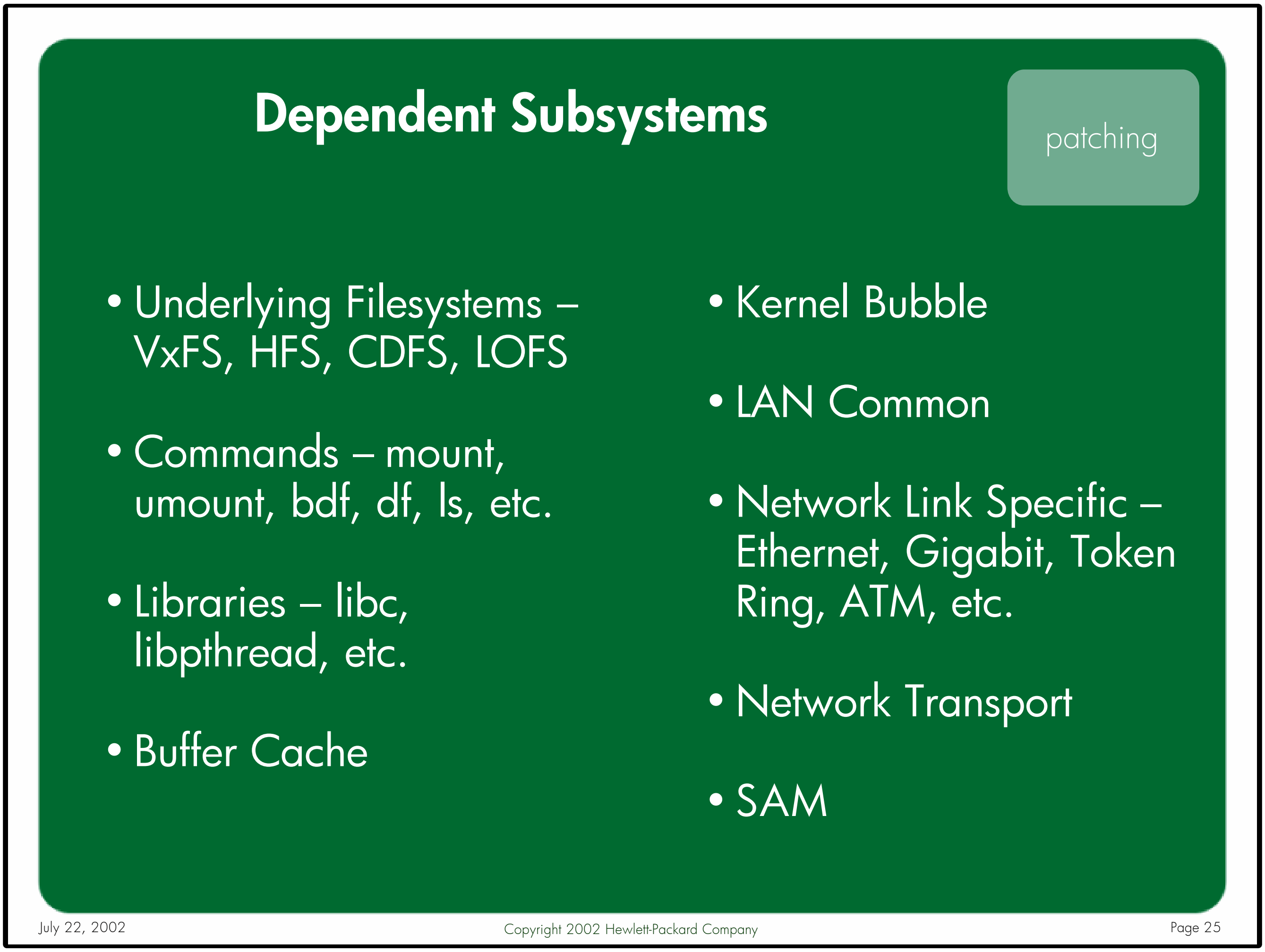
July 22, 2002

Page 24

Notes: Since NFS has so many dependencies with other components of the system, it should comes as no surprise that the list of dependent patches is lengthy.

Patches are superceded frequently so the above list of patches may be out of date by the time you view this presentation. It is merely provided as an example to show how many different subsystems NFS depends upon for correct functionality and performance.

Be sure to contact HP support to obtain a current set of NFS patches and their dependencies for your specific operating system.



Dependent Subsystems

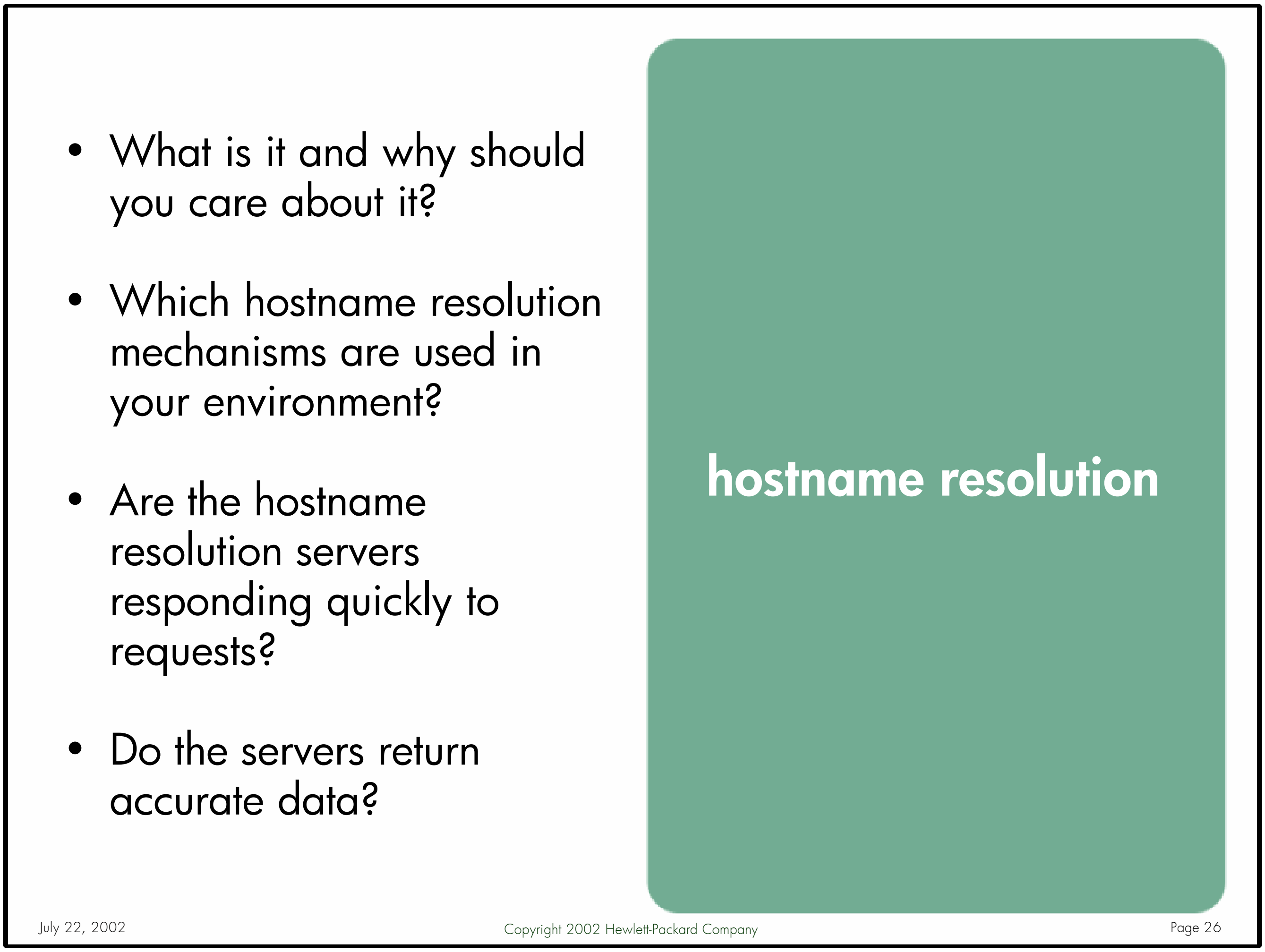
patching

- Underlying Filesystems – VxFS, HFS, CDFS, LOFS
- Commands – mount, umount, bdf, df, ls, etc.
- Libraries – libc, libpthread, etc.
- Buffer Cache
- Kernel Bubble
- LAN Common
- Network Link Specific – Ethernet, Gigabit, Token Ring, ATM, etc.
- Network Transport
- SAM

July 22, 2002

Page 25

Notes: It is very important to keep the patch levels of all the underlying subsystems (i.e. network, buffer cache, transport, etc.) current since a defect in those subsystems can negatively impact NFS behavior and performance.



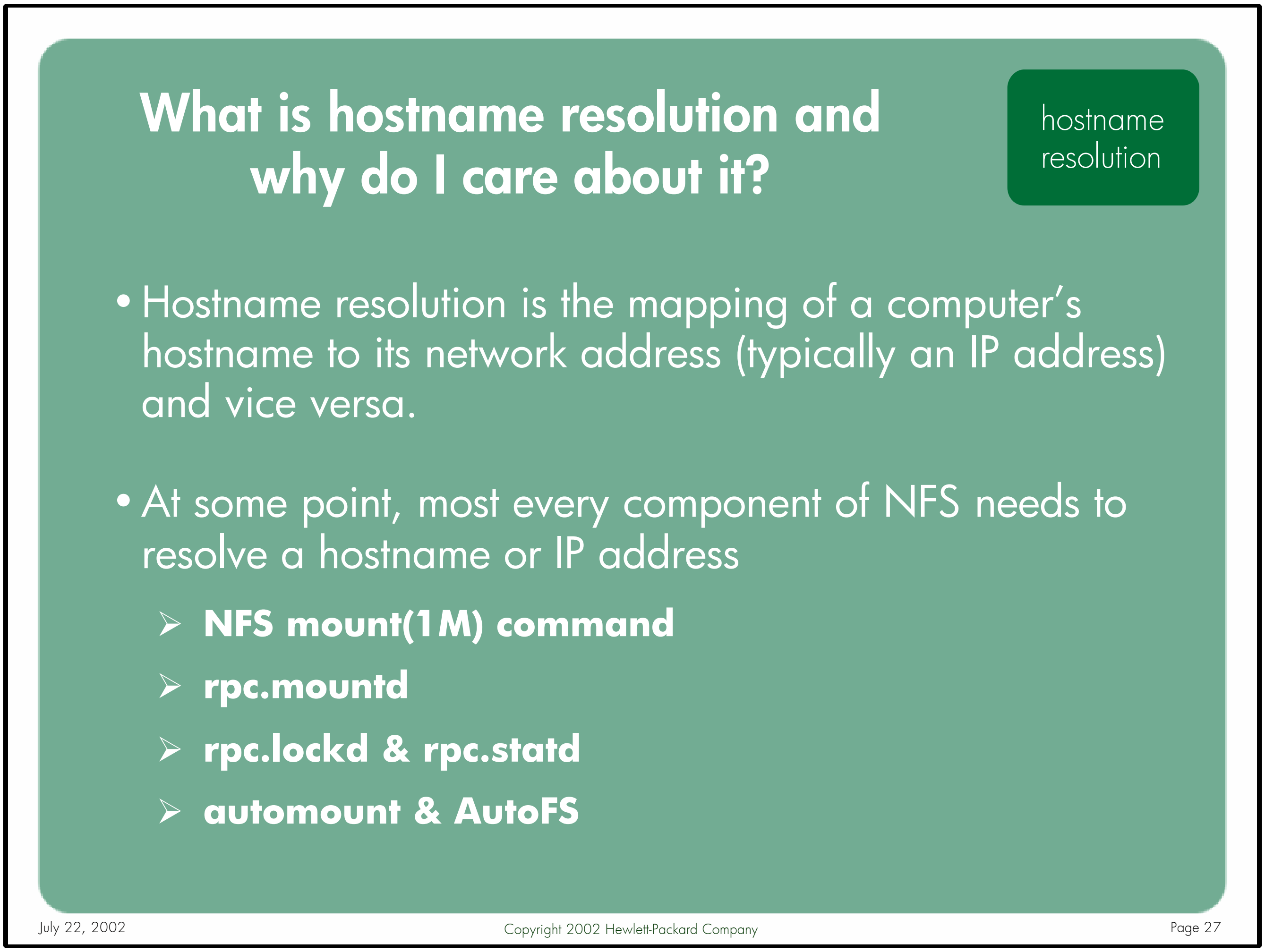
hostname resolution

- What is it and why should you care about it?
- Which hostname resolution mechanisms are used in your environment?
- Are the hostname resolution servers responding quickly to requests?
- Do the servers return accurate data?

July 22, 2002

Page 26

Notes: At some point, nearly every component of NFS needs to resolve an IP address to a hostname, or vice versa. It is therefore critical to familiarize yourself with the hostname resolution mechanisms used in your NFS environment. Any servers providing hostname resolution services should respond quickly to requests and return accurate information.



What is hostname resolution and why do I care about it?

hostname resolution

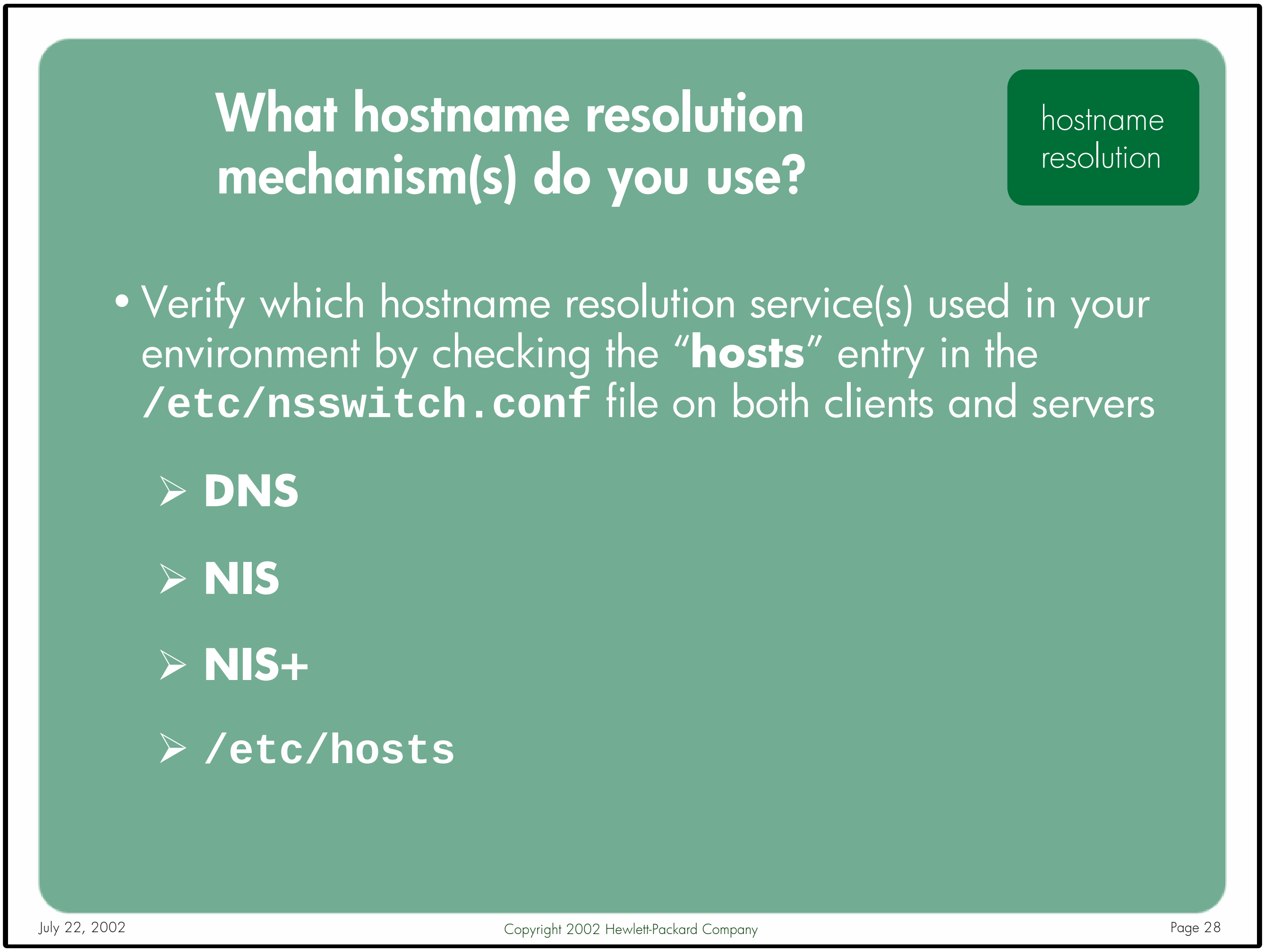
- Hostname resolution is the mapping of a computer’s hostname to its network address (typically an IP address) and vice versa.
- At some point, most every component of NFS needs to resolve a hostname or IP address
  - ➢ **NFS mount(1M) command**
  - ➢ **rpc.mountd**
  - ➢ **rpc.lockd & rpc.statd**
  - ➢ **automount & AutoFS**

July 22, 2002

Page 27

Notes: Most network-based applications, particularly client/server based applications, rely on hostname resolution services to function correctly. NFS is no exception.

Whether it be an NFS client system attempting to mount a remote filesystem (manually or via an automounter), an NFS server system processing the mount request and determining whether the client is allowed to access the requested filesystem, or a client application attempting to lock an NFS-mounted file, hostname resolution is involved.



What hostname resolution mechanism(s) do you use?

hostname resolution

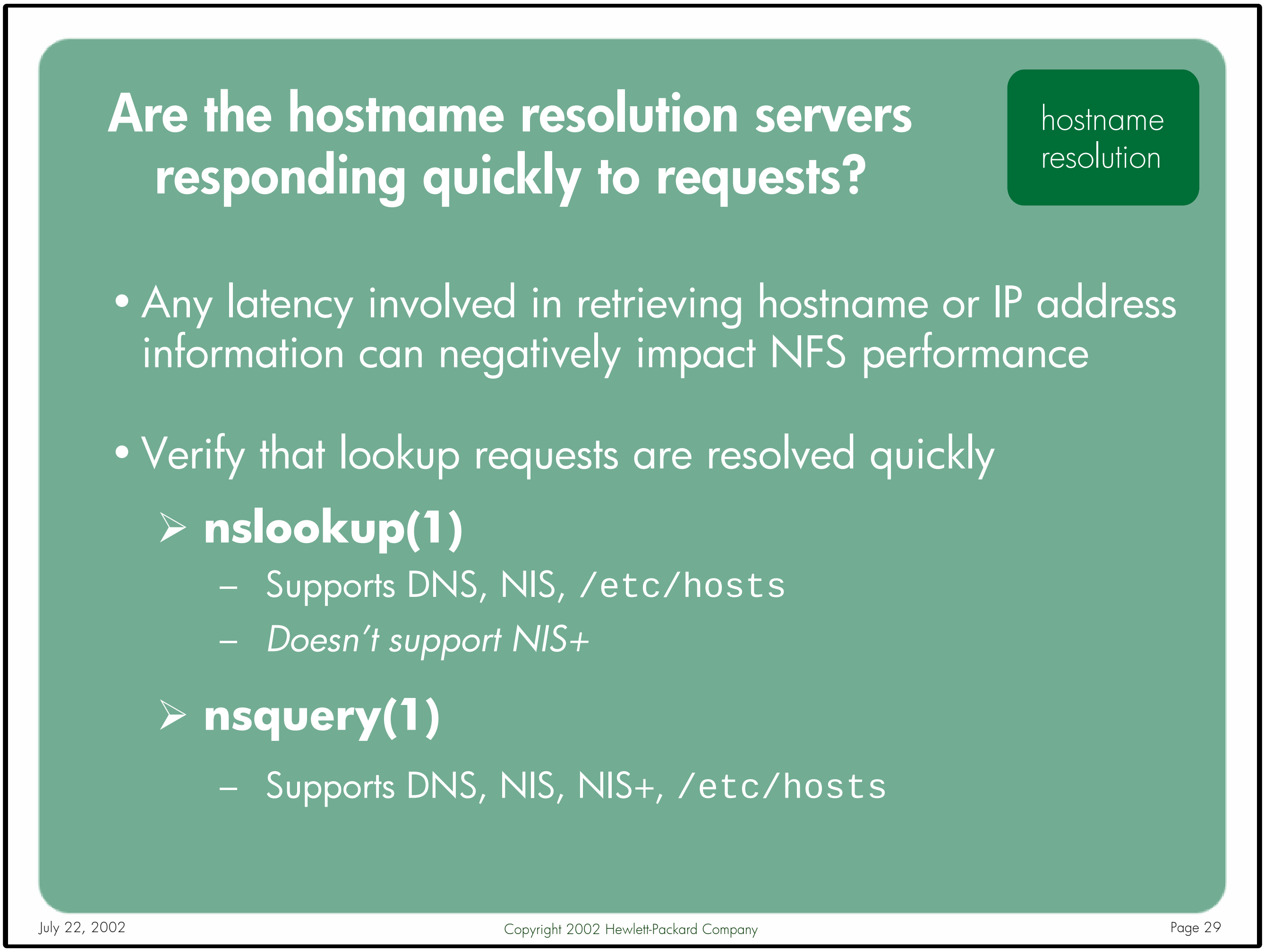
- Verify which hostname resolution service(s) used in your environment by checking the "**hosts**" entry in the **`/etc/nsswitch.conf`** file on both clients and servers
  - ➢ **DNS**
  - ➢ **NIS**
  - ➢ **NIS+**
  - ➢ `/etc/hosts`

Notes: There are several hostname resolution mechanisms available on HP-UX:

- DNS
- NIS
- NIS+
- `/etc/hosts`

Each mechanism has its own specific configuration requirements and sizing issues.

It is important to familiarize yourself with the hostname resolution method (or methods) used in your environment. This allows you to make an informed decision about where to begin troubleshooting when a hostname resolution issue arises.



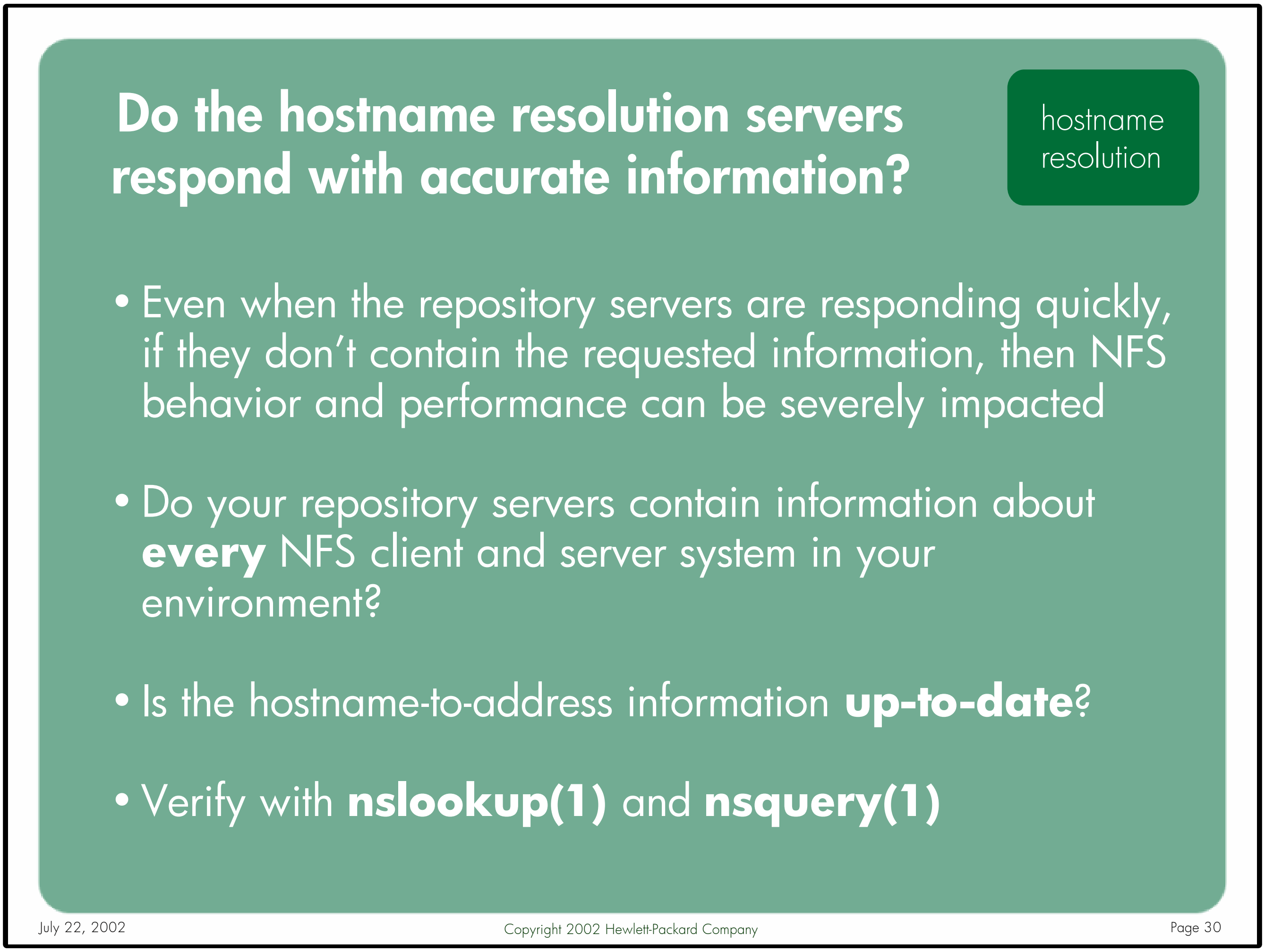
Are the hostname resolution servers responding quickly to requests?

hostname resolution

- Any latency involved in retrieving hostname or IP address information can negatively impact NFS performance
- Verify that lookup requests are resolved quickly
  - ➢ **nslookup(1)**
    - – Supports DNS, NIS, `/etc/hosts`
    - – *Doesn't support NIS+*
  - ➢ **nsquery(1)**
    - – Supports DNS, NIS, NIS+, `/etc/hosts`

Notes: HP-UX supports many different directory service back-ends for hostname resolution data – DNS, NIS, NIS+, and `/etc/hosts`. Each of these repositories has different amounts of overhead associated with retrieving hostname data from them. Any latency involved in retrieving this information can negatively impact NFS performance.

The nslookup(1) and nsquery(1) commands can be used to test the performance of your hostname resolution servers.



Do the hostname resolution servers respond with accurate information?

hostname resolution

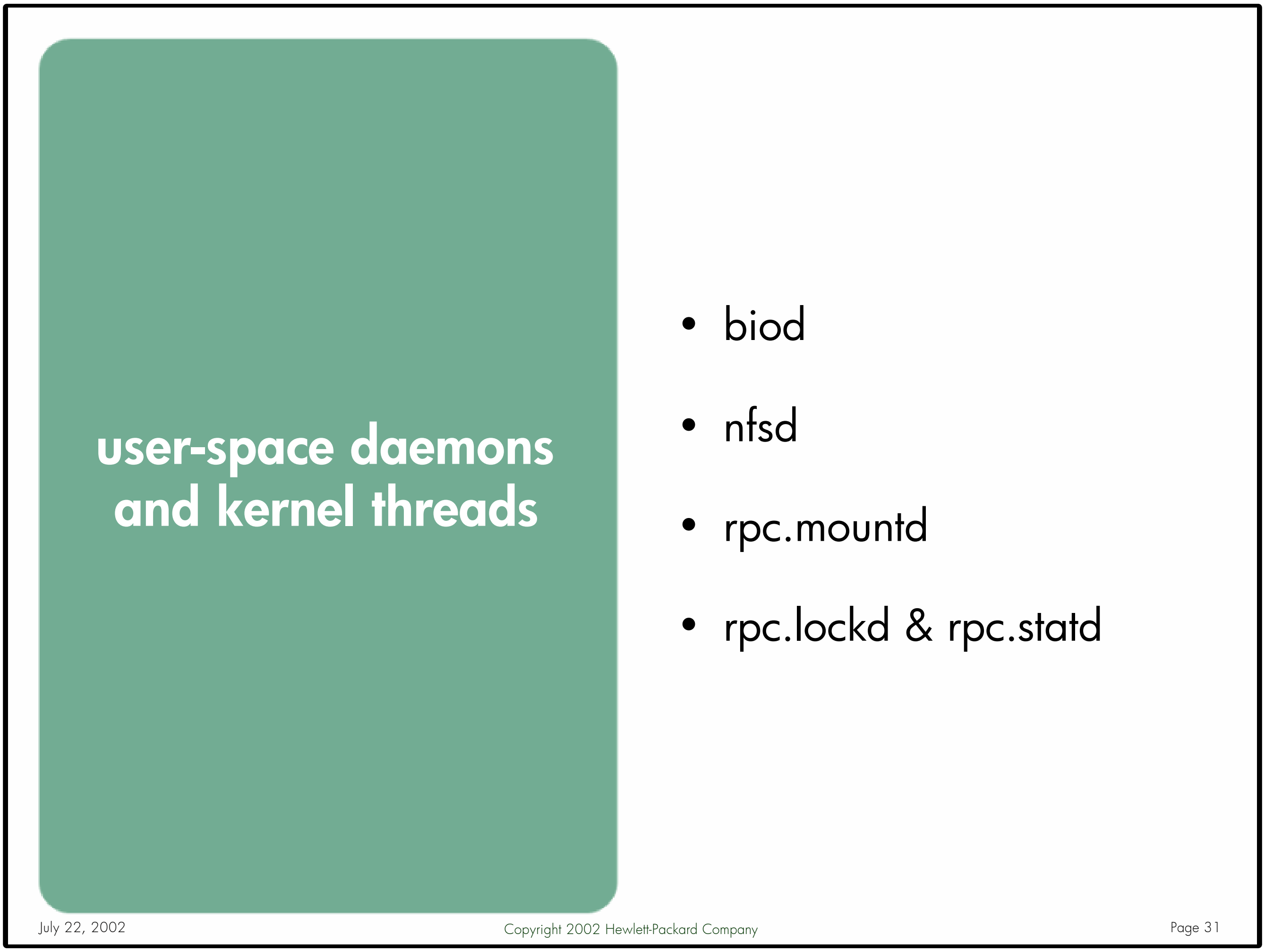
- Even when the repository servers are responding quickly, if they don’t contain the requested information, then NFS behavior and performance can be severely impacted
- Do your repository servers contain information about **every** NFS client and server system in your environment?
- Is the hostname-to-address information **up-to-date**?
- Verify with **nslookup(1)** and **nsquery(1)**

July 22, 2002

Page 30

Notes: Even when the hostname repositories are responding very quickly to resolution requests, if they don’t have the requested data or if they return inaccurate data, the results can be devastating to NFS behavior and performance. Verify that your repository servers contain up-to-date information for all the NFS client and server systems in your environment.

The nslookup(1) and nsquery(1) commands can be used to query the hostname resolution servers, allowing you to verify the accuracy of the information returned.



user-space daemons and kernel threads

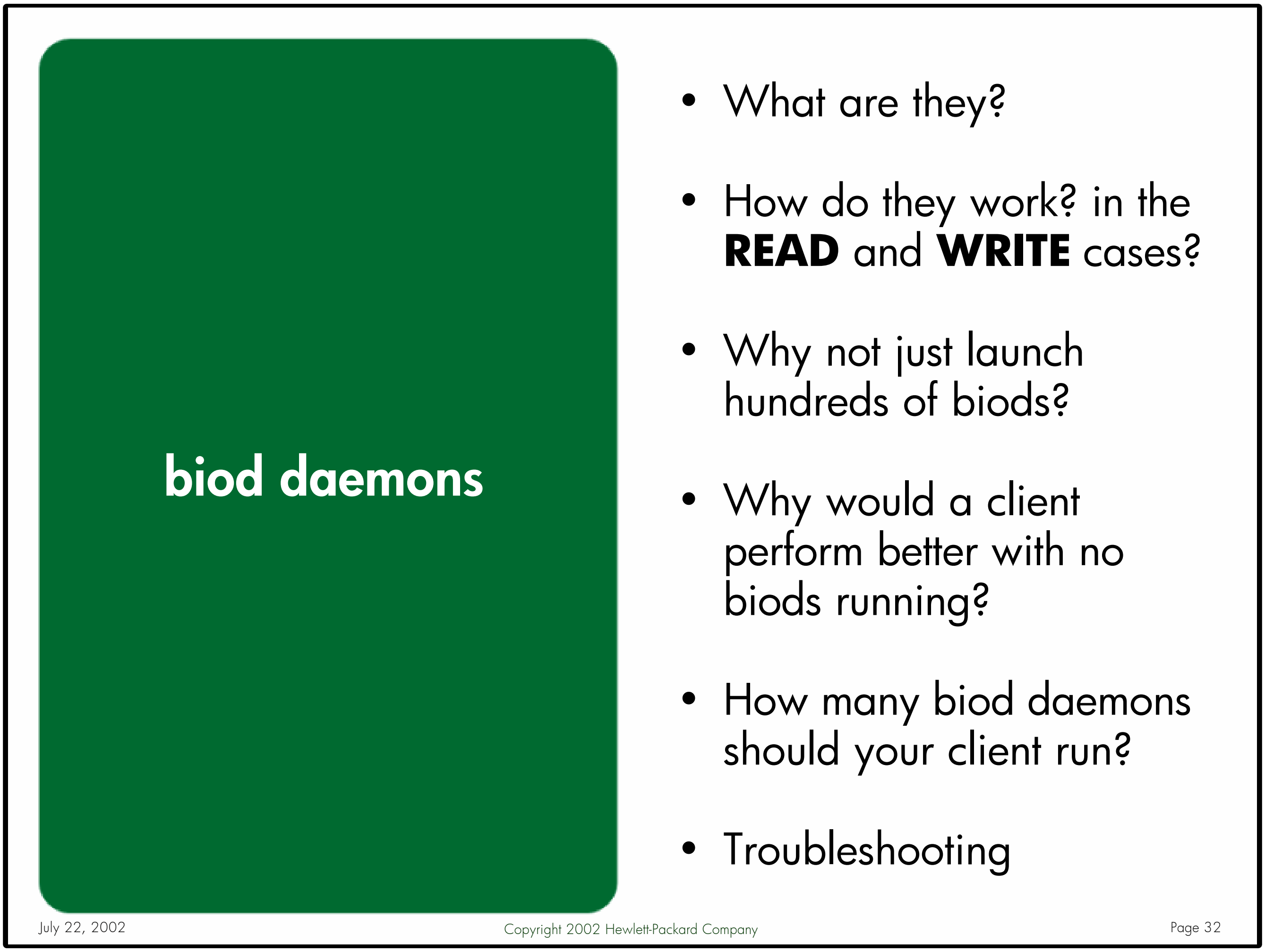
- biod
- nfsd
- rpc.mountd
- rpc.lockd & rpc.statd

July 22, 2002

Page 31

Notes: NFS is a true “client-server” application, yet most of the tuning and troubleshooting information currently available (i.e. white papers, web pages, etc.) pertains only to NFS servers. There are many components of NFS that are specific to the client, and some administrators tend to forget that without a well-tuned client it doesn’t really matter how fast your server is – you’ll still end up with poor NFS performance.

In order to obtain optimal NFS performance, both the client and the server should be sized and tuned appropriately for an application’s needs. This involves making sure that both biods (client) and nfsds (server) are considered. The rpc.mountd daemon, which runs on the server only, has specific environmental considerations that should be verified. The rpc.lockd and rpc.statd daemons run on both clients and servers, so both systems should be configured properly to run these daemons effectively.



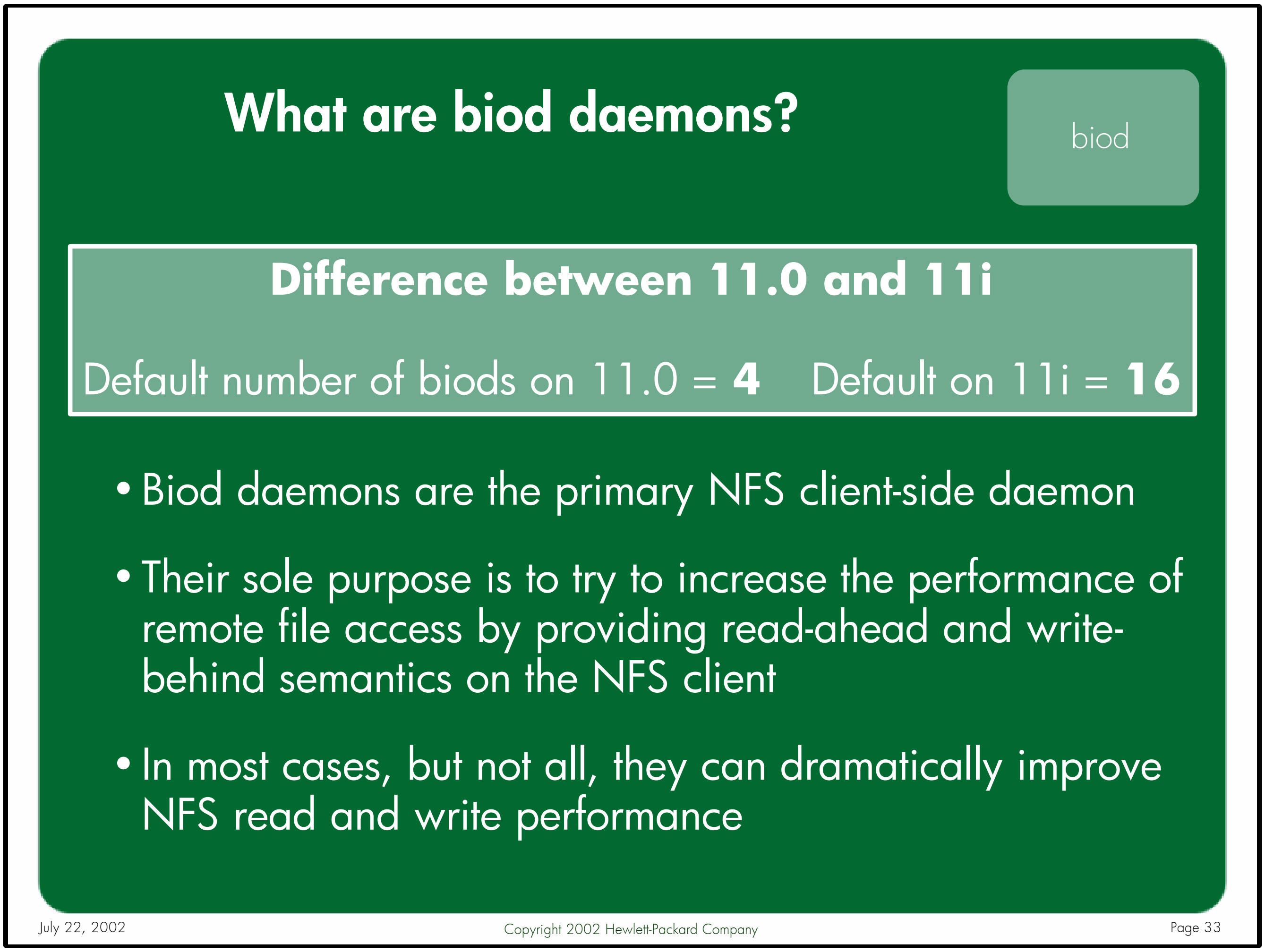
biod daemons

- What are they?
- How do they work? in the **READ** and **WRITE** cases?
- Why not just launch hundreds of biods?
- Why would a client perform better with no biods running?
- How many biod daemons should your client run?
- Troubleshooting

July 22, 2002

Page 32

Notes: This section describes how the biod daemons work in both the read and write scenarios. It explains why running a large number of biods does not necessarily result in better performance. It also covers the various factors involved in selecting the appropriate number of biod daemons to run on any given client. Finally, it describes the tools and methods available for troubleshooting biod-related problems.



What are biod daemons?

biod

Difference between 11.0 and 11i

Default number of biods on 11.0 = **4**   Default on 11i = **16**

- Biod daemons are the primary NFS client-side daemon
- Their sole purpose is to try to increase the performance of remote file access by providing read-ahead and write-behind semantics on the NFS client
- In most cases, but not all, they can dramatically improve NFS read and write performance

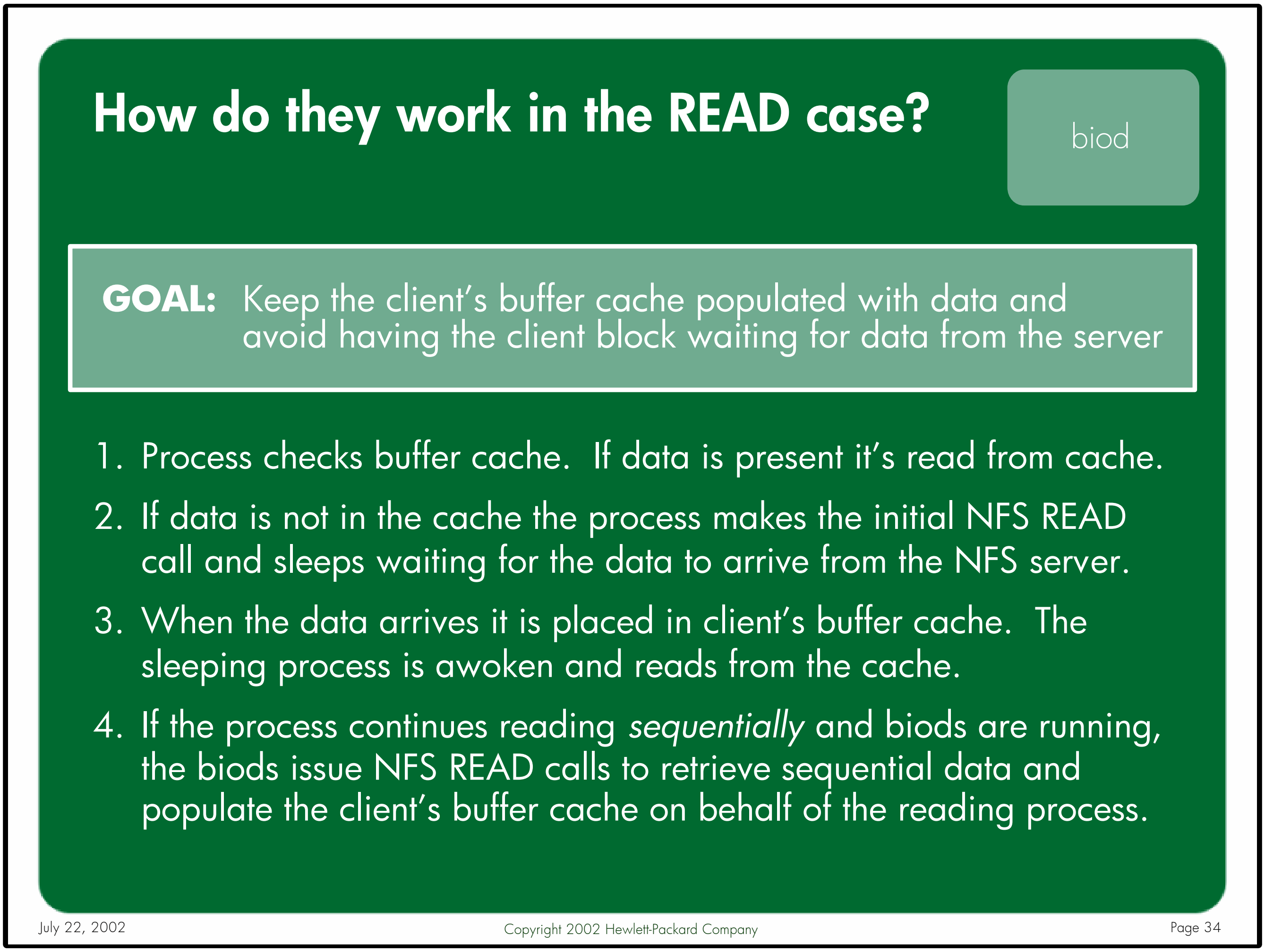
July 22, 2002

Page 33

Notes: Biods are the primary NFS client-side daemon. Although they are implemented as user-space processes, they spend the majority of their time running in the kernel. Their sole purpose is to try to increase the performance of remote file access by providing read-ahead and write-behind semantics on the NFS client. In most cases, but not all, they can dramatically improve NFS read and write performance.

Determining the appropriate number of biod processes to run on a specific NFS client, like most every other performance issue, is not easily defined and usually falls into the category of ‘It depends’. Some clients will perform best with 4 biods while others work better with 16. Still other clients perform best when no biod daemons are running.

The default number of biods launched at boot time has changed at 11i. 11.0 systems launch 4 biods by default, while 11i NFS clients launch 16.



How do they work in the READ case?

biod

GOAL: Keep the client's buffer cache populated with data and avoid having the client block waiting for data from the server

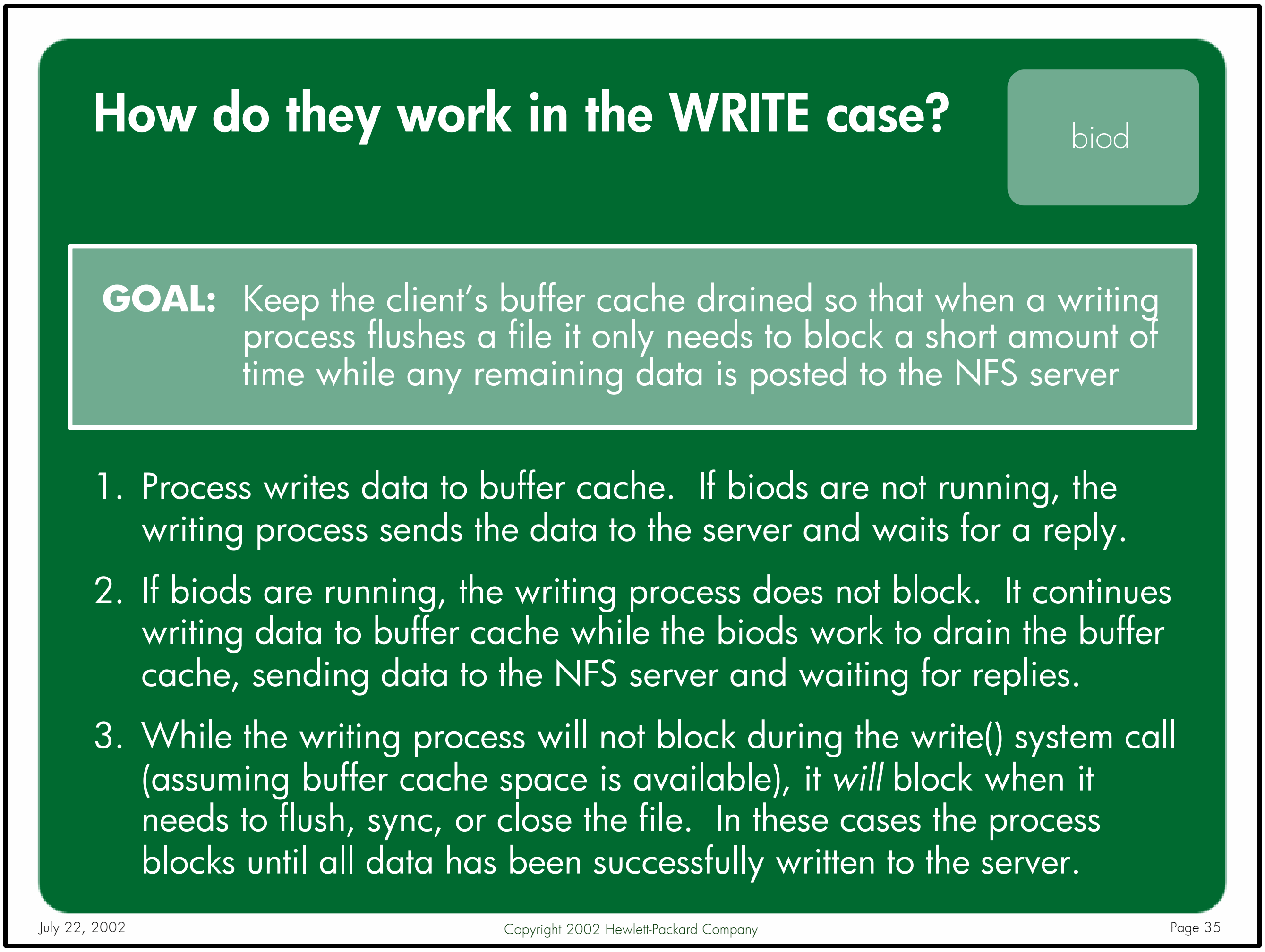
1. Process checks buffer cache. If data is present it's read from cache.
2. If data is not in the cache the process makes the initial NFS READ call and sleeps waiting for the data to arrive from the NFS server.
3. When the data arrives it is placed in client's buffer cache. The sleeping process is awoken and reads from the cache.
4. If the process continues reading *sequentially* and biods are running, the biods issue NFS READ calls to retrieve sequential data and populate the client's buffer cache on behalf of the reading process.

July 22, 2002

Page 34

Notes: When the client needs to read data from an NFS mounted file, the client will first check its local buffer cache to see if the block of data is present in the cache. If it is then the read is satisfied without generating an NFS request. If the data is not present then an NFS READ call will be made to retrieve this data. If no biods are running then the process requesting the data will generate the NFS read call in its own process context.

If biod daemons are present then the client process will send the initial read request in its own context and then block to allow future read requests to be handled by the biods. Once the biods retrieve the data from the server, the data is staged in the client's buffer cache and the blocking process is notified that the data is available. In addition, several more sequential read requests will be made on behalf of the process by the biods in the hopes of populating the client's buffer cache with the data that the reading process will be wanting next. Therefore, the goal of the biods in the read case is to keep the client's buffer cache populated with the data that the reading processes need and avoid having those processes block waiting for the data to be retrieved from the NFS server.



How do they work in the WRITE case?

biod

GOAL: Keep the client's buffer cache drained so that when a writing process flushes a file it only needs to block a short amount of time while any remaining data is posted to the NFS server

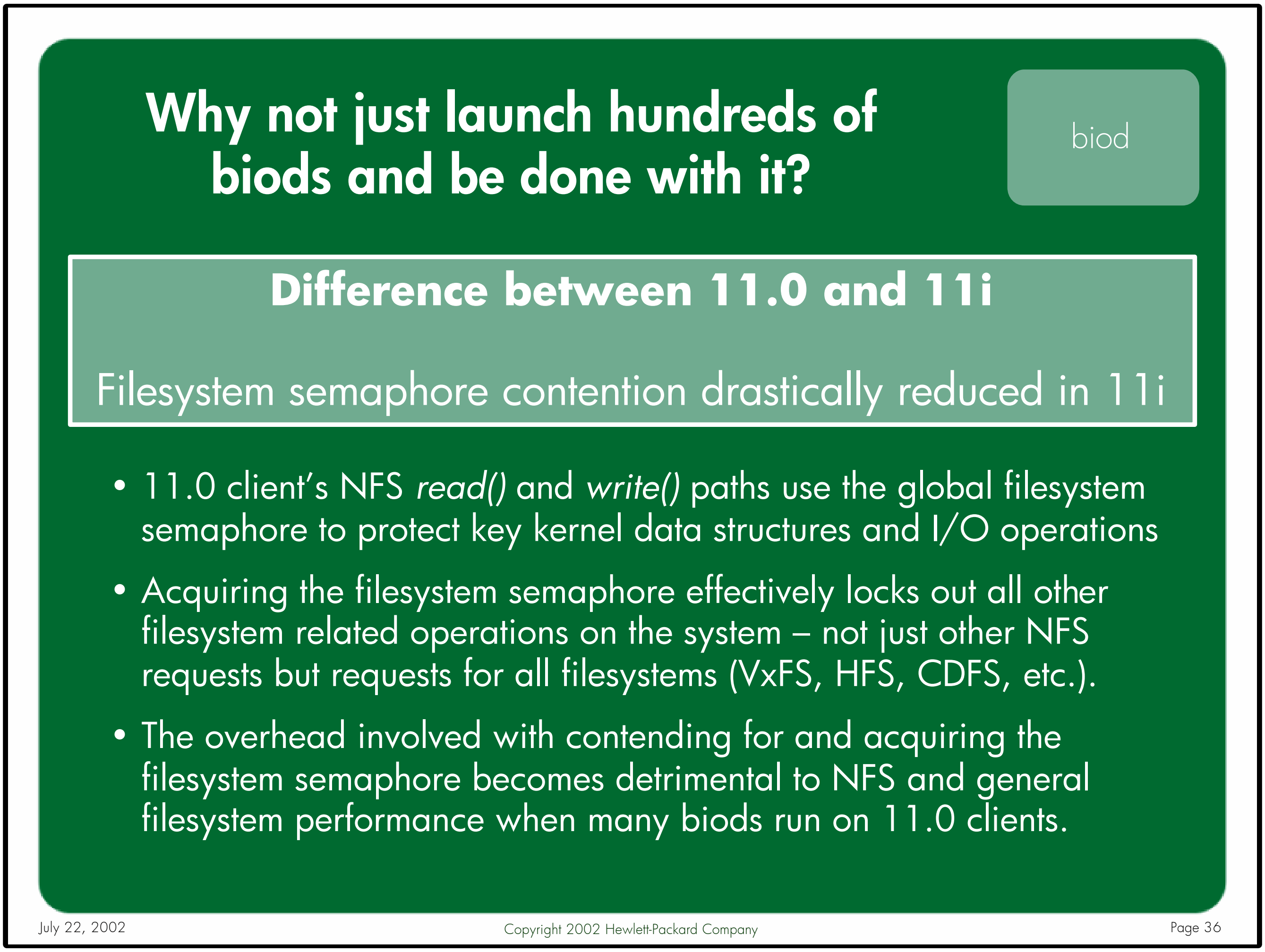
1. Process writes data to buffer cache. If biods are not running, the writing process sends the data to the server and waits for a reply.
2. If biods are running, the writing process does not block. It continues writing data to buffer cache while the biods work to drain the buffer cache, sending data to the NFS server and waiting for replies.
3. While the writing process will not block during the write() system call (assuming buffer cache space is available), it *will* block when it needs to flush, sync, or close the file. In these cases the process blocks until all data has been successfully written to the server.

July 22, 2002

Page 35

Notes: When the client needs to write data to an NFS mounted file, assuming the client is writing in asynchronous mode (which is the default behavior), the client will write the data to its local buffer cache. Once the buffer cache data has been written, if no biod daemons are running then the process writing the data will generate an NFS write request in its own process context to push the data to the NFS server.

If biods are present then the writing process will post its data to the local buffer cache and continue writing while the data is sent to the NFS server by a biod. Several more biods continue to pull data from the client's buffer cache and send it to the server, allowing the writing process to continue writing data to buffer cache as quickly as it can. While the writing process itself will not block during the actual write() system call, it will block when it needs to flush, sync, or close the file. In these cases the process sleeps until all data has been written to the server. Therefore, the goal of the biods in the write case is to drain the client's buffer cache so that when the writing process is finished writing and needs to flush, sync, or close the file, it only needs to block for a short amount of time while any remaining data is sent to the server.



Why not just launch hundreds of biods and be done with it?

biod

Difference between 11.0 and 11i

Filesystem semaphore contention drastically reduced in 11i

- 11.0 client's NFS *read()* and *write()* paths use the global filesystem semaphore to protect key kernel data structures and I/O operations
- Acquiring the filesystem semaphore effectively locks out all other filesystem related operations on the system – not just other NFS requests but requests for all filesystems (VxFS, HFS, CDFS, etc.).
- The overhead involved with contending for and acquiring the filesystem semaphore becomes detrimental to NFS and general filesystem performance when many biods run on 11.0 clients.

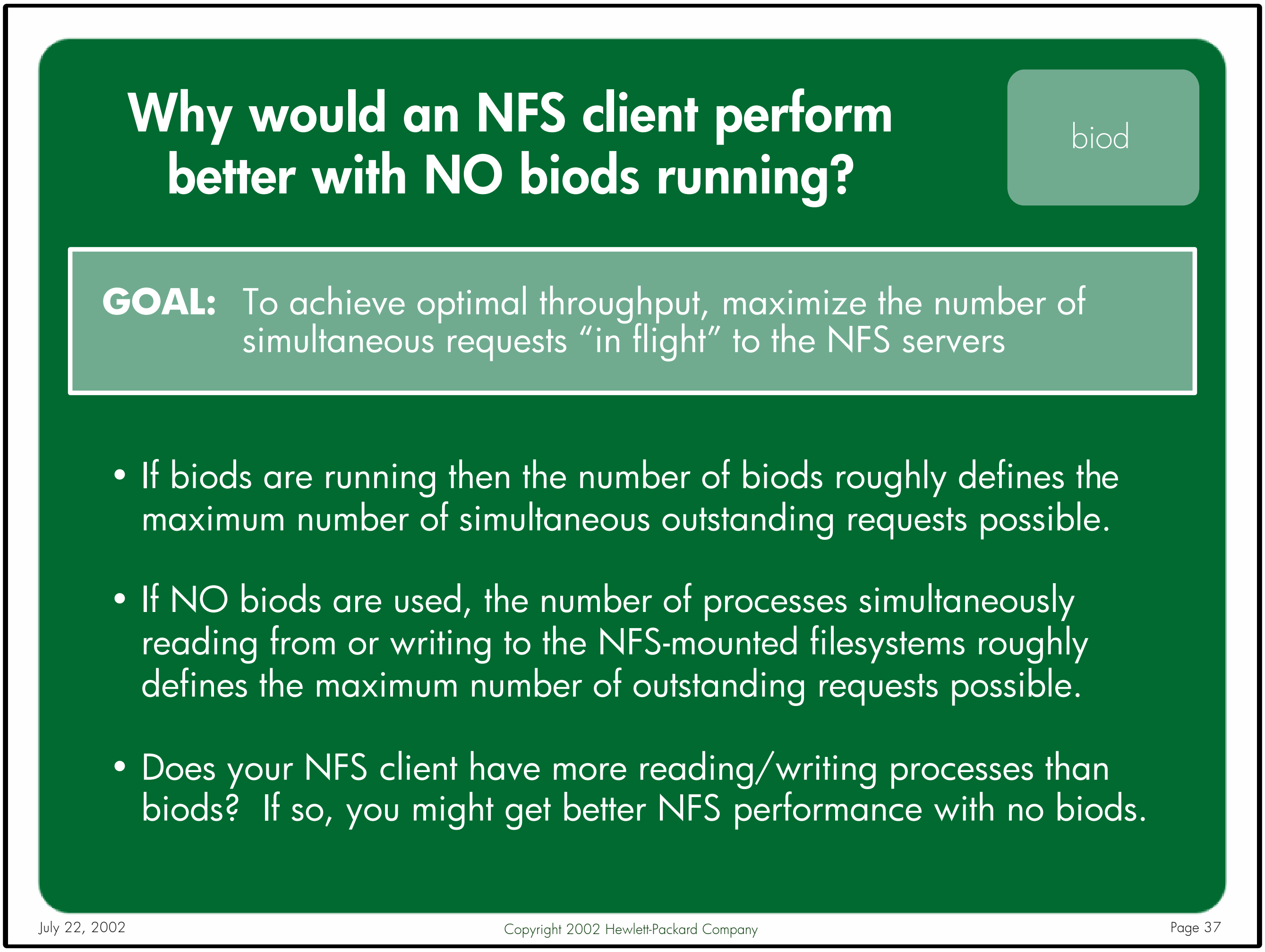
July 22, 2002

Page 36

Notes: Based upon the earlier description of what the biods do, one might conclude that the more biods running the better your NFS client will perform – so why not just launch 200 biods and be done with it?

On HP-UX 11.0 NFS clients this is not a good idea because the client's read() and write() paths use the global filesystem semaphore to protect many kernel data structures and NFS I/O operations. What this means is that whenever a biod processes a read or write request it must acquire the filesystem semaphore, which effectively locks out all other filesystem related operations on the system – not just other NFS requests but requests for all filesystems (i.e. VxFS, HFS, NFS, CDFS, etc.). Therefore, if an 11.0 NFS client runs a large number of biod daemons, the overhead involved with contending for and acquiring the filesystem semaphore becomes detrimental to NFS and general filesystem performance.

This filesystem semaphore contention issue is drastically reduced in 11i, which means that an 11i client could potentially benefit from running more biods than an 11.0 client.



Why would an NFS client perform better with NO biods running?

biod

GOAL: To achieve optimal throughput, maximize the number of simultaneous requests “in flight” to the NFS servers

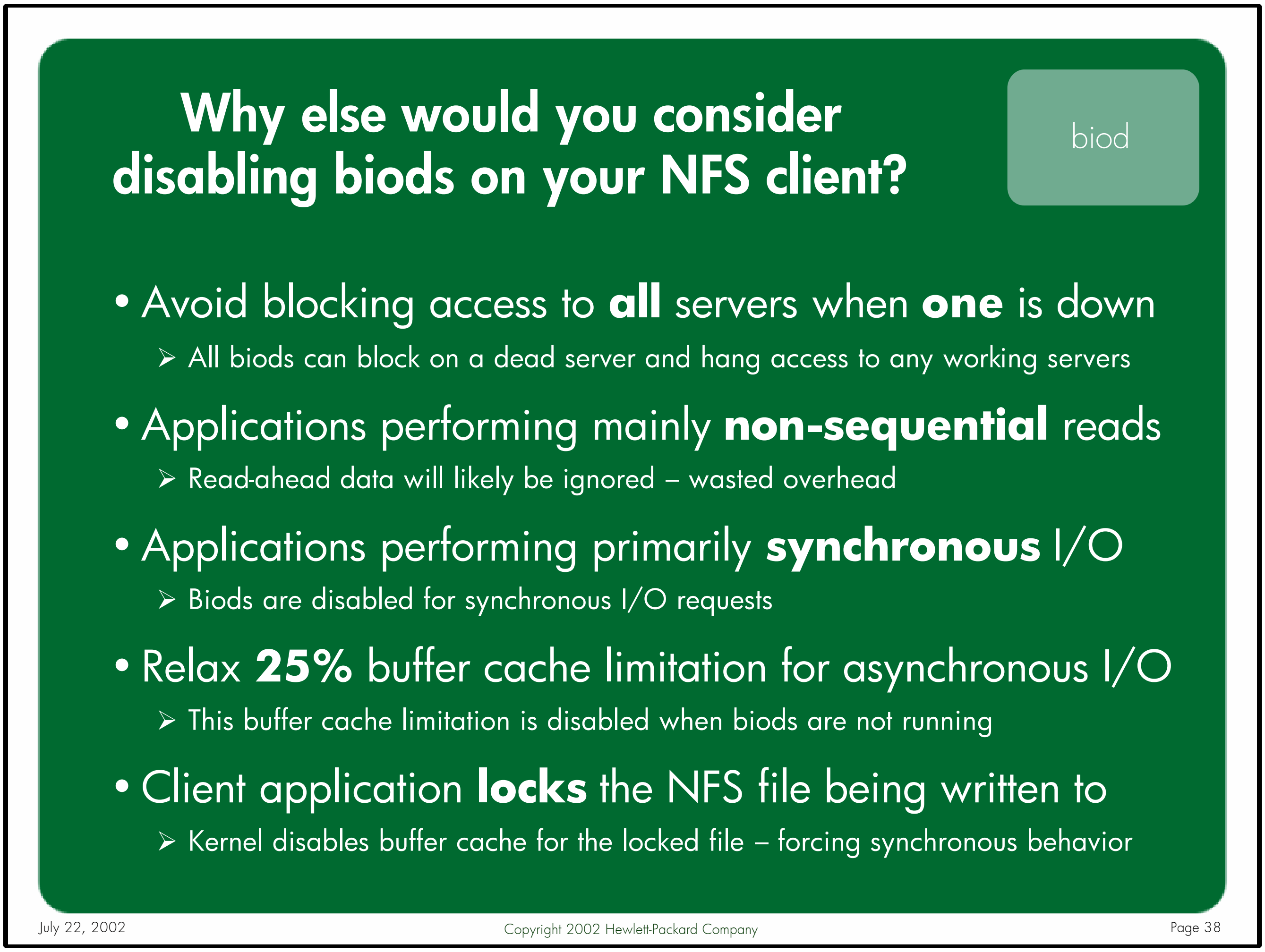
- If biods are running then the number of biods roughly defines the maximum number of simultaneous outstanding requests possible.
- If NO biods are used, the number of processes simultaneously reading from or writing to the NFS-mounted filesystems roughly defines the maximum number of outstanding requests possible.
- Does your NFS client have more reading/writing processes than biods? If so, you might get better NFS performance with no biods.

July 22, 2002

Page 37

Notes: Since any process on the NFS client that tries to read from or write to an NFS mount point will block if biod processes are running (to allow the biod processes to perform the I/O on their behalf), it stands to reason that the maximum number of simultaneous I/O operations an NFS client with biods running can perform is equal to or near the number of running biods. In other words, if 4 biods are running on the client then in most cases only 4 NFS read or write calls can occur at any one time.

If an NFS client has a number of processes simultaneously trying to read from and/or write to NFS servers, and this number of processes is greater than the number of running biods, it is possible that the client would experience better NFS performance if no biods were running. For example, if there are 4 biods running on the client but there are 100 processes trying to simultaneously read and write to the NFS servers, these 100 processes will block and wait for the 4 biods to service these requests on their behalf. If, however, there were no biod daemons running, each of the 100 processes would send their NFS requests to the respective servers on their own behalf, potentially resulting in higher throughput.



Why else would you consider disabling biods on your NFS client?

biod

- Avoid blocking access to **all** servers when **one** is down
  - ➢ All biods can block on a dead server and hang access to any working servers
- Applications performing mainly **non-sequential** reads
  - ➢ Read-ahead data will likely be ignored – wasted overhead
- Applications performing primarily **synchronous** I/O
  - ➢ Biods are disabled for synchronous I/O requests
- Relax **25%** buffer cache limitation for asynchronous I/O
  - ➢ This buffer cache limitation is disabled when biods are not running
- Client application **locks** the NFS file being written to
  - ➢ Kernel disables buffer cache for the locked file – forcing synchronous behavior

July 22, 2002

Page 38

Notes: **Avoid blocking access to all servers when one is down**

If one NFS server stops responding while biods are running, all biods can potentially hang waiting for that server to respond, effectively stopping all NFS I/O – even to available servers. If no biods are running then client processes would still be able to access any available servers.

Applications performing mainly non-sequential reads

If the client applications are performing primarily non-sequential reads, any data pre-fetched by the biods will likely be unwanted by the applications, resulting in wasted overhead from unnecessary read-aheads.

Applications performing primarily synchronous I/O

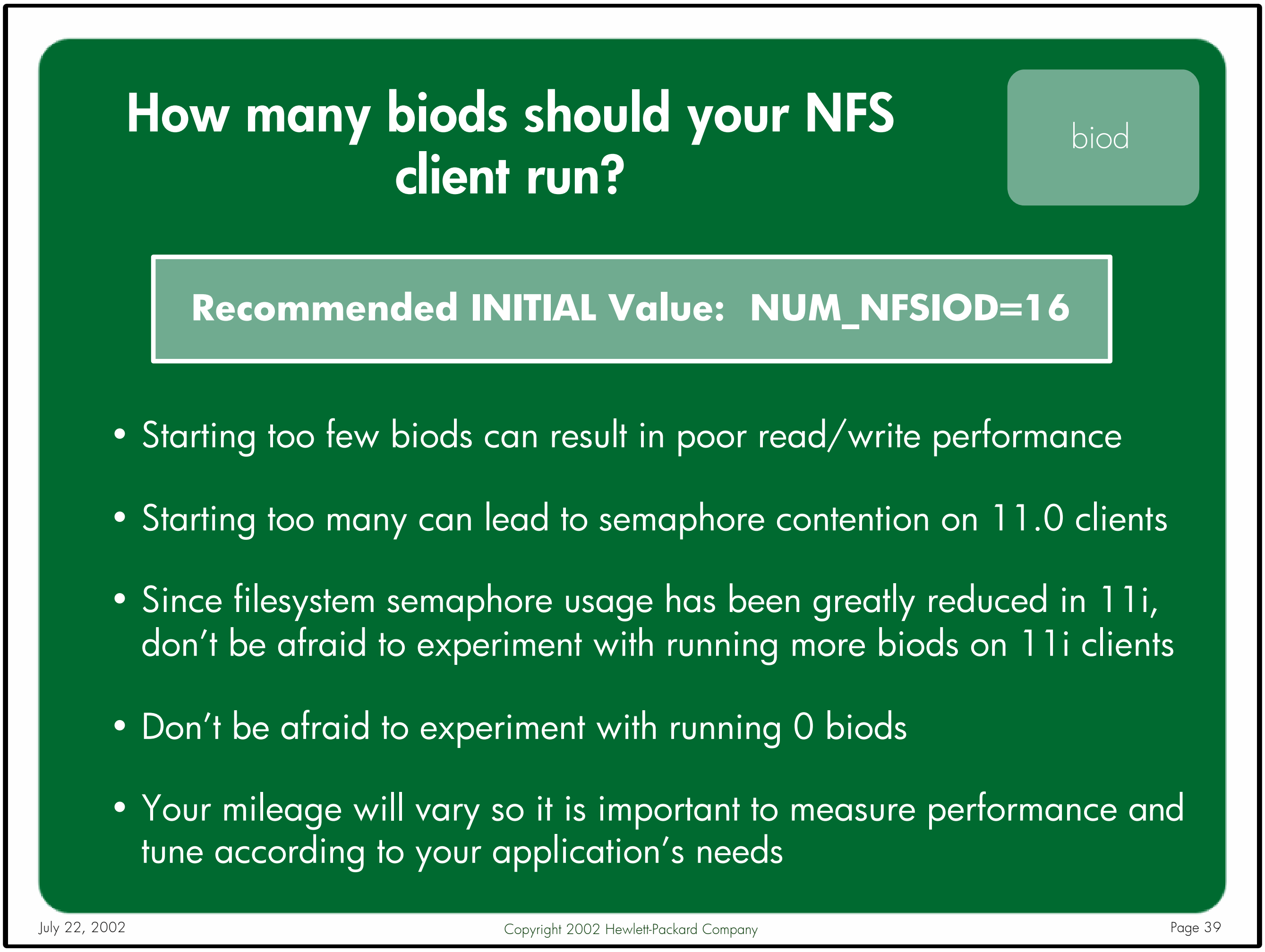
When an application opens a file with the O_SYNC, O_DSYNC, or O_RSYNC flags, the kernel effectively disables the use of biods for any reads and writes to the file.

Relax 25% buffer cache limitation for asynchronous I/O

The NFS client is allowed to use a maximum of 25% of buffer cache memory for staging asynchronous write data. This 25% limit is only enforced when biods are running.

Client application locks the NFS file being written to

When an application locks an NFS-mounted file, any read or write requests made to this file will be performed synchronously. In effect, buffer cache and biods are disabled for that file when a lock is present, so no benefit is gained by running biod daemons in this case.



How many biods should your NFS client run?

biod

Recommended INITIAL Value: NUM_NFSIOD=16

- Starting too few biods can result in poor read/write performance
- Starting too many can lead to semaphore contention on 11.0 clients
- Since filesystem semaphore usage has been greatly reduced in 11i, don't be afraid to experiment with running more biods on 11i clients
- Don't be afraid to experiment with running 0 biods
- Your mileage will vary so it is important to measure performance and tune according to your application's needs

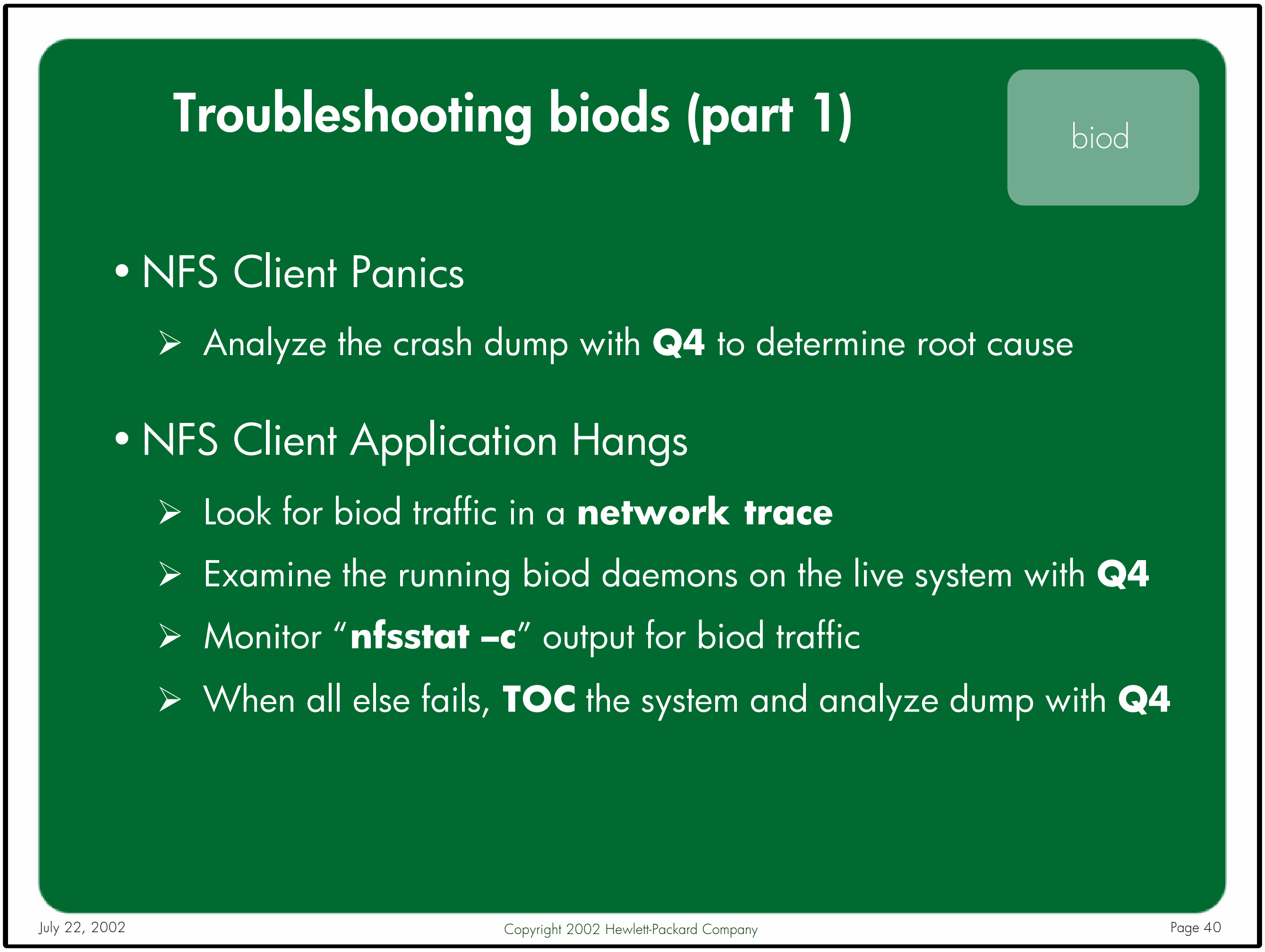
July 22, 2002 Copyright 2002 Hewlett-Packard Company Page 39

Notes: The number of biods is configured via the NUM_NFSIOD variable in the `/etc/rc.config.d/nfsconf` file.

Starting too few biods can result in poor read performance because not enough read-ahead requests are occurring to keep the client's buffer cache populated. It can also result in poor write performance if too many requests are sitting in the client's buffer cache waiting to be sent to the server. Starting too many biods can result in poor performance on 11.0 systems because of filesystem semaphore contention.

The best course of action is to choose a reasonable starting point for the number of biod daemons and then experiment with raising and lowering the number until the optimal value is identified. Don't be afraid to experiment with 0 biods, particularly if your client falls into one of the categories described in the previous two slides.

For most HP-UX 11.0 clients, a good starting point for the number of biods is 16. On 11i clients, higher throughput numbers may be achieved with more biods since it doesn't suffer from the filesystem semaphore contention issue.



Troubleshooting biods (part 1)

biod

- NFS Client Panics
  - ➢ Analyze the crash dump with **Q4** to determine root cause
- NFS Client Application Hangs
  - ➢ Look for biod traffic in a **network trace**
  - ➢ Examine the running biod daemons on the live system with **Q4**
  - ➢ Monitor “**nfsstat –c**” output for biod traffic
  - ➢ When all else fails, **TOC** the system and analyze dump with **Q4**

Notes: **NFS Client Panics**

Analyzing HP-UX system dumps is a complex and involved process, which is typically performed only by experienced HP support personnel or lab engineers. If your NFS client experiences a system panic, your best course of action is to contact HP support and request that your system memory dump be analyzed.

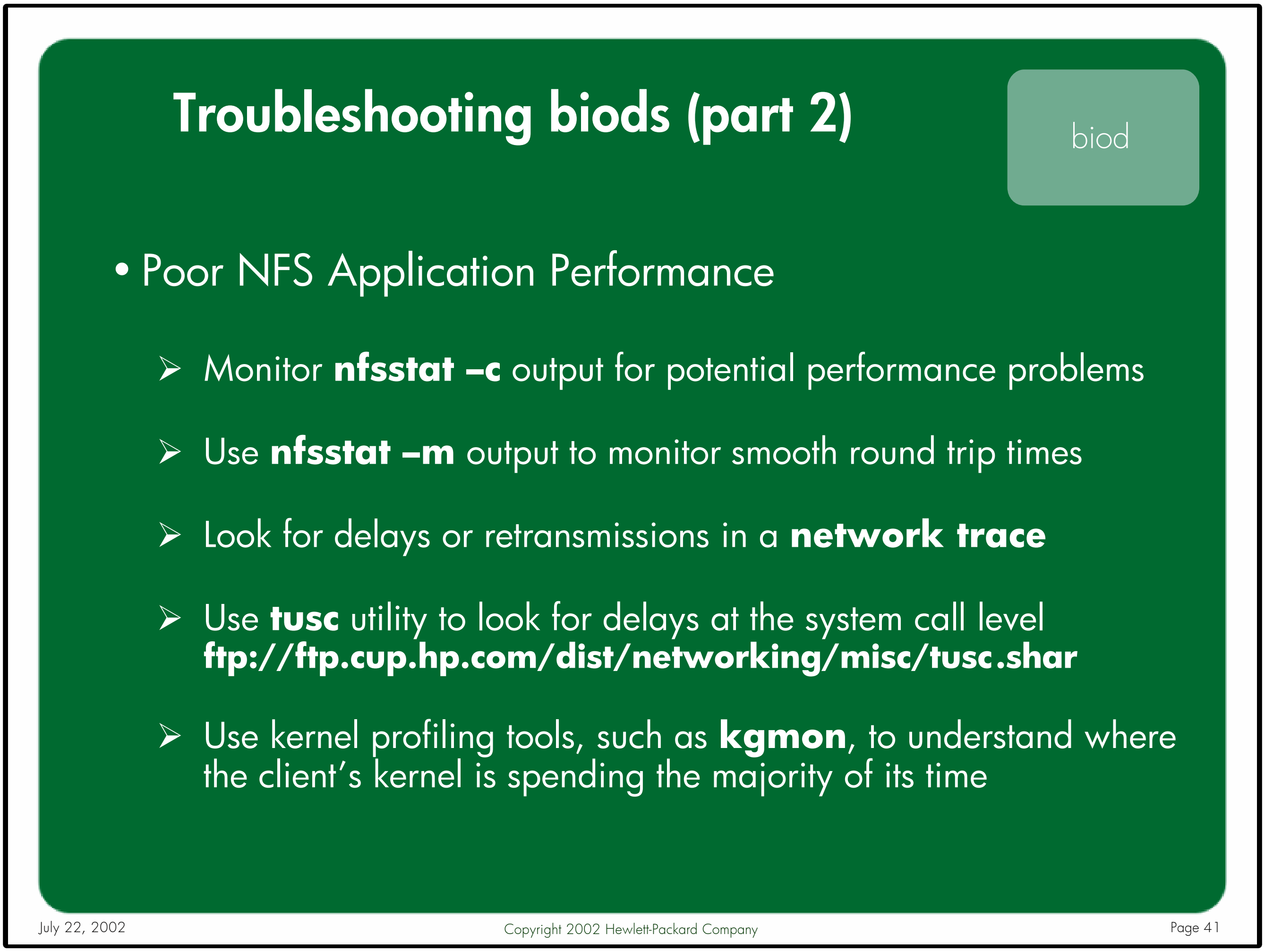
NFS Client Application Hangs

A network trace will show if any biods are sending requests to the server.

Q4 can be used by experienced system administrators or HP support personnel to examine the running biod daemons on the live system.

Successive nfsstat outputs can reveal whether outbound client calls are being sent.

When all else fails, the system may need to be TOC’d, and the memory dump analyzed by HP support personnel.



Troubleshooting biods (part 2)

biod

- Poor NFS Application Performance
  - ➢ Monitor **nfsstat –c** output for potential performance problems
  - ➢ Use **nfsstat –m** output to monitor smooth round trip times
  - ➢ Look for delays or retransmissions in a **network trace**
  - ➢ Use **tusc** utility to look for delays at the system call level **ftp://ftp.cup.hp.com/dist/networking/misc/tusc.shar**
  - ➢ Use kernel profiling tools, such as **kgmon**, to understand where the client's kernel is spending the majority of its time

Notes: **Poor NFS Application Performance**

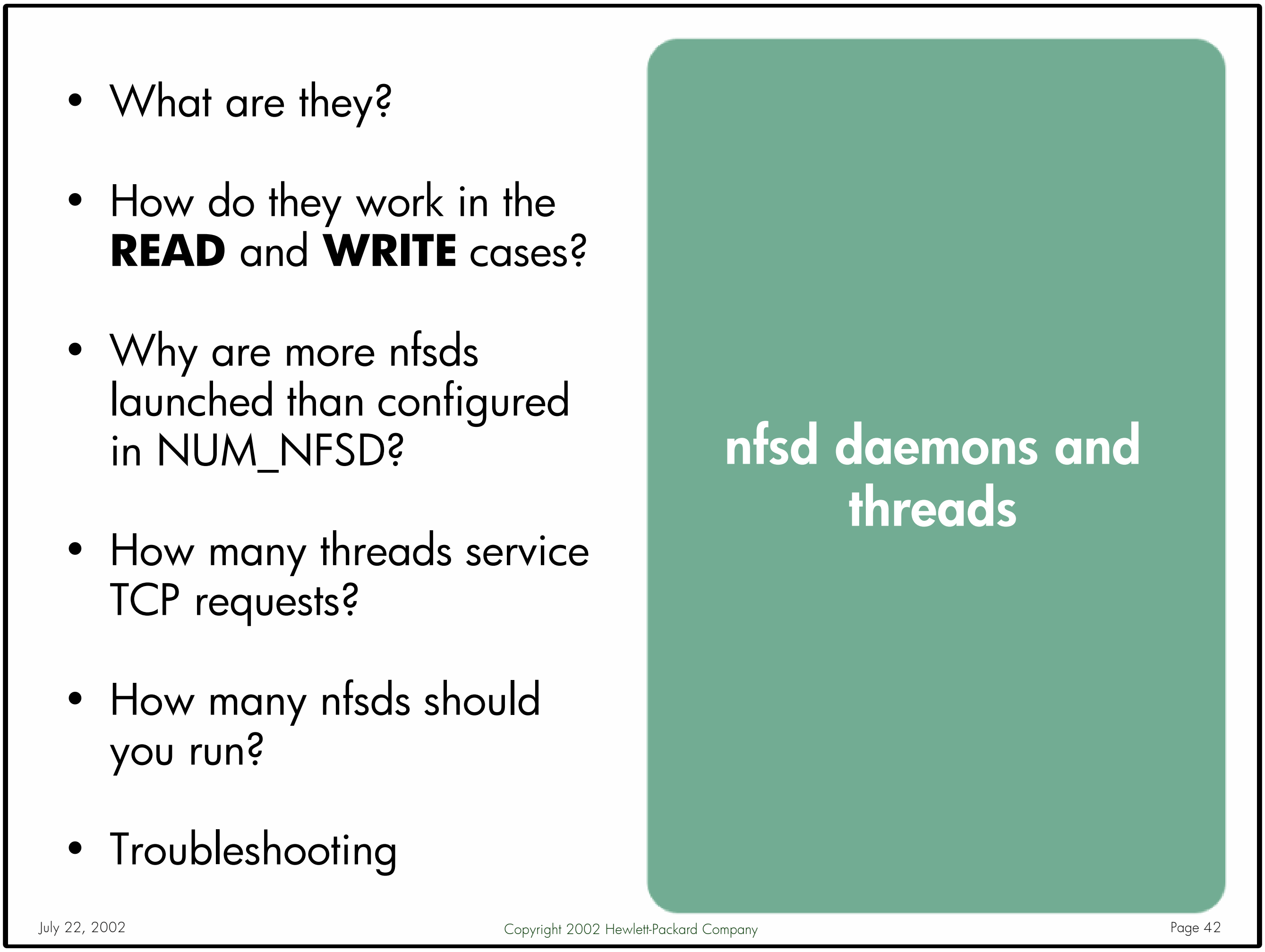
"nfsstat –c" output can reveal if retransmissions, badxids (when a server replies to the same request multiple times), etc. are occurring. These can indicate that NFS requests are being dropped by the client, the server, or the network.

"nfsstat –m" output includes the smooth round trip timers maintained for the various NFS call types (see example on page 105). Examining these values can reveal which NFS filesystems are experiencing better performance than others.

A network trace can help determine if application delays are caused by the client not sending requests quickly enough or the server taking too long to reply.

The tusc utility can show which system calls are experiencing delays. This data can help developers understand where in the application the delay is occurring.

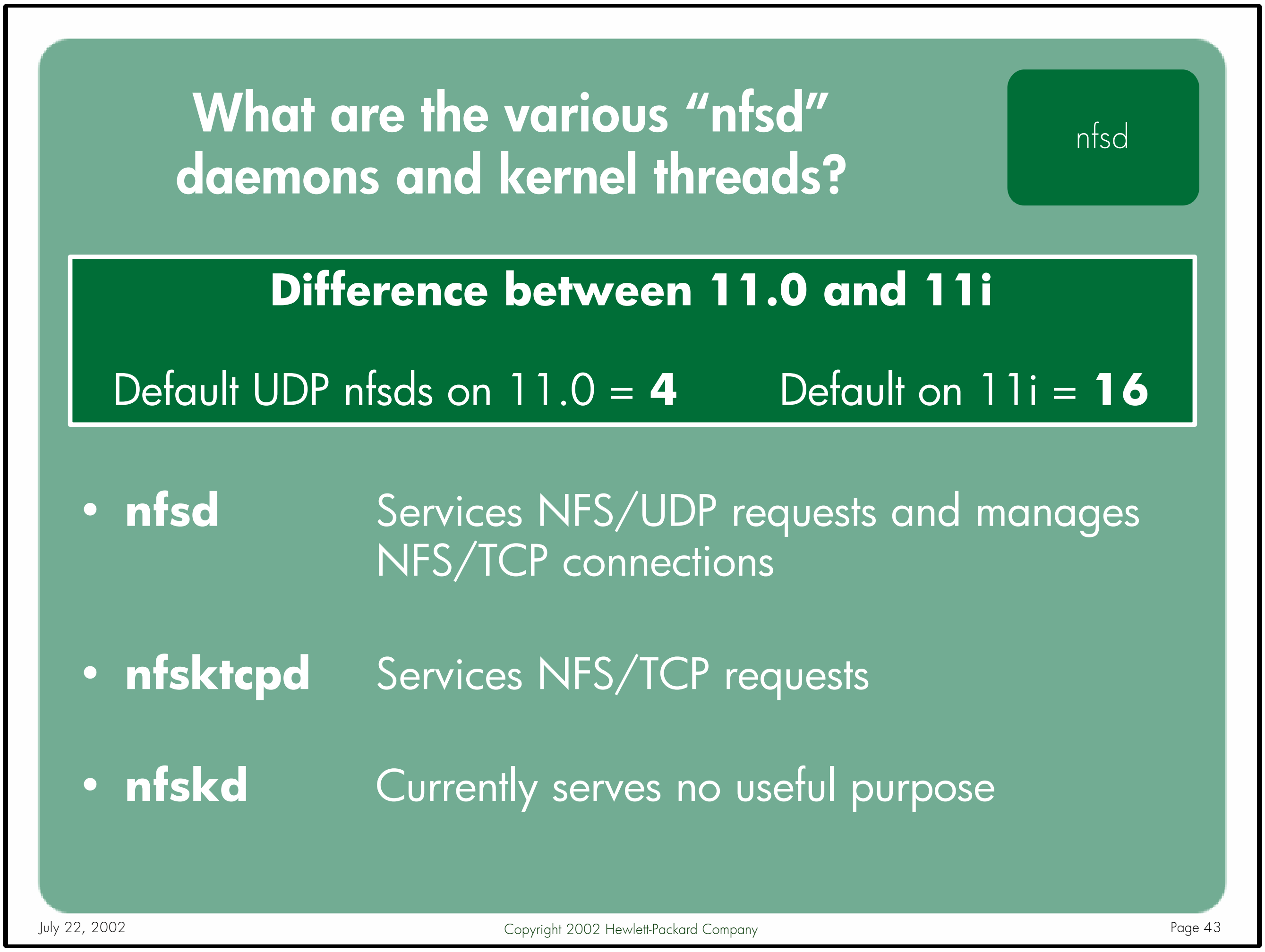
Kernel profiling information can reveal which functions the kernel is spending the bulk of its time running. This data can only be collected and analyzed by qualified HP support personnel.



nfsd daemons and threads

- What are they?
- How do they work in the **READ** and **WRITE** cases?
- Why are more nfsds launched than configured in NUM_NFSD?
- How many threads service TCP requests?
- How many nfsds should you run?
- Troubleshooting

Notes: This section discusses the various daemons and threads that handle server-side NFS requests. It describes how these daemons and threads work in both the read and write scenarios. It explains how they work differently when servicing NFS/UDP requests vs. NFS/TCP requests. It also covers the various factors involved in selecting the appropriate number of nfsd daemons to run on a given server. Finally, it describes the tools and methods available for troubleshooting nfsd and nfsktcpd problems.



What are the various "nfsd" daemons and kernel threads?

nfsd

Difference between 11.0 and 11i

Default UDP nfsds on 11.0 = **4** Default on 11i = **16**

- **nfsd** Services NFS/UDP requests and manages NFS/TCP connections
- **nfsktcpd** Services NFS/TCP requests
- **nfskd** Currently serves no useful purpose

July 22, 2002

Page 43

Notes:

nfsd

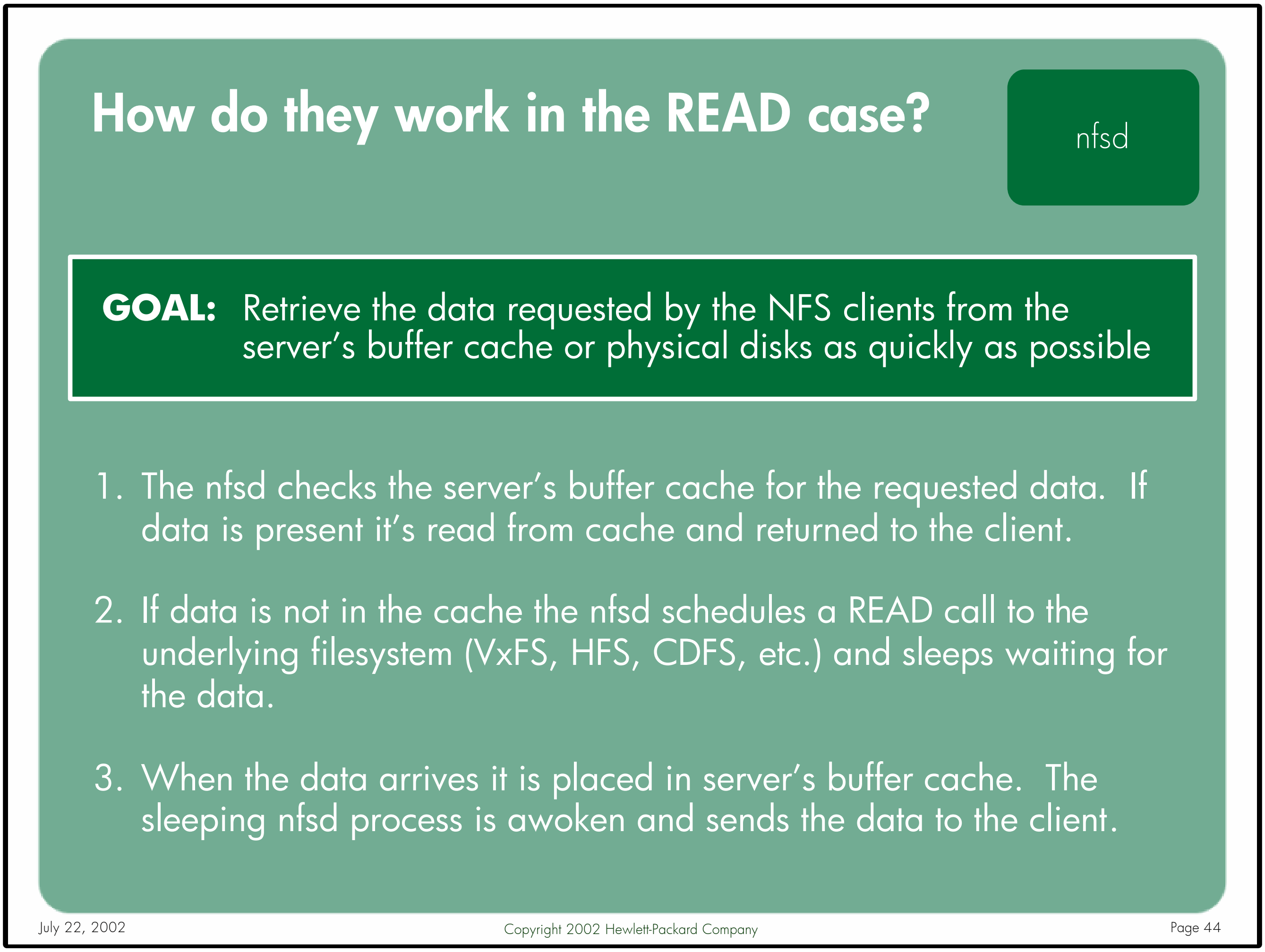
The nfsd daemons are primarily used to service NFS/UDP requests. When NFS/TCP is enabled, a single nfsd is also used to manage the NFS/TCP connection establishment and teardown functions.

nfsktcpd

The nfsktcpd daemon is the process where NFS/TCP kernel threads associate themselves. The process is dynamically created by the NFS server's kernel the first time it receives a request from an NFS/TCP client. Since the process is created in the kernel, the parent pid will be "0" and it cannot be killed.

nfskd

The nfskd daemon currently serves no useful purpose on HP-UX systems. It was originally intended to be the UDP-equivalent of nfsktcpd – i.e. the daemon process where NFS/UDP kernel threads would associate themselves. HP has not implemented server-side UDP kernel threads at this time so this daemon remains idle. Like nfsktcpd, it is dynamically created and cannot be killed.



How do they work in the READ case?

nfsd

GOAL: Retrieve the data requested by the NFS clients from the server's buffer cache or physical disks as quickly as possible

1. The nfsd checks the server's buffer cache for the requested data. If data is present it's read from cache and returned to the client.
2. If data is not in the cache the nfsd schedules a READ call to the underlying filesystem (VxFS, HFS, CDFS, etc.) and sleeps waiting for the data.
3. When the data arrives it is placed in server's buffer cache. The sleeping nfsd process is awoken and sends the data to the client.

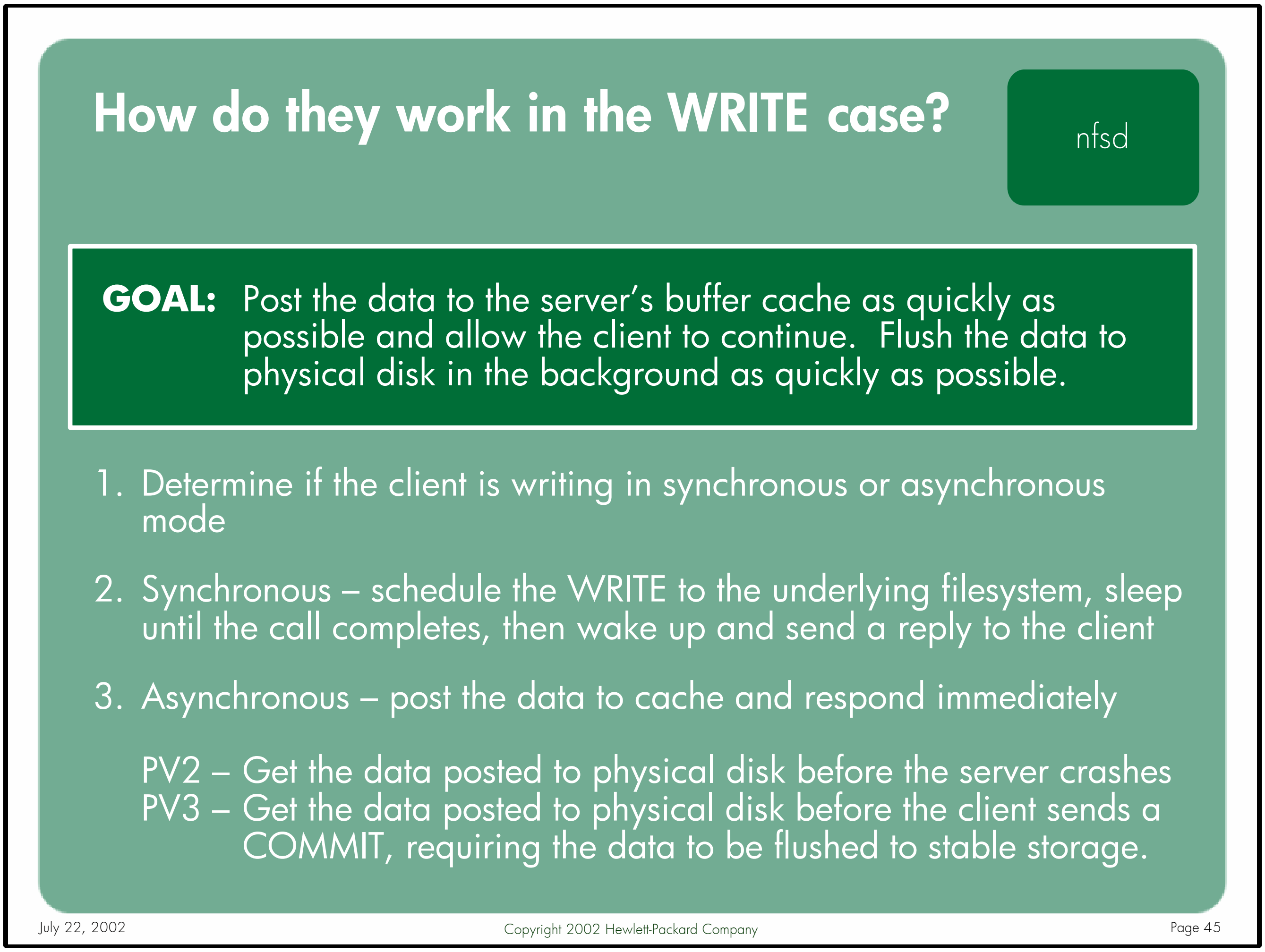
July 22, 2002

Page 44

Notes: When a client system needs to read data from an NFS mounted file, it will first check its local buffer cache to see if the block of data is already present in cache. If it is then the read is satisfied without generating an NFS request. If the data is not present then an NFS READ call will be made to retrieve this data.

When the server receives the READ request it first checks it's buffer cache to see if the requested data is already present in it's cache. If so, it sends the data back to the client without performing a physical disk I/O.

If the requested data is not present in the server's buffer cache, the nfsd schedules a READ from the underlying filesystem (i.e. VxFS, HFS, CDFS, etc.) and blocks waiting for the disk subsystem to retrieve the data. When the data is returned from the filesystem it is placed in server's buffer cache. The sleeping nfsd process is notified that the data is available. The nfsd retrieves the data from cache and sends it to the waiting NFS client.



How do they work in the WRITE case?

nfsd

GOAL: Post the data to the server’s buffer cache as quickly as possible and allow the client to continue. Flush the data to physical disk in the background as quickly as possible.

1. Determine if the client is writing in synchronous or asynchronous mode
2. Synchronous – schedule the WRITE to the underlying filesystem, sleep until the call completes, then wake up and send a reply to the client
3. Asynchronous – post the data to cache and respond immediately

   PV2 – Get the data posted to physical disk before the server crashes
   PV3 – Get the data posted to physical disk before the client sends a COMMIT, requiring the data to be flushed to stable storage.

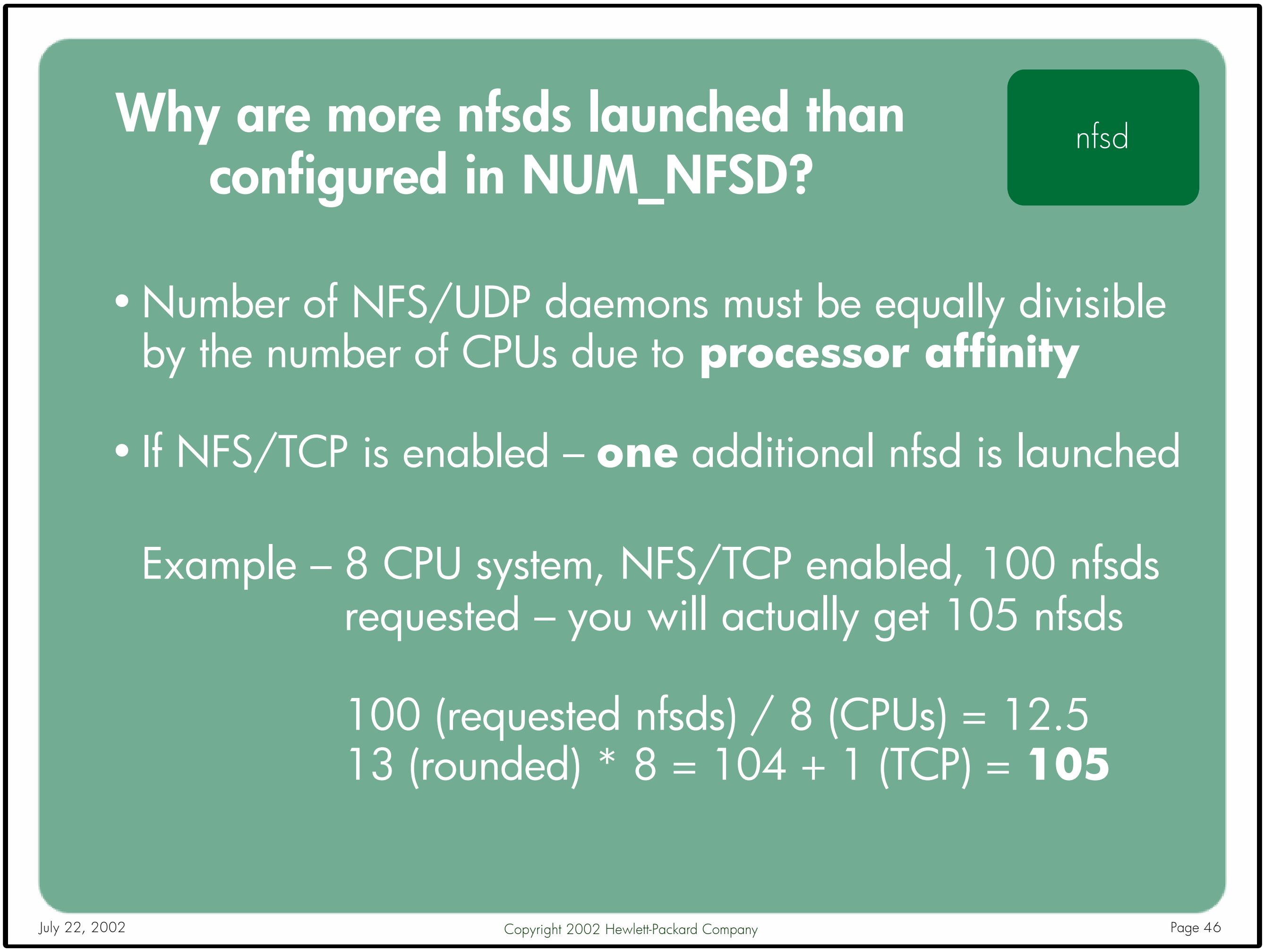
July 22, 2002

Page 45

Notes: The nfsds (or kernel threads) behave differently depending up on whether the NFS client is sending its write requests asking for synchronous or asynchronous semantics.

In synchronous mode (the default behavior for NFS PV2), the nfsds take the write data from the request and schedule a disk I/O operation to get the data posted to stable storage as quickly as possible. The nfsd blocks waiting for the I/O to complete before responding to the client and allowing it to continue.

In asynchronous mode (the default behavior for PV3, and can be enabled on PV2 by exporting the filesystem with the “–async” option), the nfsd takes the data from the request, posts it to the server’s buffer cache, and sends a reply immediately allowing the client process to continue before the data is actually posted to stable storage. In the PV2 case, the NFS server’s goal is to get the data posted to physical disk before the server experiences a system crash. In the PV3 case, the NFS server tries to flush the data to stable storage before the client sends a COMMIT request, since nfsd is not allowed to reply to a COMMIT request until all data specified in the request is posted to physical disk.



Why are more nfsds launched than configured in NUM_NFSD?

nfsd

- Number of NFS/UDP daemons must be equally divisible by the number of CPUs due to **processor affinity**
- If NFS/TCP is enabled – **one** additional nfsd is launched

Example – 8 CPU system, NFS/TCP enabled, 100 nfsds requested – you will actually get 105 nfsds

100 (requested nfsds) / 8 (CPUs) = 12.5
13 (rounded) * 8 = 104 + 1 (TCP) = **105**

July 22, 2002 Page 46

Notes: There are several factors that determine the number of nfsd processes that get launched on any given HP-UX 11.0 or 11i NFS server:

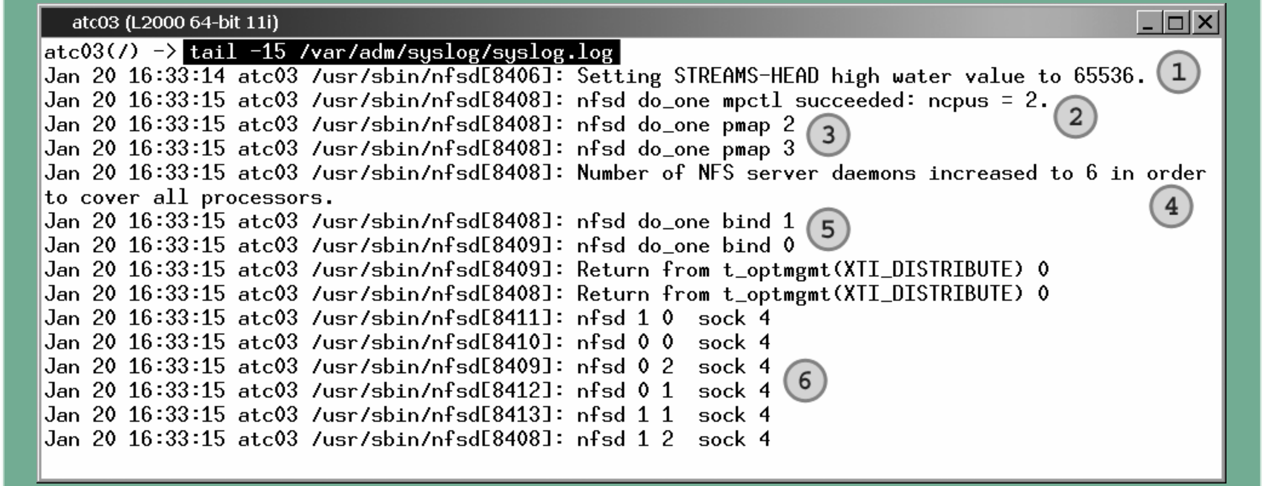
- The number of daemons requested on the nfsd command line or specified via the NUM_NFSD variable in the `/etc/rc.config.d/nfsconf` file
- The number of CPUs in the server
- Whether NFS/TCP is enabled on the server

The number of NFS/UDP daemons must be a number equally divisible by the number of CPUs because the kernel enables processor affinity for the nfsds at initialization time. The affinity mode is advisory, meaning that the daemons can run on a CPU other than the one they were originally bound to. The kernel automatically increases the number of nfsds to the next number which is evenly divisible by the number of CPUs.

If NFS/TCP is enabled, an additional nfsd daemon is launched to manage the TCP connection establishment and teardown functions.

What happens at nfsd start time?

nfsd



```
atc03(/) -> tail -15 /var/adm/syslog/syslog.log
Jan 20 16:33:14 atc03 /usr/sbin/nfsd[8406]: Setting STREAMS-HEAD high water value to 65536.
Jan 20 16:33:15 atc03 /usr/sbin/nfsd[8408]: nfsd do_one mpctl succeeded: ncpus = 2.
Jan 20 16:33:15 atc03 /usr/sbin/nfsd[8408]: nfsd do_one pmap 2
Jan 20 16:33:15 atc03 /usr/sbin/nfsd[8408]: nfsd do_one pmap 3
Jan 20 16:33:15 atc03 /usr/sbin/nfsd[8408]: Number of NFS server daemons increased to 6 in order
to cover all processors.
Jan 20 16:33:15 atc03 /usr/sbin/nfsd[8408]: nfsd do_one bind 1
Jan 20 16:33:15 atc03 /usr/sbin/nfsd[8409]: nfsd do_one bind 0
Jan 20 16:33:15 atc03 /usr/sbin/nfsd[8409]: Return from t_optmgmt(XTI_DISTRIBUTE) 0
Jan 20 16:33:15 atc03 /usr/sbin/nfsd[8408]: Return from t_optmgmt(XTI_DISTRIBUTE) 0
Jan 20 16:33:15 atc03 /usr/sbin/nfsd[8411]: nfsd 1 0  sock 4
Jan 20 16:33:15 atc03 /usr/sbin/nfsd[8410]: nfsd 0 0  sock 4
Jan 20 16:33:15 atc03 /usr/sbin/nfsd[8409]: nfsd 0 2  sock 4
Jan 20 16:33:15 atc03 /usr/sbin/nfsd[8412]: nfsd 0 1  sock 4
Jan 20 16:33:15 atc03 /usr/sbin/nfsd[8413]: nfsd 1 1  sock 4
Jan 20 16:33:15 atc03 /usr/sbin/nfsd[8408]: nfsd 1 2  sock 4
```

1. Stream Head Buffer Size Calculated

2. Per-CPU nfsd Pools Created

3. rpcbind(1M) Bindings Established

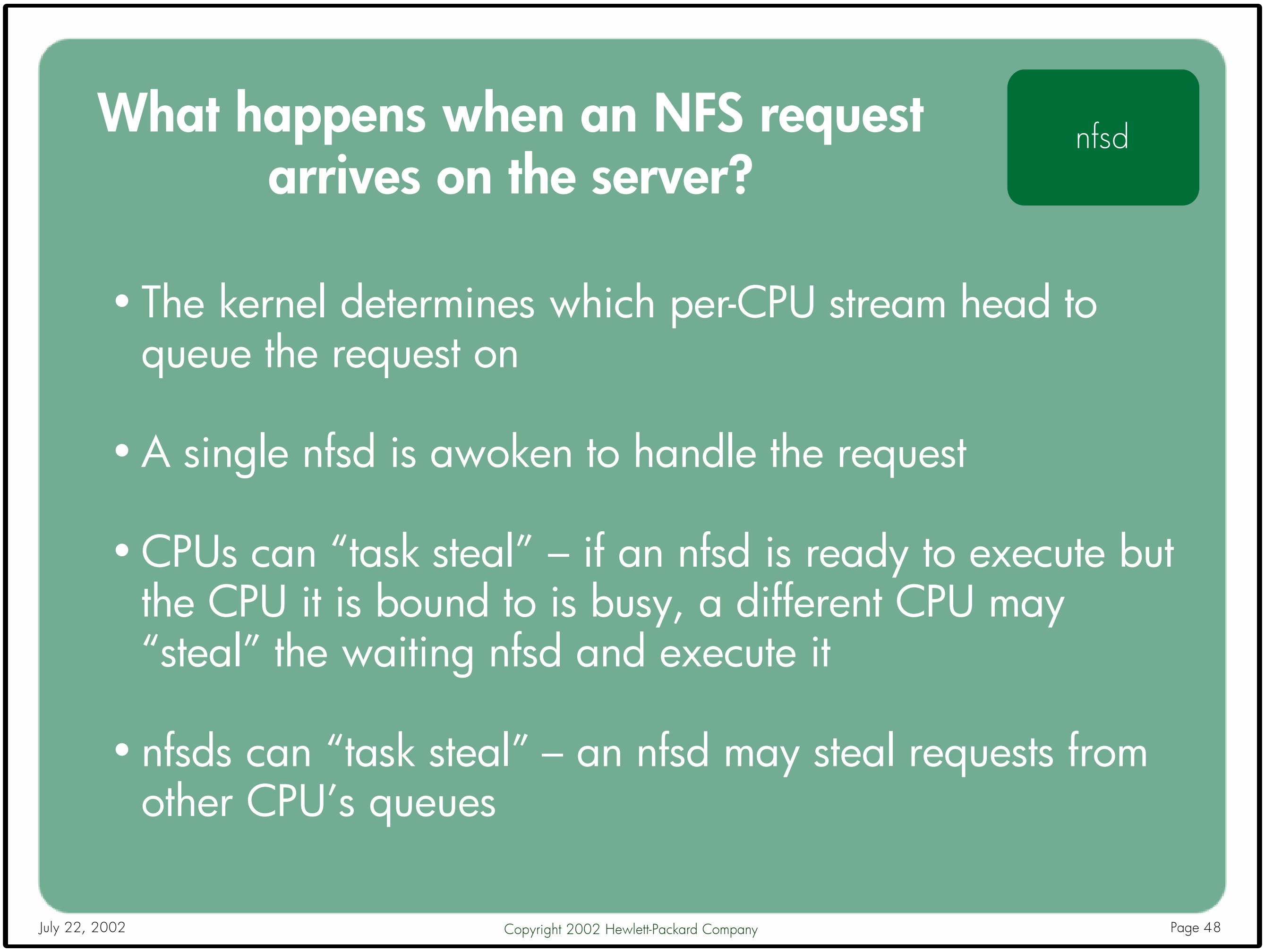
4. Number of nfsds Increased if Needed

5. Stream Head Replicated per-CPU

6. nfsds Bind to per-CPU Pools

Notes: When the nfsds are started, several important things happen:

1. **Stream Head Buffer Size Calculated**
   The size of the stream head buffer is calculated based on the number of nfsds launched. This buffer size ranges between 64KB and 512KB. The size increases by 8KB for each new nfsd launched. This means that at least 64 nfsds must be running in order for the kernel to allocate the maximum allowable buffer space for each stream head (i.e. 64 nfsds * 8KB = 512KB).
2. **Per-CPU nfsd Pools Created**
   A separate pool of nfsds is created for each CPU.
3. **rpcbind(1M) Bindings Established**
   The daemons register support for NFS PV2 and PV3 for both UDP and TCP transport protocols.
4. **Number of nfsds Increased if Needed**
   The kernel allocates an equal number of nfsds to each per-CPU pool.
5. **Stream Head Replicated per-CPU**
   A separate stream head is allocated for each CPU, allowing each CPU to maintain a separate queue of requests to process.
6. **nfsds Bind to per-CPU Pools**
   The individual nfsd daemons are evenly distributed among the per-CPU pools.



What happens when an NFS request arrives on the server?

nfsd

- The kernel determines which per-CPU stream head to queue the request on
- A single nfsd is awoken to handle the request
- CPUs can “task steal” – if an nfsd is ready to execute but the CPU it is bound to is busy, a different CPU may “steal” the waiting nfsd and execute it
- nfsds can “task steal” – an nfsd may steal requests from other CPU’s queues

July 22, 2002

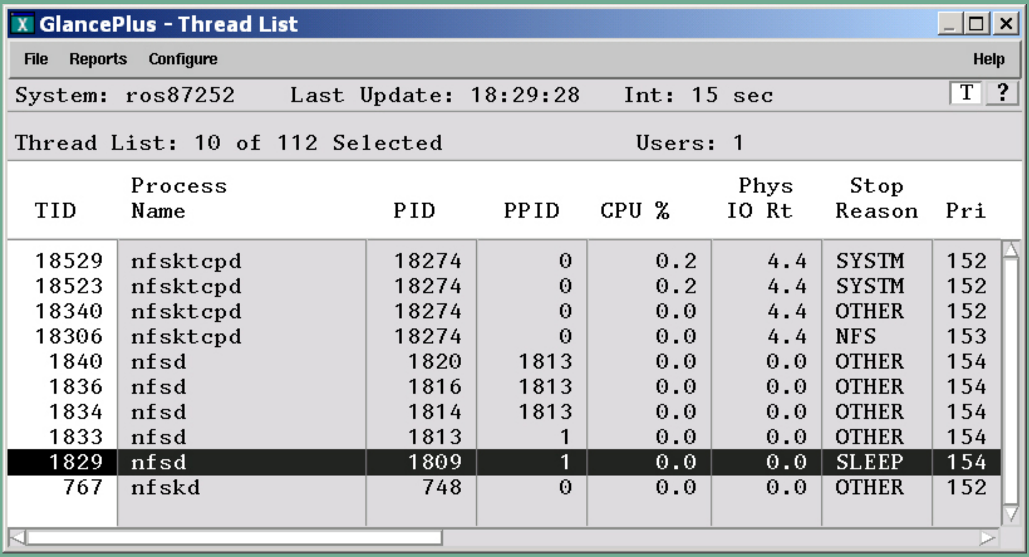
Page 48

Notes: When an NFS request arrives on a multi-CPU server, the kernel must first determine which CPU’s stream head to queue the request on. Remember that the server creates separate pools of nfsds for each CPU and allocates an equal number of daemons to each queue. Once the kernel decides where to queue the request, a single nfsd is awoken to process the request.

If an nfsd is ready to execute but the CPU it is associated with is busy, an available CPU may “steal” the nfsd and execute it. In addition, once an nfsd finishes processing a request, before returning to sleep it will check the queue of the CPU it is bound to to see if any other requests are waiting to be processed. If it finds a request waiting it will service it. If the nfsd doesn’t find a request in the CPU’s queue, the nfsd will then check the queues of the other CPUs to see if any requests are waiting in their queues. If it finds a request waiting to be serviced, the nfsd may “steal” the request from the queue of a different CPU and execute it. This is commonly referred to as “task stealing.”

Which nfsd is used by NFS/TCP?

nfsd

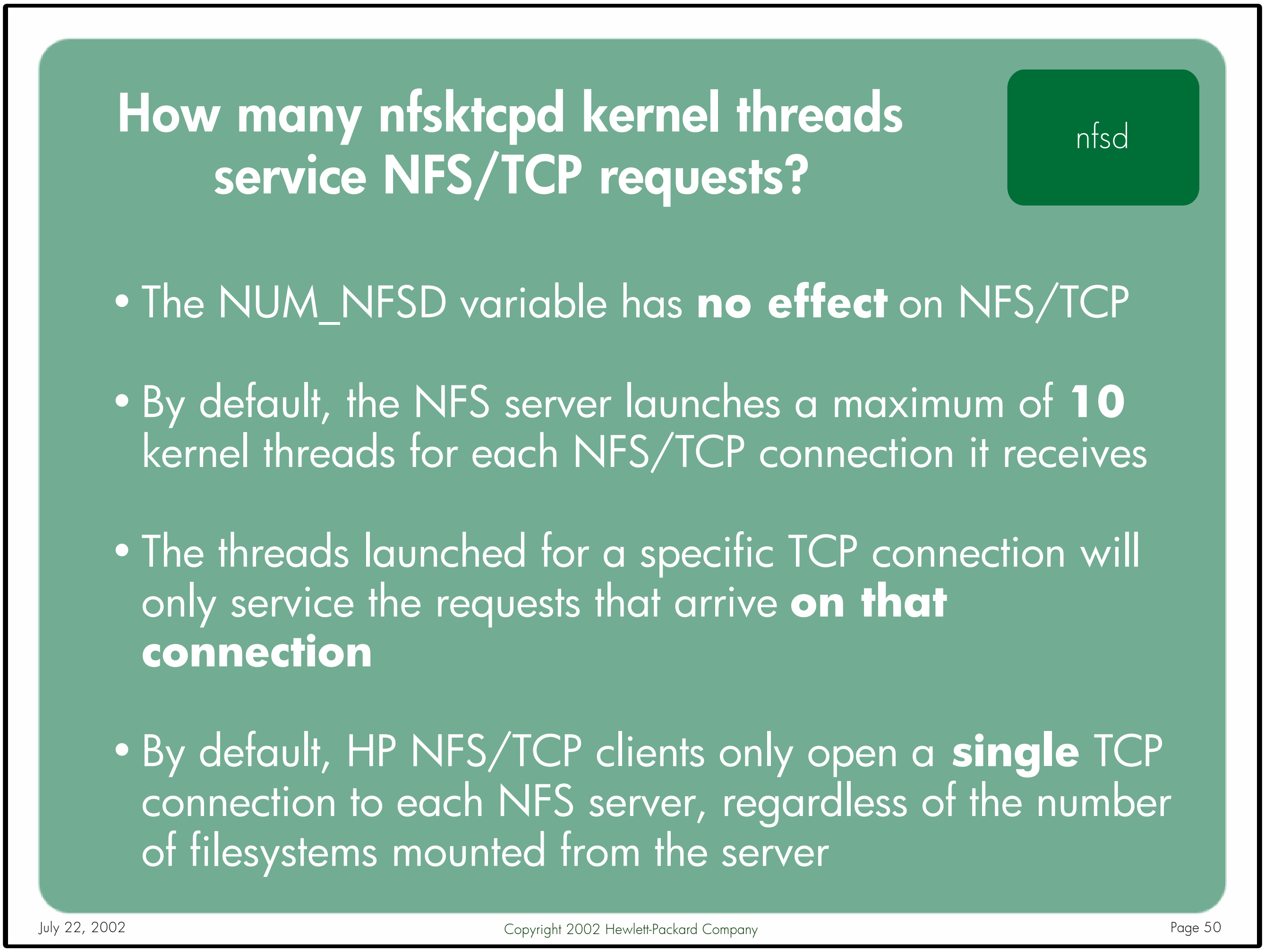


```
System: ros87252     Last Update: 18:29:28    Int: 15 sec

Thread List: 10 of 112 Selected                Users: 1

          Process                                             Phys     Stop
  TID     Name                PID      PPID     CPU %    IO Rt    Reason   Pri

  18529   nfsktcpd           18274         0       0.2      4.4   SYSTM    152
  18523   nfsktcpd           18274         0       0.2      4.4   SYSTM    152
  18340   nfsktcpd           18274         0       0.0      4.4   OTHER    152
  18306   nfsktcpd           18274         0       0.0      4.4   NFS      153
   1840   nfsd                1820      1813       0.0      0.0   OTHER    154
   1836   nfsd                1816      1813       0.0      0.0   OTHER    154
   1834   nfsd                1814      1813       0.0      0.0   OTHER    154
   1833   nfsd                1813         1       0.0      0.0   OTHER    154
   1829   nfsd                1809         1       0.0      0.0   SLEEP    154
    767   nfskd                748         0       0.0      0.0   OTHER    152
```

The nfsd process used by NFS/TCP has a parent process ID of 1 (init) and has no child processes. In this example – TID 1829 (PID 1809).

Notes: Looking at the above screen output, you see the entire list of daemons and kernel threads used for servicing NFS/UDP and NFS/TCP requests. The nfsds responsible for servicing UDP requests are using TID numbers 1833, 1834, 1836, and 1840. The nfsd process used for managing NFS/TCP connections is distinguishable from the NFS/UDP nfsds by the fact that it has a parent process ID of 1 (init) and it has no child processes. In the above screenshot, there is only one nfsd daemon that meets these criteria – the one using TID 1829 (process ID 1809).

There are currently four nfsktcpd kernel threads servicing NFS/TCP requests – 18306, 18340, 18523, and 18529. Finally, there is the nfskd daemon, which currently serves no purpose on HP-UX systems.



How many nfsktcpd kernel threads service NFS/TCP requests?

nfsd

- The NUM_NFSD variable has **no effect** on NFS/TCP
- By default, the NFS server launches a maximum of **10** kernel threads for each NFS/TCP connection it receives
- The threads launched for a specific TCP connection will only service the requests that arrive **on that connection**
- By default, HP NFS/TCP clients only open a **single** TCP connection to each NFS server, regardless of the number of filesystems mounted from the server

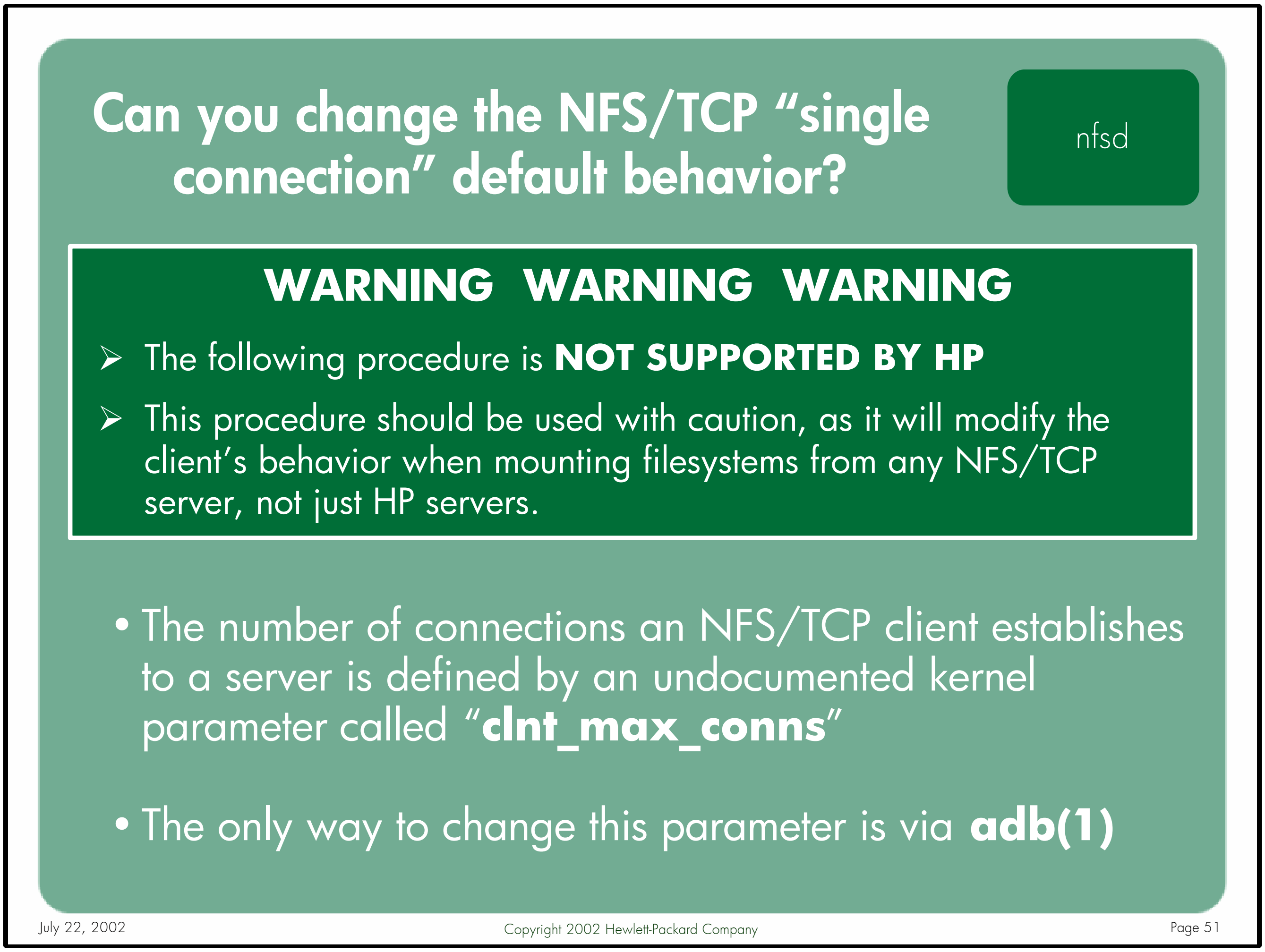
July 22, 2002

Page 50

Notes: The total number of server-side kernel threads running at any given time to handle inbound NFS/TCP requests is directly related to the number of established NFS/TCP connections from all clients. By default, the server will launch a maximum of 10 kernel threads to service requests for each TCP connection. Since NFS/TCP clients only open a single connection to each NFS server (by default), this limits the server to using only 10 kernel threads to service requests from any specific NFS client, regardless of how many NFS filesystems the client mounts from the server.

Under most conditions the single TCP connection and 10 threads per connection limits should not be an issue. However, in the case where large, multi-processor clients are mounting many NFS/TCP filesystems from a single server and multiplexing large numbers of requests across that connection, these characteristics could result in a potential bottleneck.

The threads that are launched to service requests for a given NFS/TCP connection are dedicated to that connection and cannot be used to process inbound requests on other NFS/TCP connections.



Can you change the NFS/TCP “single connection” default behavior?

nfsd

WARNING WARNING WARNING

- ➢ The following procedure is **NOT SUPPORTED BY HP**
- ➢ This procedure should be used with caution, as it will modify the client’s behavior when mounting filesystems from any NFS/TCP server, not just HP servers.

- The number of connections an NFS/TCP client establishes to a server is defined by an undocumented kernel parameter called “**clnt_max_conns**”
- The only way to change this parameter is via **adb(1)**

July 22, 2002

Page 51

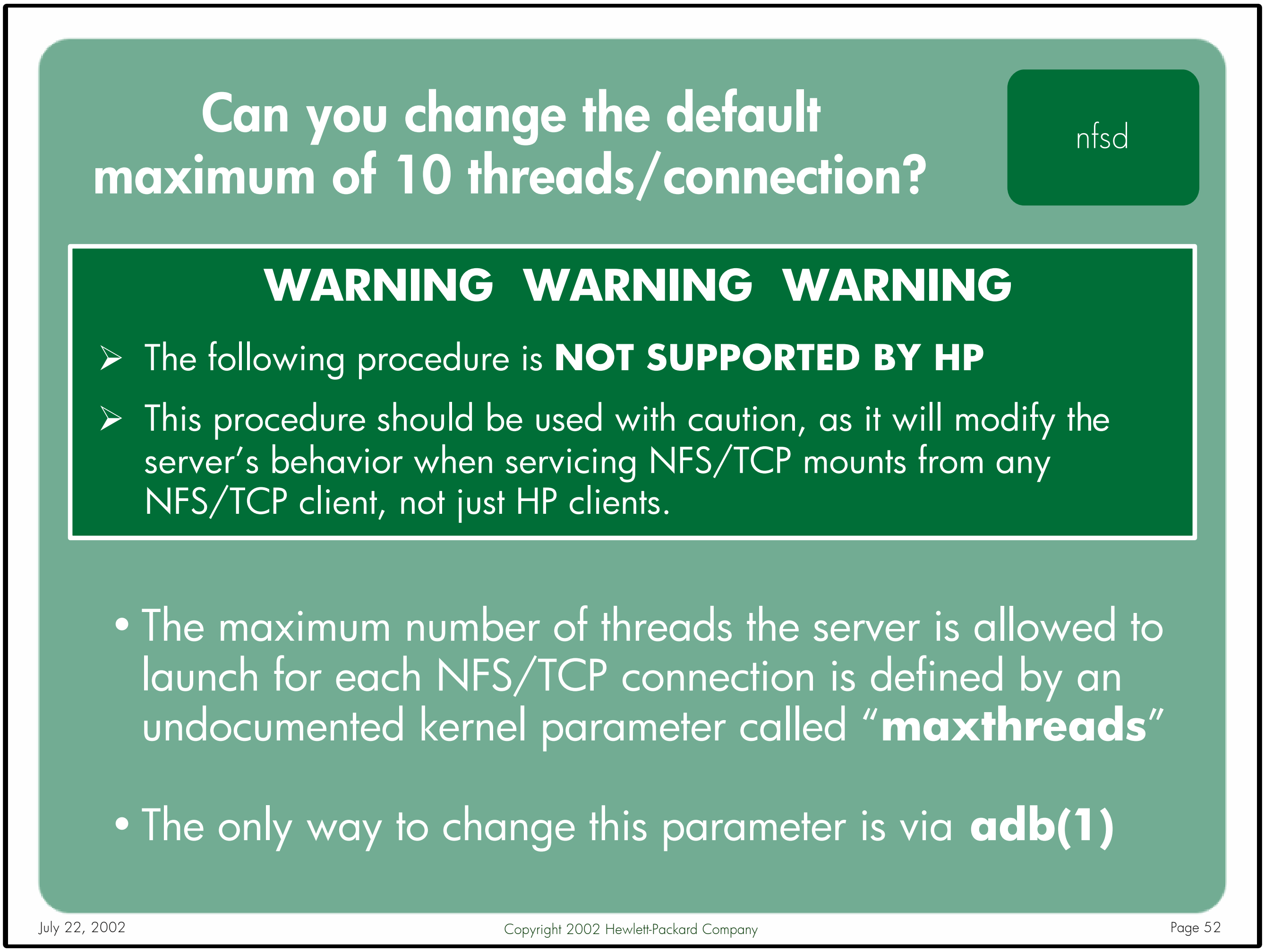
Notes: By default, HP-UX NFS/TCP clients only open a single connection to each NFS server, regardless of how many NFS filesystems the client mounts from the server. This behavior can be changed by modifying an undocumented kernel variable with adb(1). This procedure is **NOT SUPPORTED BY HP**. Use at your own risk.

To force an HP-UX 11.0 or 11i client to open more than one NFS/TCP connection to each server it mounts filesystems from, log into the client as a root user and type:

```
# echo “clnt_max_conns/W 0d2” | adb -w /stand/vmunix /dev/kmem
# echo “clnt_max_conns?W 0d2” | adb -w /stand/vmunix /dev/kmem
```

In the above example, the “0d2” parameter instructs the client to open 2 connections per server. The above commands modify both the on-disk kernel file and kernel memory so the change takes effect immediately, and will remain in effect even if the client system is rebooted. These commands would need to be repeated if the kernel is rebuilt, either manually or via a kernel patch installation.

To return to the default behavior, use “0d1” in the above commands.



Can you change the default maximum of 10 threads/connection?

nfsd

WARNING WARNING WARNING

- ➢ The following procedure is **NOT SUPPORTED BY HP**
- ➢ This procedure should be used with caution, as it will modify the server’s behavior when servicing NFS/TCP mounts from any NFS/TCP client, not just HP clients.

- The maximum number of threads the server is allowed to launch for each NFS/TCP connection is defined by an undocumented kernel parameter called “**maxthreads**”
- The only way to change this parameter is via **adb(1)**

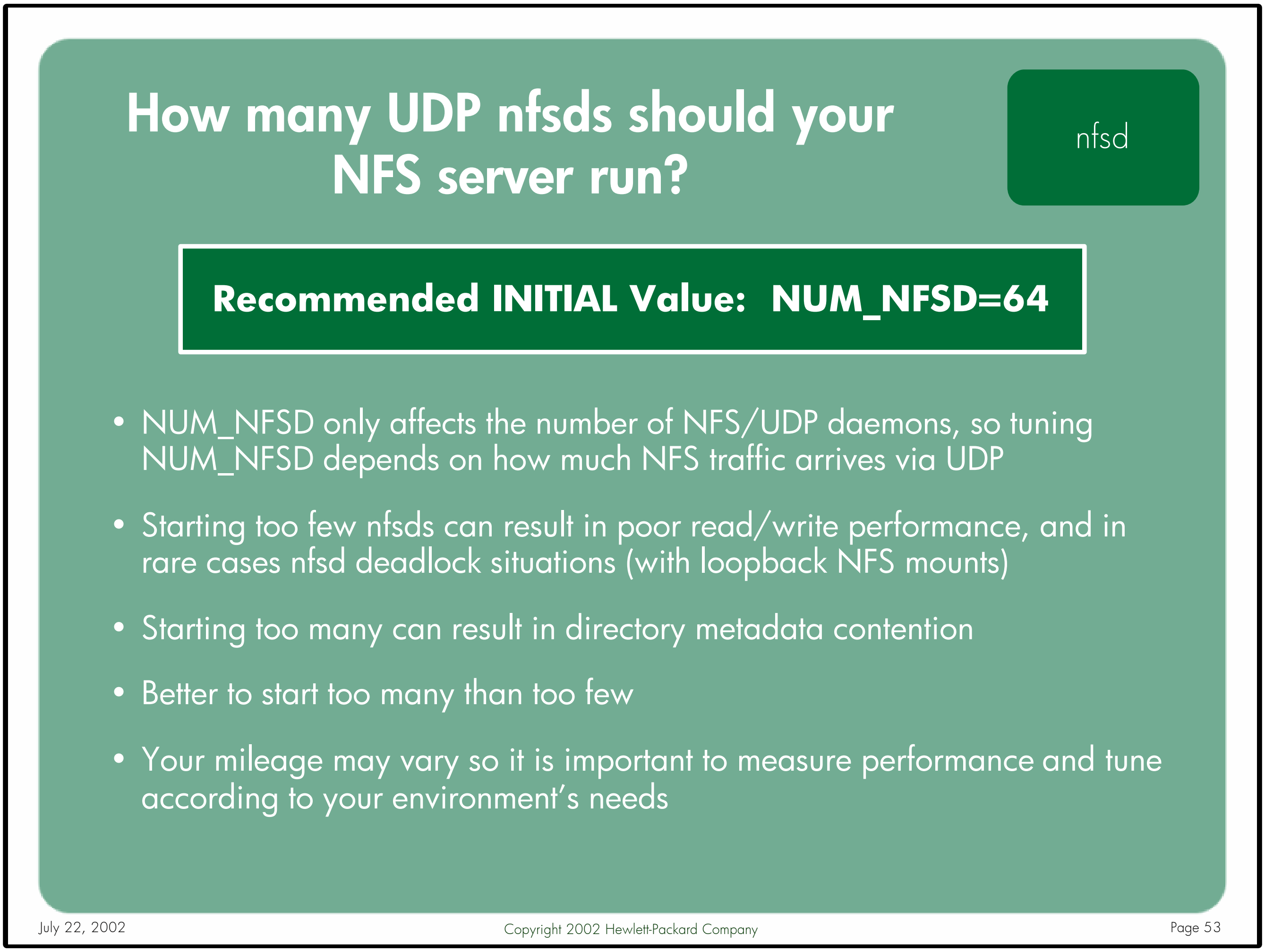
July 22, 2002 Copyright 2002 Hewlett-Packard Company Page 52

Notes: By default, HP-UX NFS/TCP servers launch a maximum of 10 kernel threads to service requests for each NFS/TCP connection, regardless of how many NFS filesystems the client accesses on the server. This behavior can be changed by modifying an undocumented kernel variable with adb(1). This procedure is **NOT SUPPORTED BY HP**. Use at your own risk.

To allow an HP-UX 11.0 or 11i server to launch more than 10 NFS/TCP threads per connection, log into the server as a root user and type:

```
# echo “maxthreads/W 0d20” | adb -w /stand/vmunix /dev/kmem
# echo “maxthreads?W 0d20” | adb -w /stand/vmunix /dev/kmem
```

In the above example, the “0d20” parameter instructs the server to launch a maximum of 20 threads per connection. These commands modify both the on-disk kernel file and kernel memory so the change takes effect immediately, and remain in effect even if the server system is rebooted. The adb commands must repeated if the kernel is rebuilt, either manually or via a kernel patch installation. To return to the default behavior, use “0d10” in the above commands.



How many UDP nfsds should your NFS server run?

nfsd

Recommended INITIAL Value: NUM_NFSD=64

- NUM_NFSD only affects the number of NFS/UDP daemons, so tuning NUM_NFSD depends on how much NFS traffic arrives via UDP
- Starting too few nfsds can result in poor read/write performance, and in rare cases nfsd deadlock situations (with loopback NFS mounts)
- Starting too many can result in directory metadata contention
- Better to start too many than too few
- Your mileage may vary so it is important to measure performance and tune according to your environment's needs

July 22, 2002

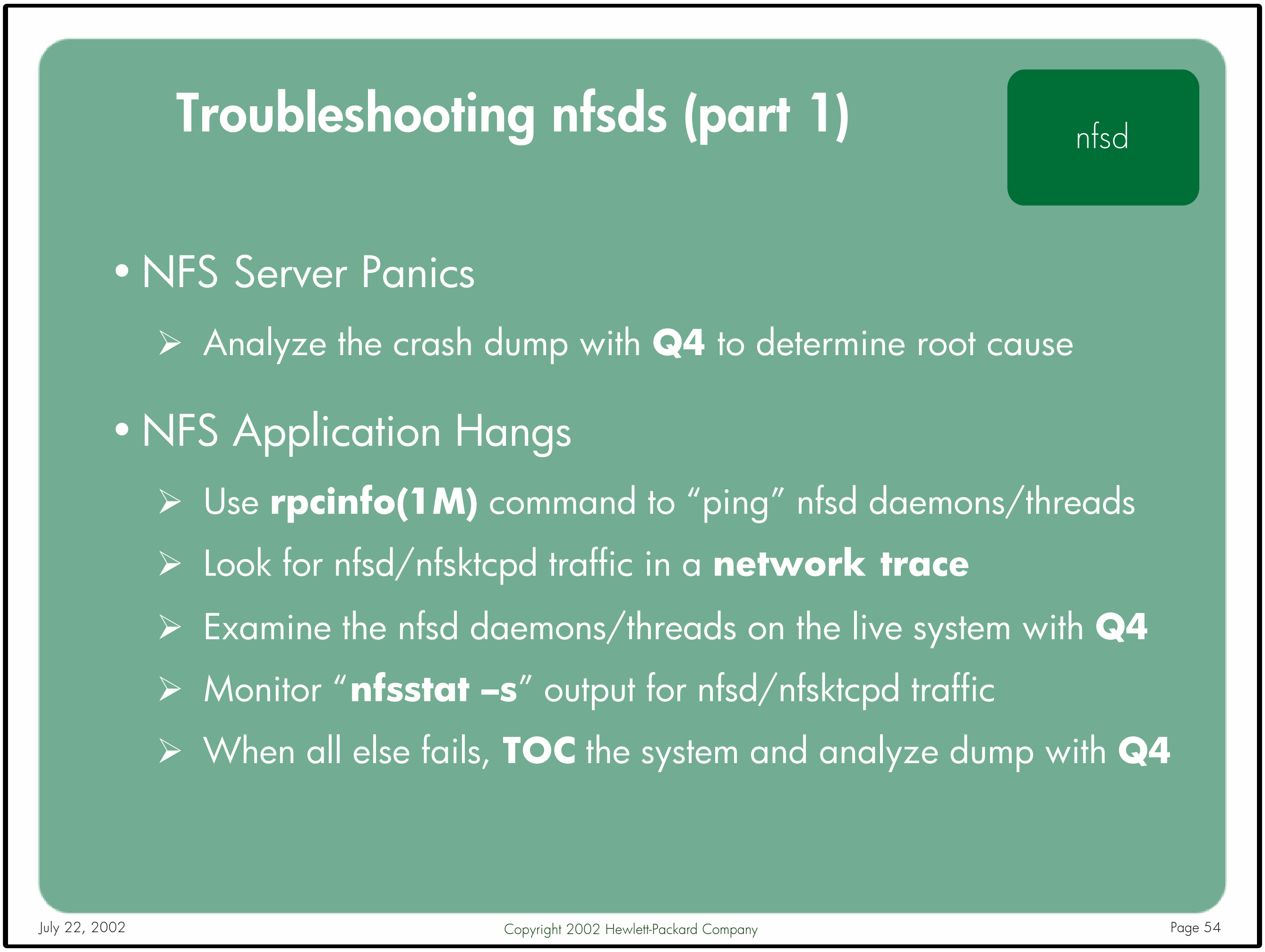
Page 53

Notes: The number of nfsds used to service NFS/UDP requests is configured via the NUM_NFSD variable in the `/etc/rc.config.d/nfsconf` file. This variable has no impact on the number of NFS/TCP kernel threads launched by the server.

Starting too few nfsds can result in poor NFS/UDP performance because not enough daemons are available to service the requests from all NFS clients. Starting too many can (in some cases) result in poor performance when many of the nfsd daemons are all trying to update the metadata contents of a shared directory – particularly directories containing thousands of files.

The best course of action is to choose a reasonable starting point for the number of nfsds and then experiment with raising and lowering the number until the optimal value is identified. There is typically very little overhead involved in launching too many nfsd daemons, so don't be afraid to experiment with larger numbers.

For most HP-UX servers a good starting point for the number of nfsds is 64, as this will allow the server to allocate the maximum amount of stream head buffer memory for receiving inbound requests (refer to page 46 for more details).



Troubleshooting nfsds (part 1)

nfsd

- NFS Server Panics
  - ➢ Analyze the crash dump with **Q4** to determine root cause
- NFS Application Hangs
  - ➢ Use **rpcinfo(1M)** command to “ping” nfsd daemons/threads
  - ➢ Look for nfsd/nfsktcpd traffic in a **network trace**
  - ➢ Examine the nfsd daemons/threads on the live system with **Q4**
  - ➢ Monitor “**nfsstat –s**” output for nfsd/nfsktcpd traffic
  - ➢ When all else fails, **TOC** the system and analyze dump with **Q4**

July 22, 2002

Page 54

Notes: **NFS Server Panics**

Analyzing HP-UX system dumps is a complex and involved process, which is typically performed by experienced HP support personnel or lab engineers.

NFS Application Hangs

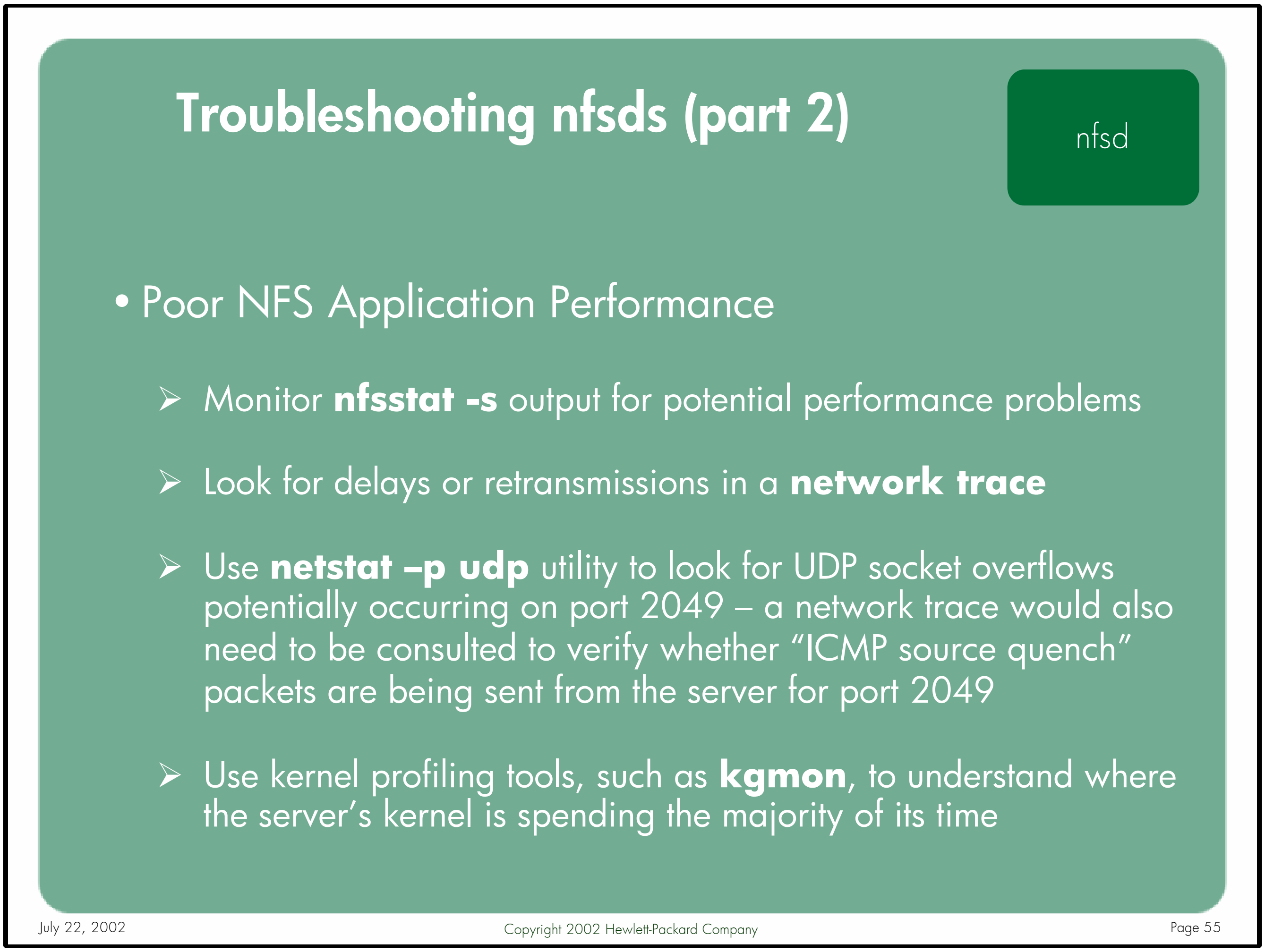
The rpcinfo(1M) command can be used to “ping” the UDP/TCP daemons and threads on the NFS server via the “-u” (UDP) and “-t” (TCP) options.

A network trace will show if any nfsds are responding to requests.

Q4 can be used by experienced system administrators or HP support personnel to examine the running nfsd daemons/threads on the live system.

Successive nfsstat outputs can show whether outbound NFS replies are being sent by the server.

When all else fails, the system might need to be TOC’d and the memory dump analyzed by HP support personnel.



Troubleshooting nfsds (part 2)

nfsd

- Poor NFS Application Performance
  - ➢ Monitor **nfsstat -s** output for potential performance problems
  - ➢ Look for delays or retransmissions in a **network trace**
  - ➢ Use **netstat –p udp** utility to look for UDP socket overflows potentially occurring on port 2049 – a network trace would also need to be consulted to verify whether "ICMP source quench" packets are being sent from the server for port 2049
  - ➢ Use kernel profiling tools, such as **kgmon**, to understand where the server's kernel is spending the majority of its time

July 22, 2002 Copyright 2002 Hewlett-Packard Company Page 55

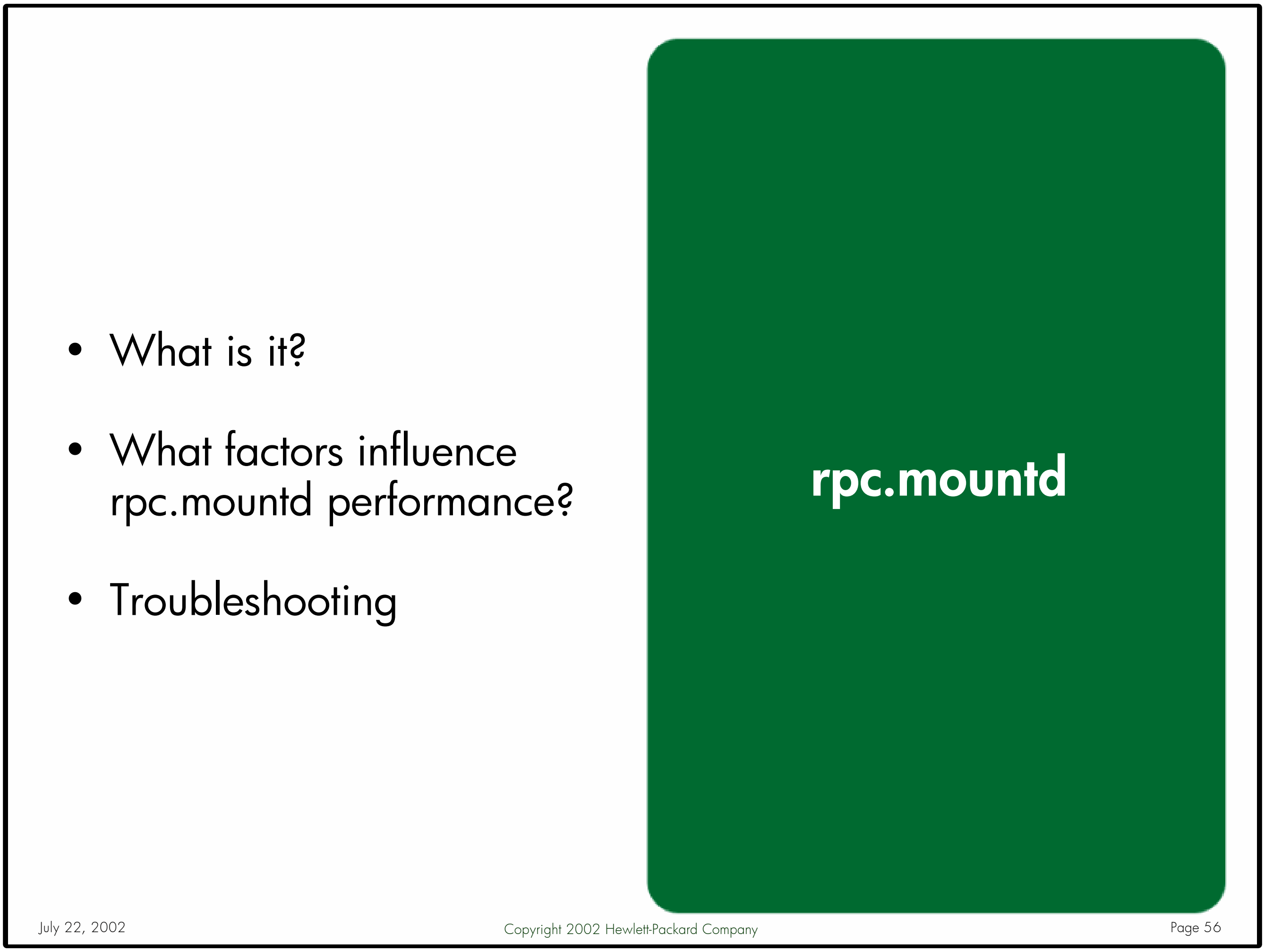
Notes: **Poor NFS Application Performance**

"nfsstat –s" output can reveal if badcalls, dupreqs, etc. are occurring. These can indicate that NFS requests are potentially being dropped by the client, the server, or the network resulting in poor performance.

A network trace can help determine if application delays are caused by the client not sending requests quickly enough or the server taking too long to reply.

The "netstat –p udp" command can be used to determine if UDP socket overflows are occurring on the NFS server. If the UDP socket used for NFS (port 2049) is overflowing, this could indicate that more nfsd daemons need to be launched to handle the inbound request rate. A network trace should be consulted to verify that "ICMP source quench" packets are being sent by the server for port 2049.

Kernel profiling information can reveal which functions the kernel is spending the bulk of its time running. This data can only be collected and analyzed by qualified HP support personnel.



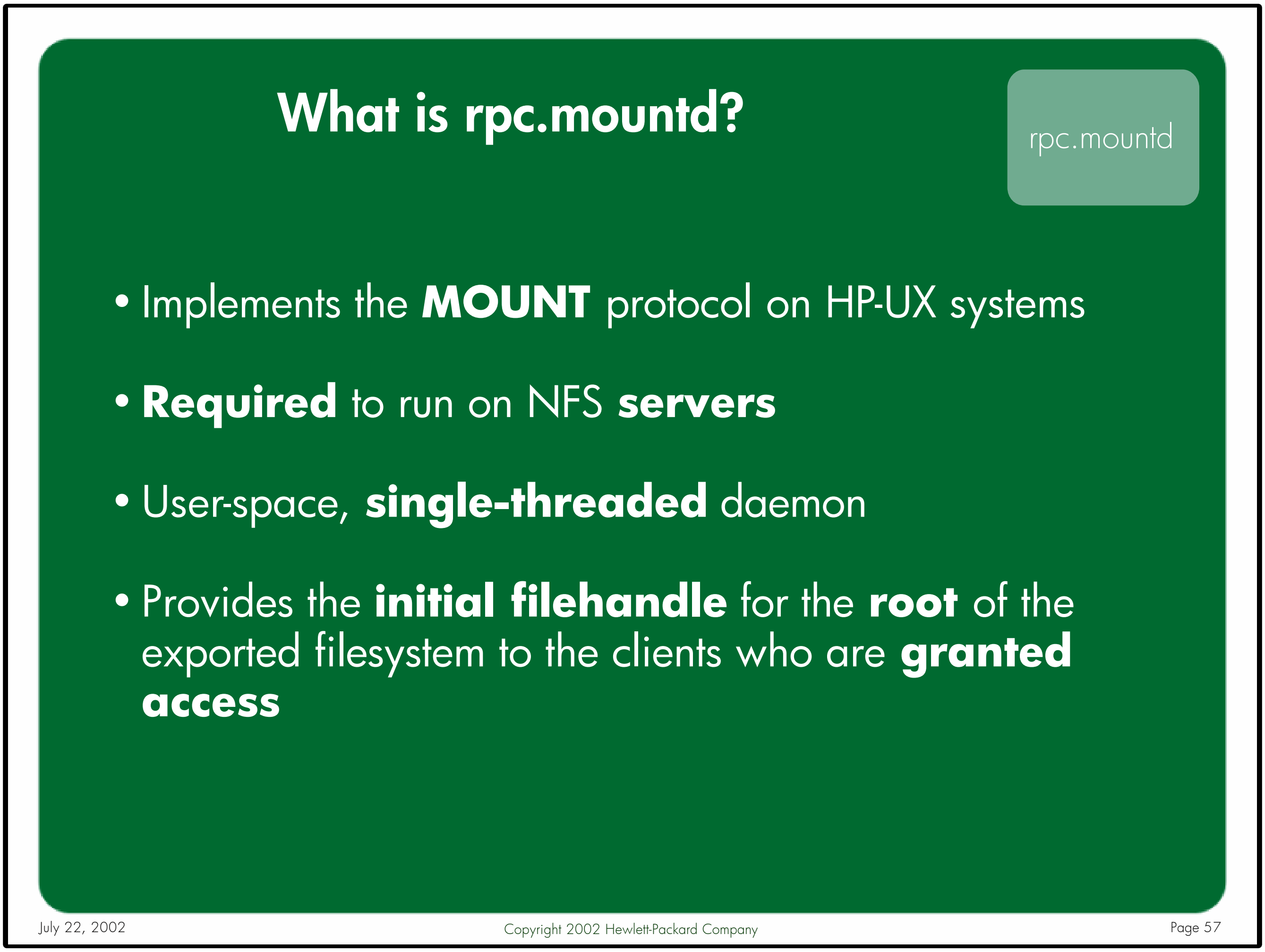
rpc.mountd

- What is it?
- What factors influence rpc.mountd performance?
- Troubleshooting

July 22, 2002

Page 56

Notes: This section describes the rpc.mountd daemon (commonly referred to as simply "mountd"), which is used to mount NFS filesystems. Included is a discussion of the ways you can tune your environment for optimal rpc.mountd performance, as well as some recommendations for troubleshooting NFS filesystem mounting problems.



What is rpc.mountd?

rpc.mountd

- Implements the **MOUNT** protocol on HP-UX systems
- **Required** to run on NFS **servers**
- User-space, **single-threaded** daemon
- Provides the **initial filehandle** for the **root** of the exported filesystem to the clients who are **granted access**

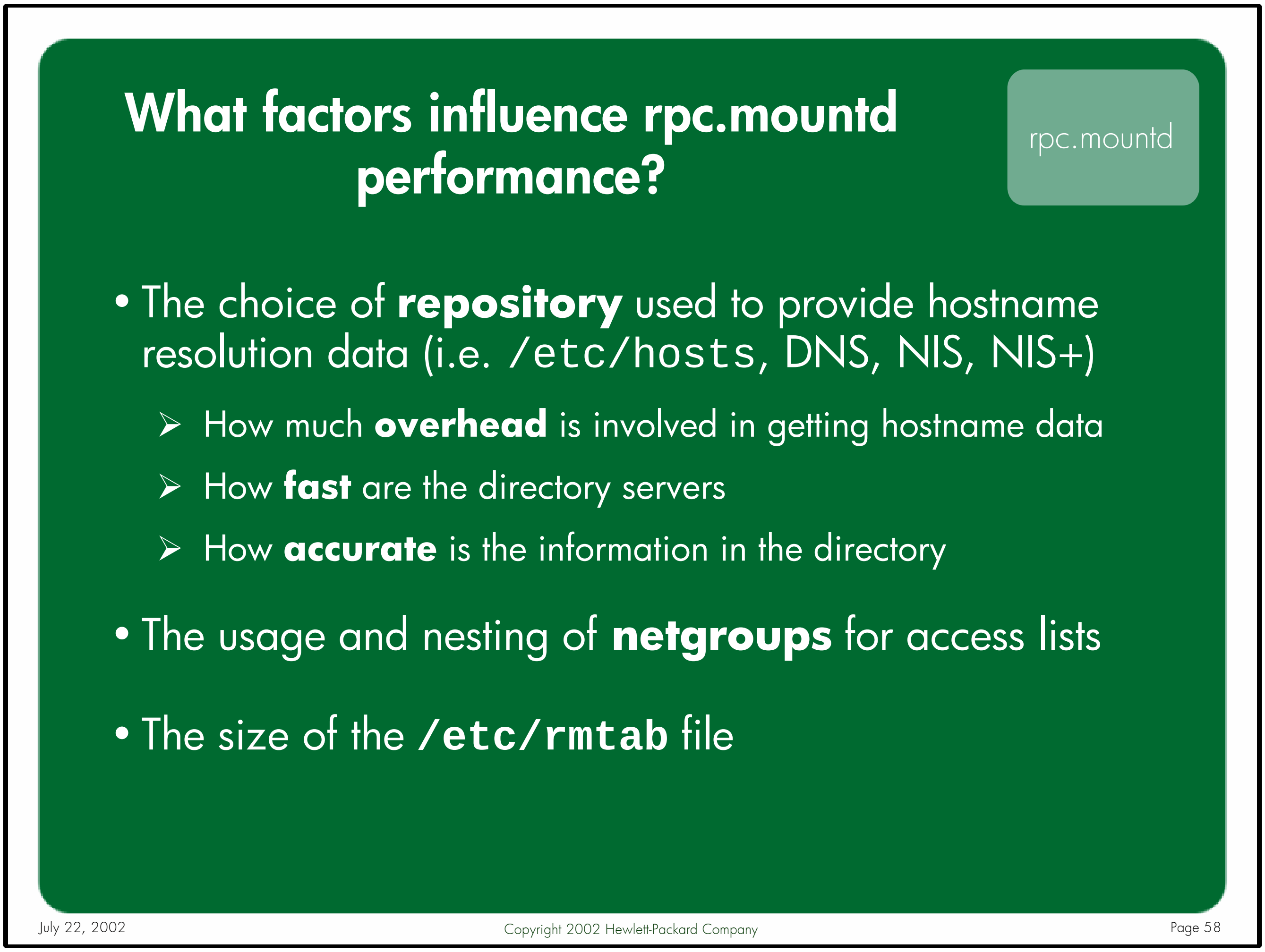
July 22, 2002

Page 57

Notes: Before an NFS client can begin reading from or writing to files residing on an NFS server, it first needs to contact the server and request permission to access the exported filesystem. Once the server determines that the client is allowed to access the mountpoint in question, it returns the filehandle for the root of the exported filesystem.

These access requests are not handled by the NFS protocol directly (although this functionality will become part of the NFS protocol in NFS PV4). Instead, a separate MOUNT protocol was designed to run on the NFS server and process these inquiries. On HP-UX systems, the MOUNT protocol is implemented via the rpc.mountd daemon.

rpc.mountd is a single-threaded process that runs in user-space, as opposed to being implemented in the kernel. That being the case, there is very little that can be done to influence the performance of this daemon directly. However, rpc.mountd performance can be affected by external factors such as hostname resolution speed.



What factors influence rpc.mountd performance?

rpc.mountd

- The choice of **repository** used to provide hostname resolution data (i.e. `/etc/hosts`, DNS, NIS, NIS+)
  - ➢ How much **overhead** is involved in getting hostname data
  - ➢ How **fast** are the directory servers
  - ➢ How **accurate** is the information in the directory
- The usage and nesting of **netgroups** for access lists
- The size of the **`/etc/rmtab`** file

July 22, 2002

Page 58

Notes:

Choice of directory service used for hostname resolution data

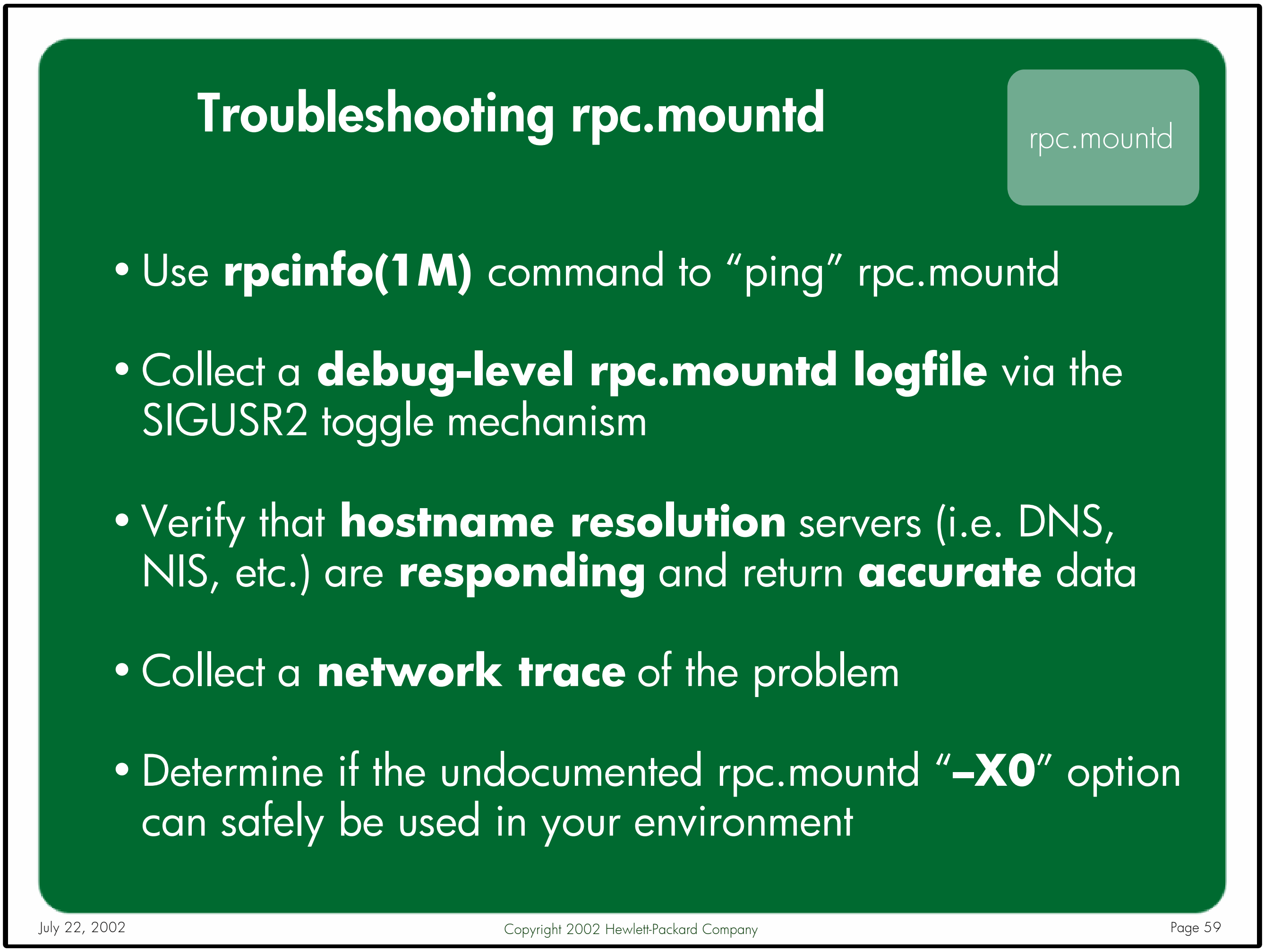
HP-UX supports many different hostname resolution repositories – DNS, NIS, NIS+, and `/etc/hosts`. Since rpc.mountd depends heavily on hostname resolution, the speed and accuracy of this data is critical to performance.

The usage and nesting of netgroups for access lists

Netgroups can take a relatively long time to traverse, especially if they are managed by a directory service such as NIS. Searching netgroups during every MOUNT request can be burdensome, particularly if the netgroups are heavily nested. If netgroups are required in your environment, flatten them if possible.

The size of the `/etc/rmtab` file

The `/etc/rmtab` file is used by rpc.mountd to keep track of the filesystems mounted by each NFS client. mountd reads this file at startup. A large rmtab file can take several minutes to load, during which time no mount requests can be processed. This file should periodically be removed during scheduled maintenance windows.



Troubleshooting rpc.mountd

rpc.mountd

- Use **rpcinfo(1M)** command to “ping” rpc.mountd
- Collect a **debug-level rpc.mountd logfile** via the SIGUSR2 toggle mechanism
- Verify that **hostname resolution** servers (i.e. DNS, NIS, etc.) are **responding** and return **accurate** data
- Collect a **network trace** of the problem
- Determine if the undocumented rpc.mountd “**–X0**” option can safely be used in your environment

July 22, 2002

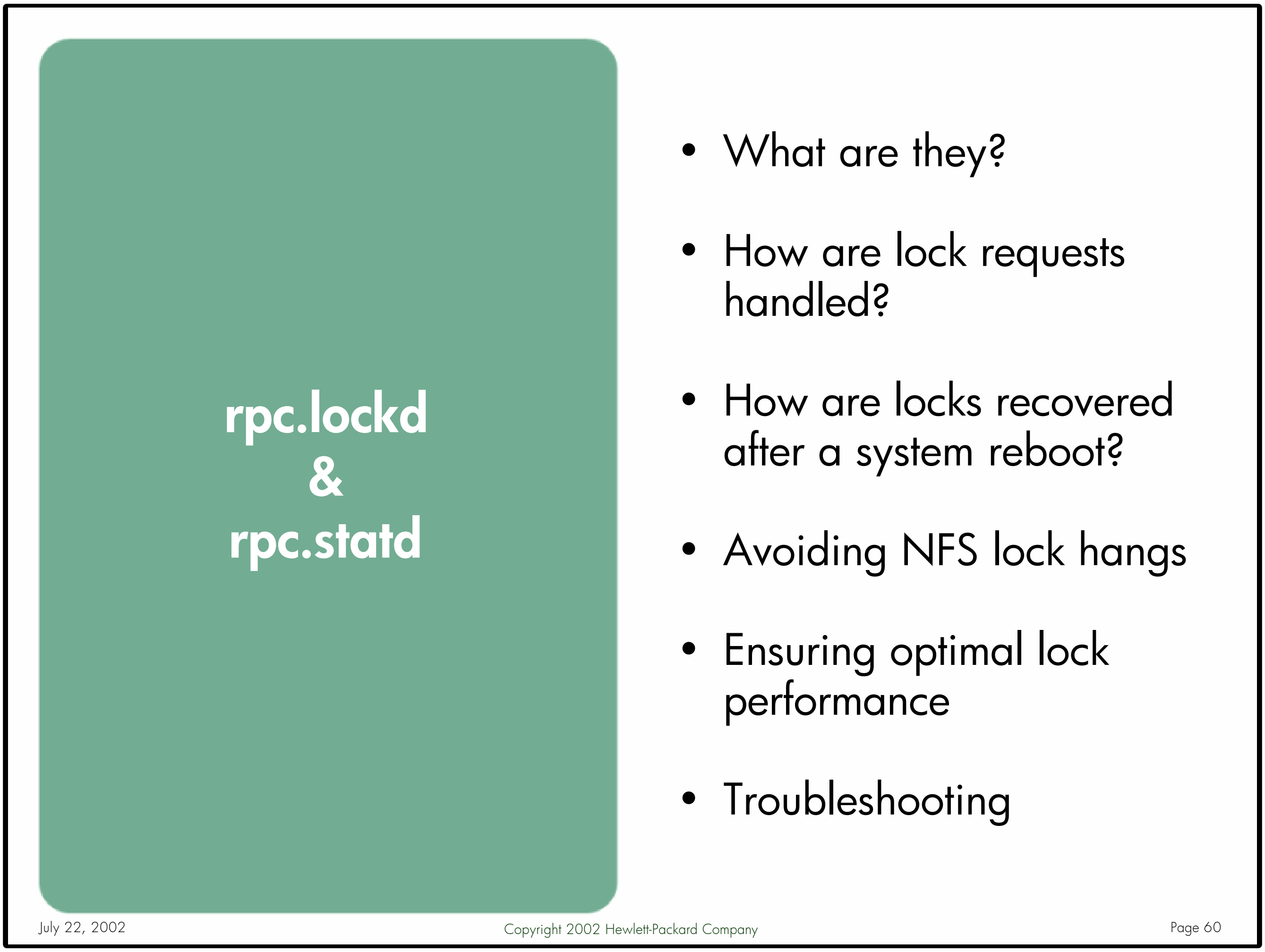
Page 59

Notes: The rpcinfo(1M) command can “ping” rpc.mountd to quickly determine if the daemon is at least responding to requests.

One of the best sources of information for troubleshooting rpc.mountd problems is a debug mountd logfile collected while reproducing the failure. This debug logging can be toggled on and off by sending the SIGUSR2 signal to the running mountd process (i.e. kill –17 <mountd pid>). By default, the debug information is appended to the `/var/adm/mountd.log` file. In some cases, a network trace is also needed to fully understand the root cause of rpc.mountd’s non-responsiveness.

A large majority of rpc.mountd issues are caused by hostname resolution problems (i.e. down DNS/NIS servers). Be sure to verify that hostname resolution is working.

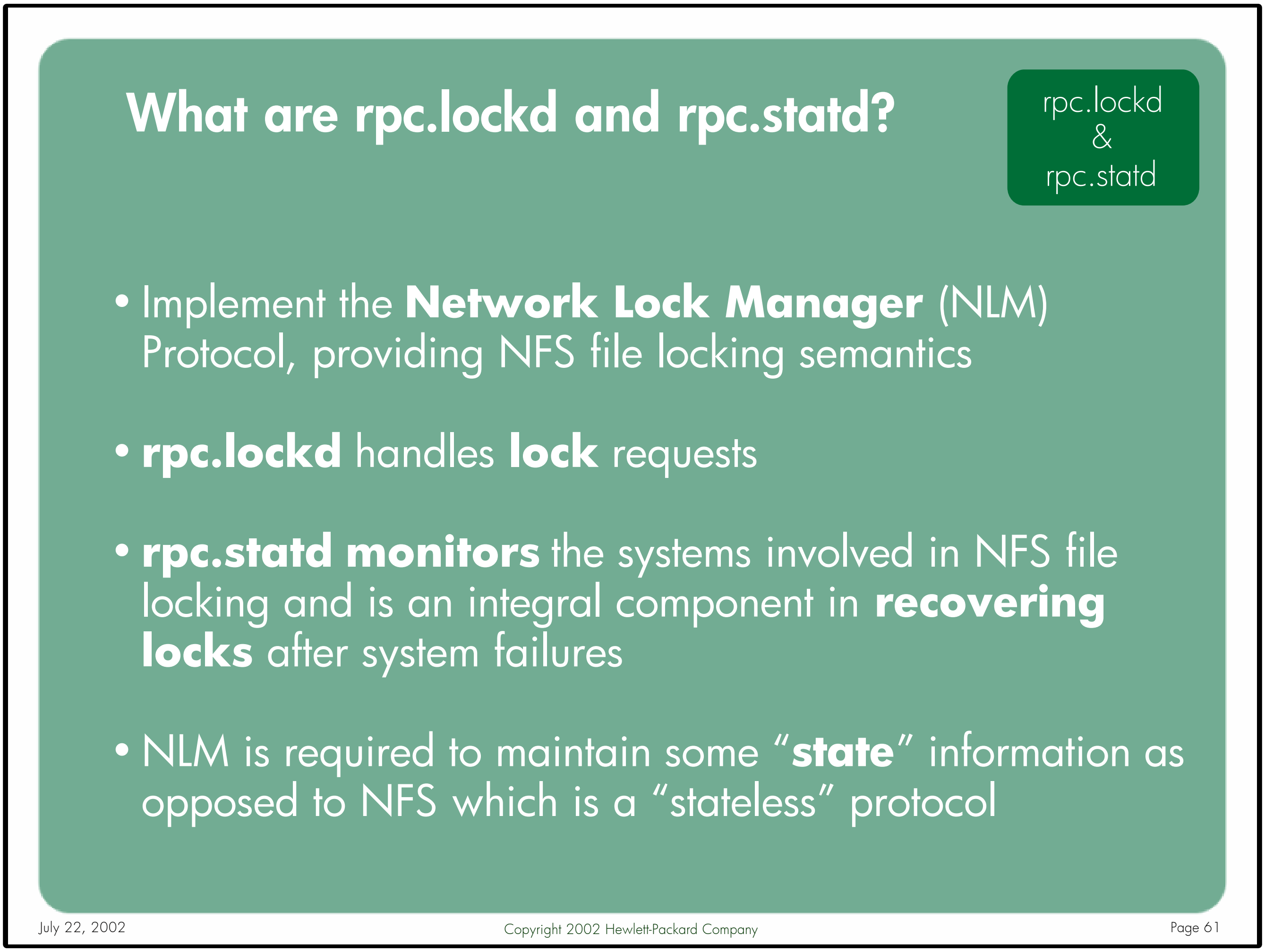
An rpc.mountd/automount deadlock situation was discovered involving a MOUNT request of a symbolic link that referenced a loopback NFS mount point managed by automounter. The fix for this extreme corner-case issue involved adding many new conditional checks to mountd’s MOUNT function. Launching rpc.mountd with the undocumented “-X0” option disables these checks.



rpc.lockd & rpc.statd

- What are they?
- How are lock requests handled?
- How are locks recovered after a system reboot?
- Avoiding NFS lock hangs
- Ensuring optimal lock performance
- Troubleshooting

Notes: This section describes how the rpc.lockd and rpc.statd daemons make up the Network Lock Manager (NLM) protocol. It discusses how lock requests are processed, and how locks are recovered after server failures. It also explains why many NFS file lock hangs occur and how to avoid them. This is followed by a discussion on the factors in your environment that can adversely affect the performance of rpc.lockd and rpc.statd. Finally, a troubleshooting section describes the recommended tools and procedures to use when investigating NFS file locking issues.



What are rpc.lockd and rpc.statd?

rpc.lockd
&
rpc.statd

- Implement the **Network Lock Manager** (NLM) Protocol, providing NFS file locking semantics
- **rpc.lockd** handles **lock** requests
- **rpc.statd monitors** the systems involved in NFS file locking and is an integral component in **recovering locks** after system failures
- NLM is required to maintain some "**state**" information as opposed to NFS which is a "stateless" protocol

July 22, 2002

Page 61

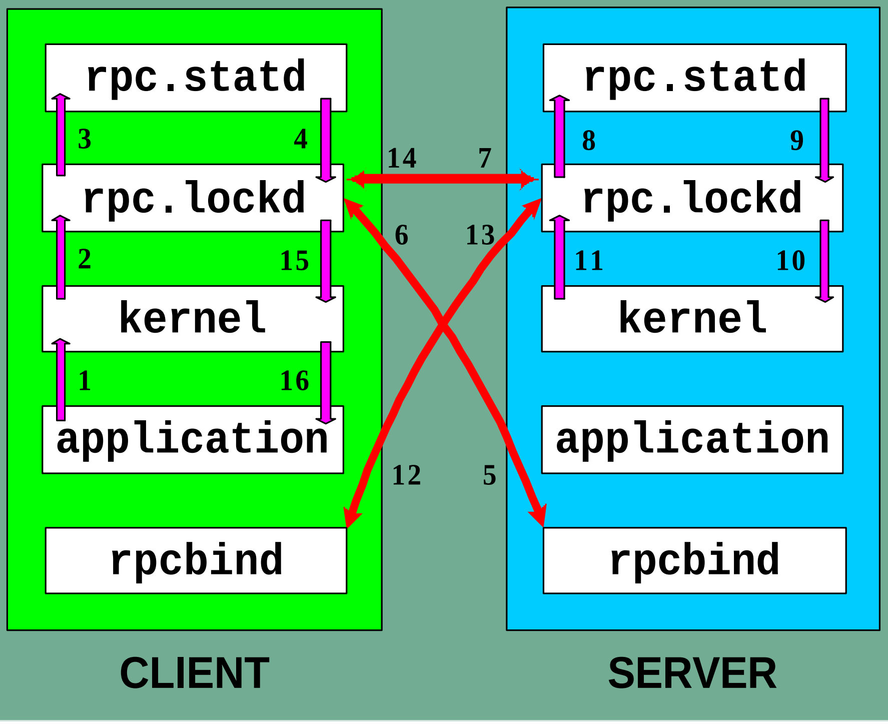
Notes: The NFS protocol is commonly described as being "stateless" because the NFS server maintains no state information about the clients. Every NFS request is treated individually by the server and no assumptions are made based on previous requests. This design works fine, except when it comes to NFS file locking.

In the file locking case, both the client and the server need to maintain a list of which files are locked, by which processes, and on which NFS clients. Some of this information must be stored on local disk in order for it to survive a client or server reboot (since anything in memory is lost during a reboot). In order for NFS to maintain this "state" information on top of a "stateless" protocol, a new mechanism was created to manage NFS file locks – the Network Lock Manager (NLM) protocol.

On HP-UX systems, the rpc.lockd and rpc.statd daemons (commonly referred to as simply "lockd" and "statd") implement the Network Lock Manager protocol.

How are NFS file lock requests handled by lockd and statd?

rpc.lockd
&
rpc.statd



July 22, 2002

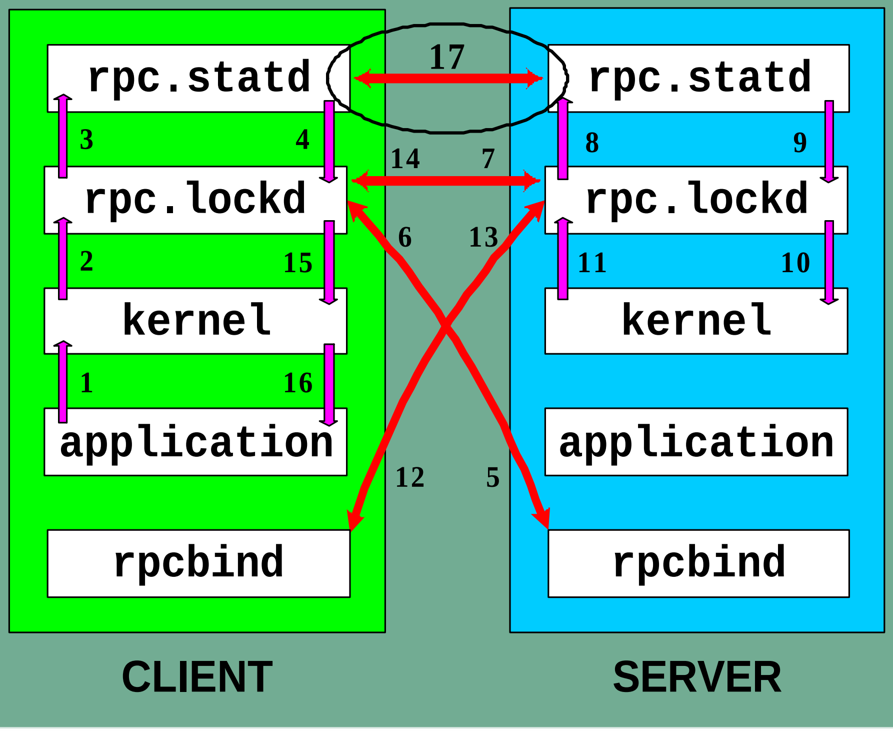
Page 62

Notes:

1. Application on the client requests to lock a file – the request goes to the kernel
2. Kernel resolves pathname to NFS rnode – forwards request to local rpc.lockd
3. Clients lockd contacts the local rpc.statd asking statd to monitor this system
4. Client's statd replies to the local lockd that it is monitoring the client system
5. Client's lockd contacts the server's rpcbind to get port number of server's lockd
6. Server's rpcbind replies with the port number of the server's lockd
7. Client's lockd sends the lock request to the server's lockd
8. Server's lockd contacts its local statd asking statd to monitor the client system
9. Server's statd replies to the local lockd that it now monitoring the client system
10. Server's lockd forwards the lock request to the server's kernel
11. Server's kernel performs the lock and replies to the server's lockd with status
12. Server's lockd contacts the client's rpcbind to get port number of client's lockd
13. Client's rpcbind replies with the port number of the client's lockd
14. Server's lockd sends the lock results back to the client's lockd
15. Client's lockd forwards these results back to the local kernel
16. Client's kernel forwards the results back to the requesting application

How are NFS file locks recovered after a client or server reboot?

rpc.lockd
&
rpc.statd



July 22, 2002

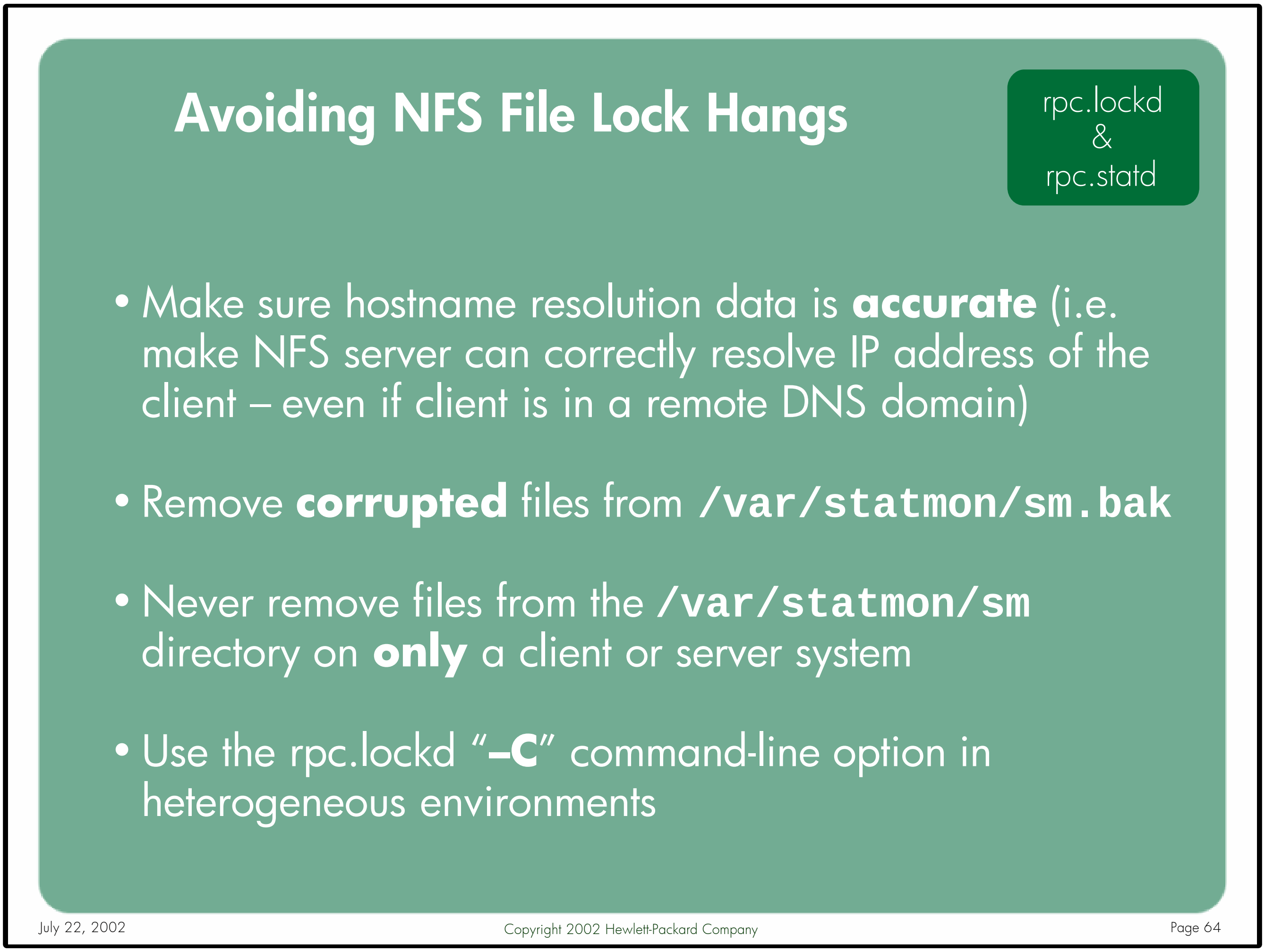
Notes:

17. Notification process takes place between client's and server's rpc.statd daemons during a "change in state"

A state change refers to any time where lockd and statd are stopped and restarted, either due to the client or server rebooting, or when the daemons are killed and restarted manually. In this situation, the statd daemon is instrumental in notifying the remote system, either client or server, of the change in state on the local system.

If the NFS client is the system experiencing the state change then its statd notifies the statd process on every NFS servers that the client was performing file locking with, letting those systems know they should discard any locks they are currently holding for the client.

If the server experiences the change in state, any locks it was holding prior to the state change are gone. The server's statd therefore needs to contact the statd process on all NFS clients who have issued file lock requests with the server, informing them that any locks they were holding with the server are gone and must be reclaimed within 50 seconds or the server could give the locks to another client.



Avoiding NFS File Lock Hangs

rpc.lockd
&
rpc.statd

- Make sure hostname resolution data is **accurate** (i.e. make NFS server can correctly resolve IP address of the client – even if client is in a remote DNS domain)
- Remove **corrupted** files from **`/var/statmon/sm.bak`**
- Never remove files from the **`/var/statmon/sm`** directory on **only** a client or server system
- Use the rpc.lockd "**–C**" command-line option in heterogeneous environments

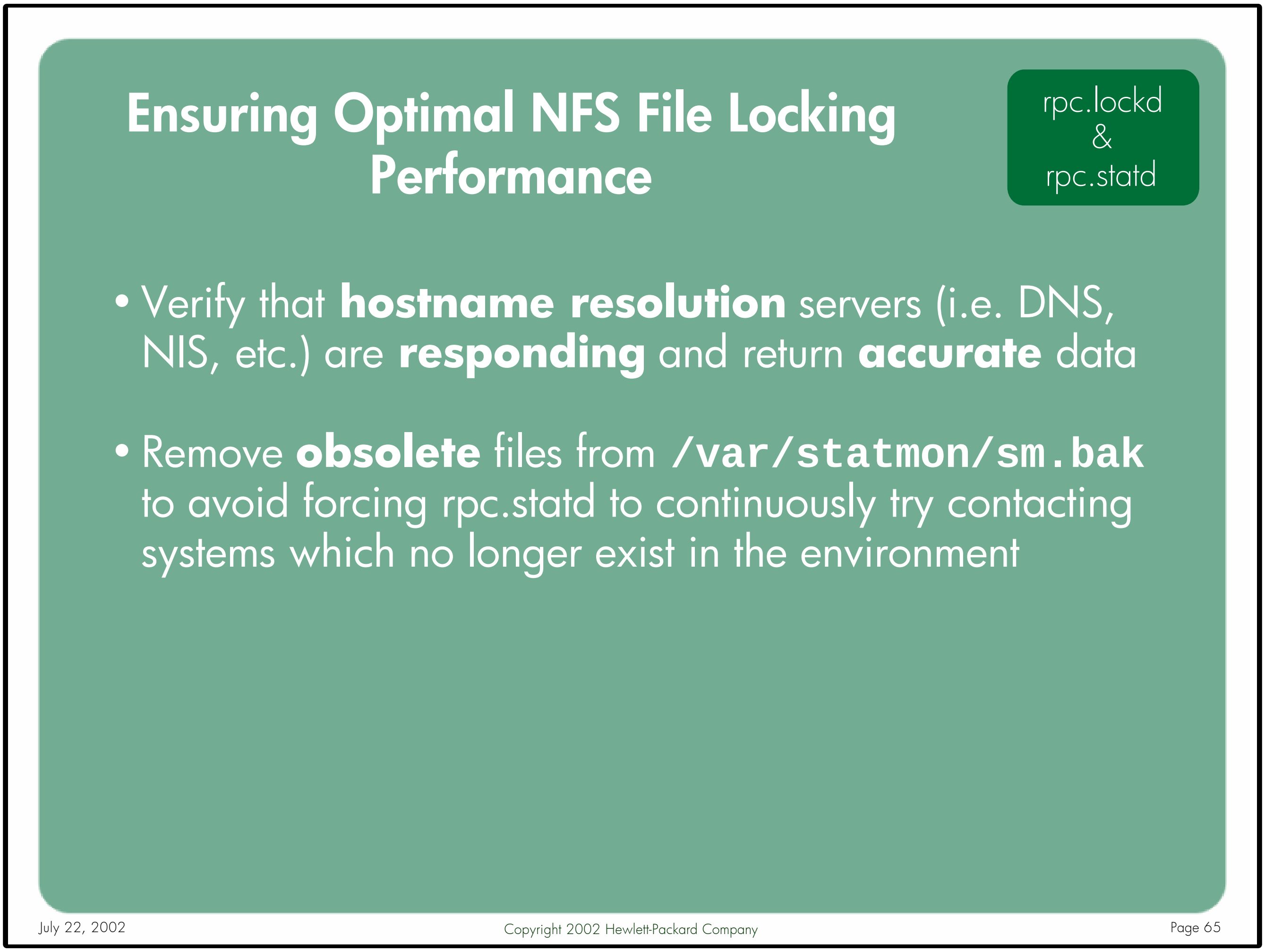
July 22, 2002

Page 64

Notes: It is critical that hostname/IP address data be accurate. In the DNS case, the potential exists for lock requests to fail when the NFS client and server reside in different DNS domains – even if the DNS tables are accurate. This is because the NFS client only sends it's hostname in lock requests. Either configure search paths for DNS or set the client's hostname equal to the fully-qualified DNS domain name.

The files in `/var/statmon/sm.bak` refer to systems that rpc.statd could not contact after a state change. If these files get corrupted (i.e. the contents of the files don't match the name of the file) then they should be removed during the next maintenance window. The files in `/var/stamon/sm` refer to systems that rpc.lockd has performed file locking with since the last state change. These files should never be removed on only a client or a server. If they must be removed, they should be removed from both systems simultaneously.

A defect in HP's NLM implementation causes locks to hang (in a specific case) when working with non-HP servers. The rpc.lockd "–C" option corrects this, but this option must be enabled on all HP systems in the environment simultaneously or the HP systems could potentially stop working correctly with each other.



Ensuring Optimal NFS File Locking Performance

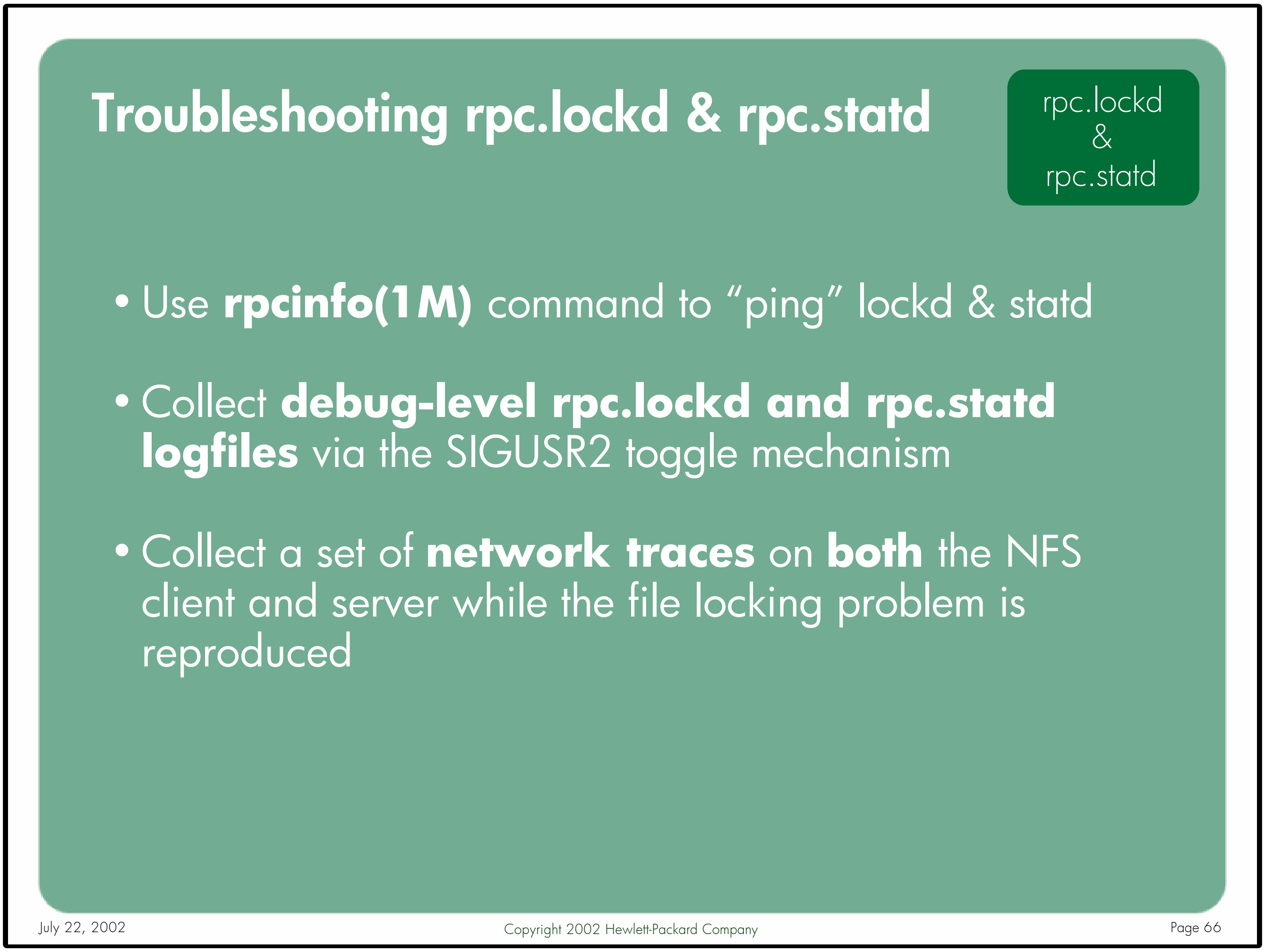
rpc.lockd
&
rpc.statd

- Verify that **hostname resolution** servers (i.e. DNS, NIS, etc.) are **responding** and return **accurate** data
- Remove **obsolete** files from **`/var/statmon/sm.bak`** to avoid forcing rpc.statd to continuously try contacting systems which no longer exist in the environment

July 22, 2002 Copyright 2002 Hewlett-Packard Company Page 65

Notes: Even when NFS file locking is functionally working, hostname resolution still plays a key role in rpc.lockd and rpc.statd performance. There are several places in the lockd and statd source code where the gethostbyname() routine is called to retrieve the IP address of the remote system. Therefore, if hostname resolution performance is poor then NFS file locking performance will suffer as a result. Again, this could involve DNS, NIS, NIS+, or the `/etc/hosts` file – depending upon how hostname resolution is configured in the `/etc/nsswitch.conf` file.

During NFS lock recovery, if statd is unable to notify a system associated with a `/var/statmon/sm.bak` file then this file remains in the "sm.bak" directory. Statd will continuously attempt to notify this remote host every 15 seconds. If statd is busy trying to contact these non-existent or dead clients it will either be unable to handle new monitor requests for legitimate NFS clients, or it will take a long time to process these requests. In either case, file lock performance can suffer. It is therefore important to periodically monitor the contents of the "sm.bak" directory. If files associated with obsolete clients are found then they should be removed during the next scheduled maintenance window.



Troubleshooting rpc.lockd & rpc.statd

rpc.lockd
&
rpc.statd

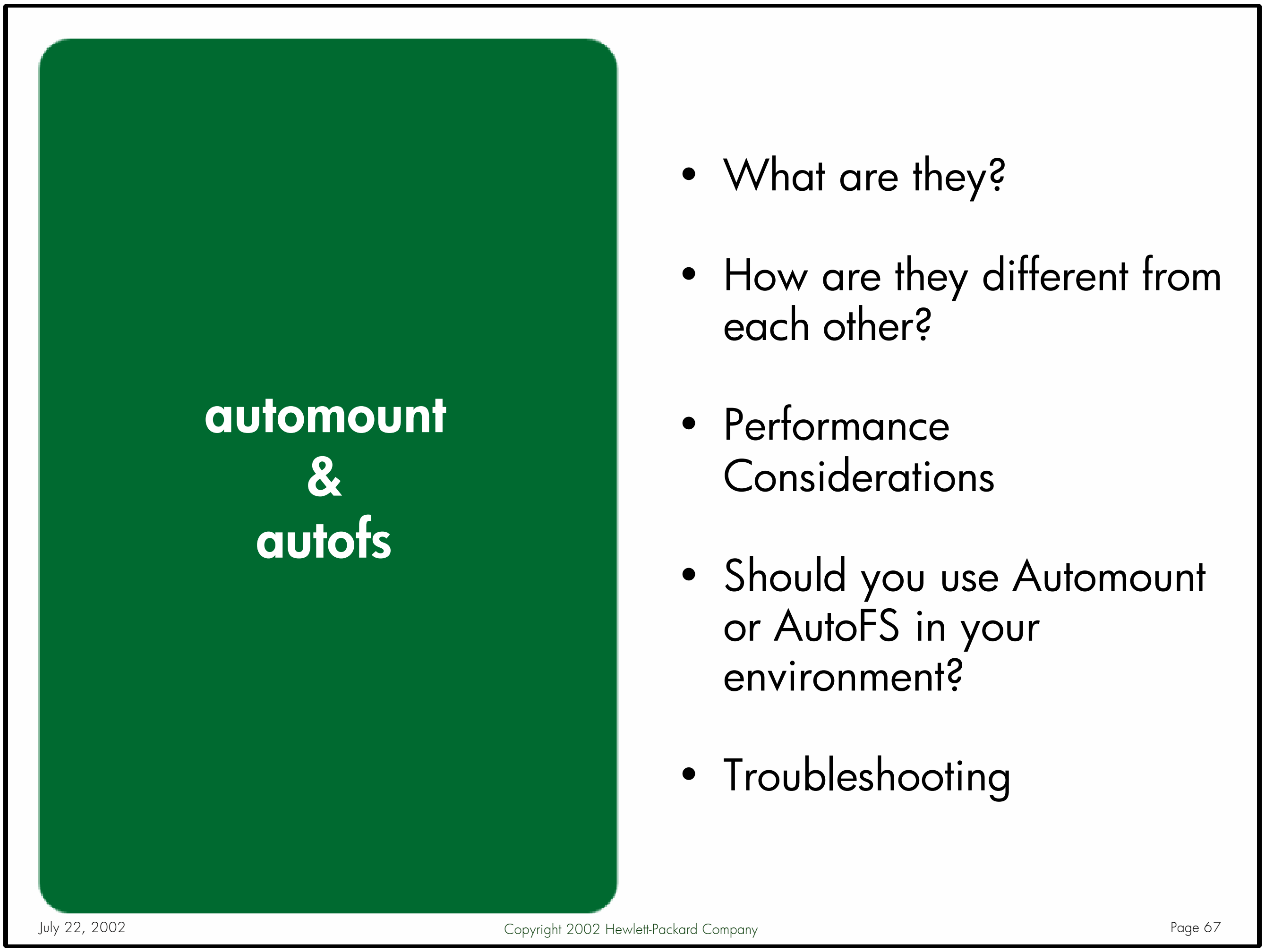
- Use **rpcinfo(1M)** command to “ping” lockd & statd
- Collect **debug-level rpc.lockd and rpc.statd logfiles** via the SIGUSR2 toggle mechanism
- Collect a set of **network traces** on **both** the NFS client and server while the file locking problem is reproduced

July 22, 2002 Copyright 2002 Hewlett-Packard Company Page 66

Notes: One of the quickest and most non-intrusive methods of determining whether the lockd and statd processes are responding is to use the rpcinfo(1M) command to “ping” the running daemons. Since both client’s and server’s lockd and statd daemons are used during NFS file locking, the recommendation is to test the server’s lockd and statd from the client system and vice versa.

Among the best sources of information for troubleshooting NFS file locking problems are debug lockd and statd logfiles collected while reproducing the failure. Debug logging can be toggled on and off by sending the SIGUSR2 signal to the running lockd or statd process (i.e. kill –17 <pid>). By default, the debug information is logged to the `/var/adm/rpc.lockd.log` and `/var/adm/rpc.statd.log` files respectively.

In some cases, a network trace is also needed to fully understand the root cause of an NFS file locking problem. Since the problem could be that the server is not sending replies at all or is sending the replies to the wrong system/port number, it is important to collect network traces on both the client and server to determine whether the lock requests and replies are traversing the network successfully.



automount
&
autofs

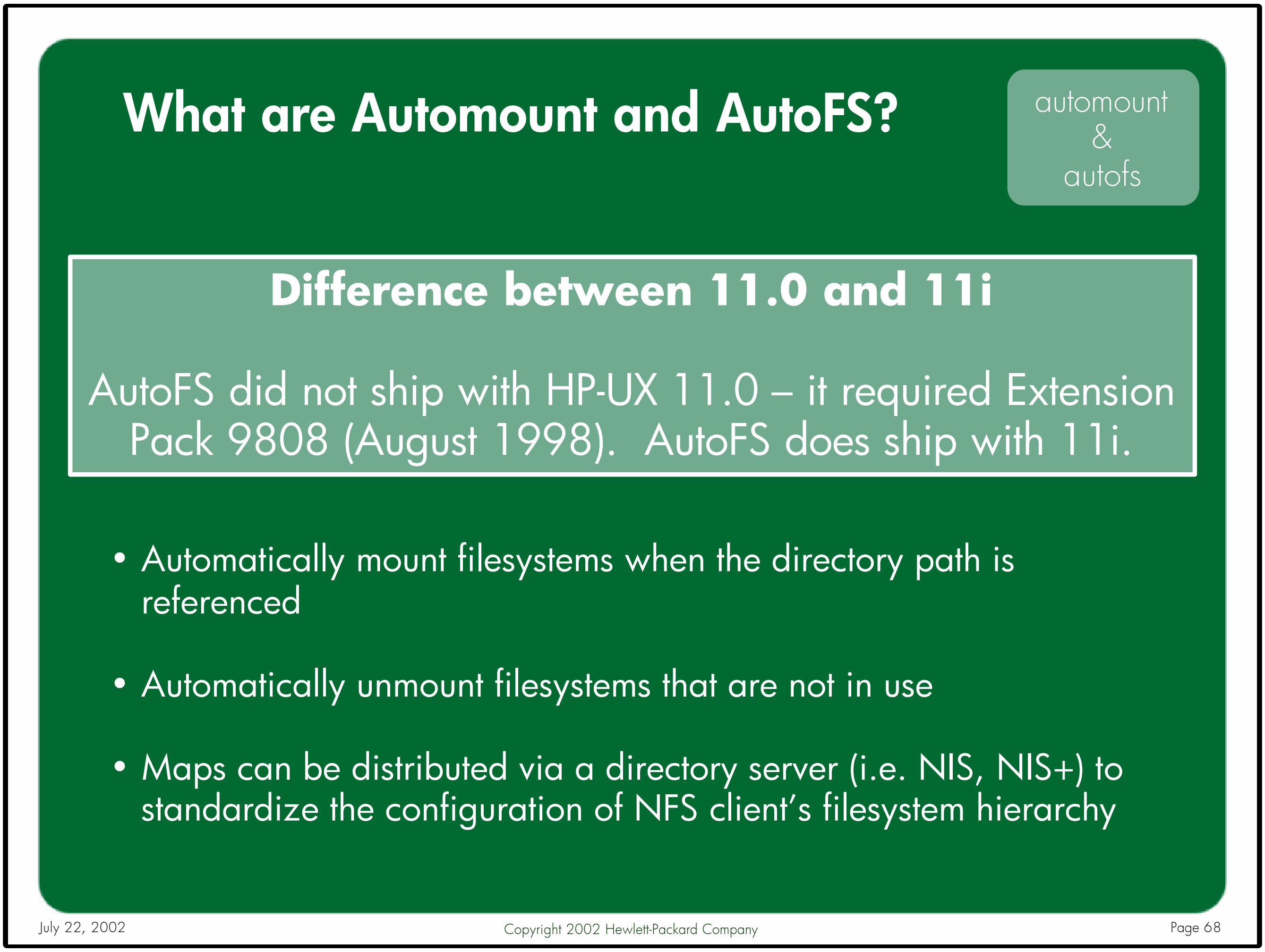
- What are they?
- How are they different from each other?
- Performance Considerations
- Should you use Automount or AutoFS in your environment?
- Troubleshooting

July 22, 2002

Page 67

Notes: Automount and AutoFS are generally not considered daemons used for performance reasons. In fact, in performance sensitive environments, the use of any automounter should be carefully scrutinized as it will typically add some amount of overhead to the NFS client. The decision whether or not to use these products is typically made by system administrators who are familiar with the client environment. For those administrators who do choose to use one of these products, there are several performance and behavioral issues to consider.

This section describes how the classic automounter differs from the newer AutoFS product in terms of functionality and performance. Also described are the many configuration-related factors that can influence automounter performance. A discussion on whether the classic automounter or newer AutoFS is a better choice for your environment is included. Finally, a troubleshooting section describes the methodology used to investigate most automount and AutoFS related problems.



What are Automount and AutoFS?

automount
&
autofs

Difference between 11.0 and 11i

AutoFS did not ship with HP-UX 11.0 – it required Extension Pack 9808 (August 1998). AutoFS does ship with 11i.

- Automatically mount filesystems when the directory path is referenced
- Automatically unmount filesystems that are not in use
- Maps can be distributed via a directory server (i.e. NIS, NIS+) to standardize the configuration of NFS client's filesystem hierarchy

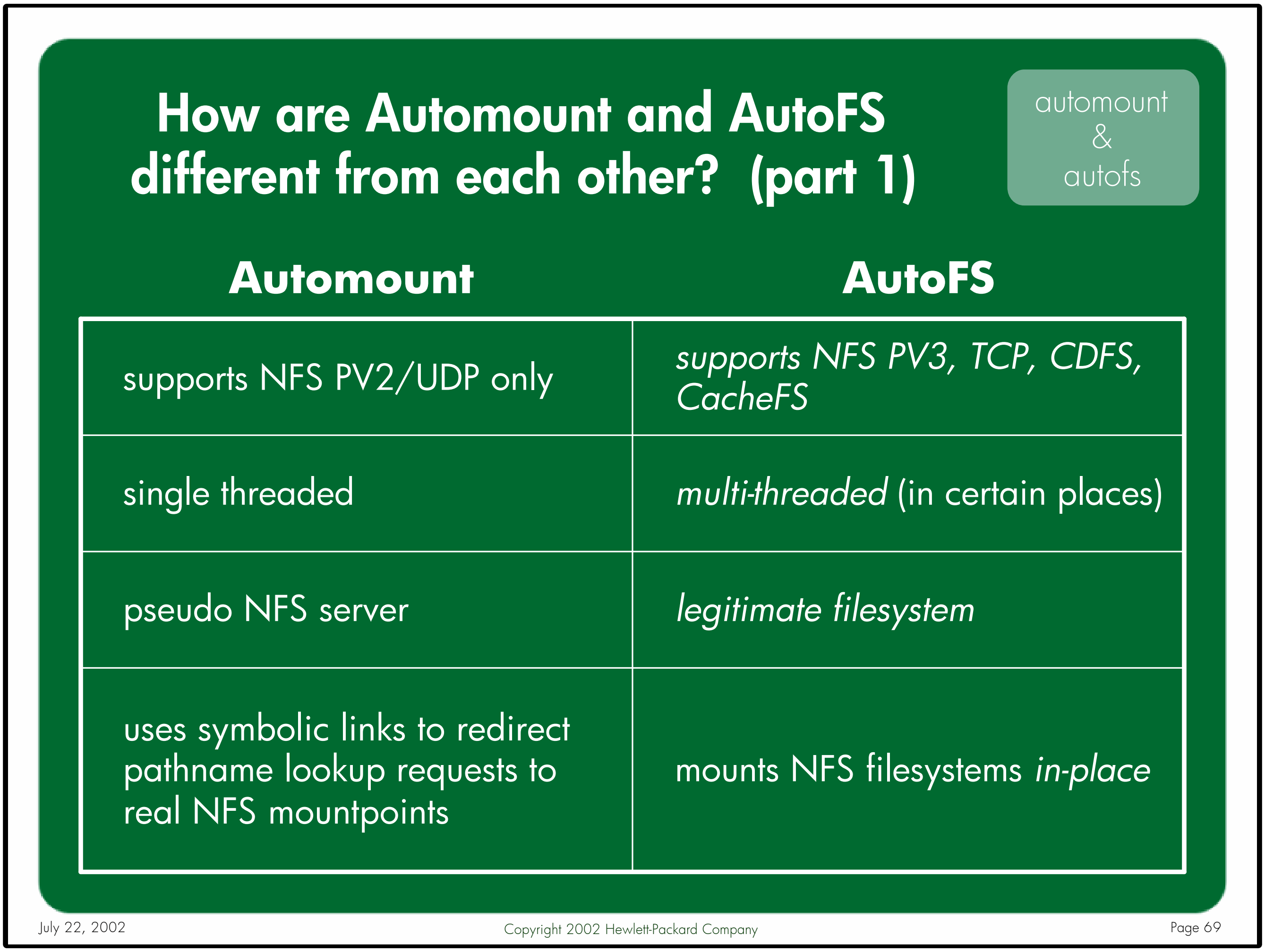
July 22, 2002 Copyright 2002 Hewlett-Packard Company Page 68

Notes: An automounter is used to automatically mount and unmount filesystems on an NFS client. The filesystems are mounted when the configured directory path is referenced, and they are unmounted after a configurable idle period.

The maps used by automount and AutoFS can be maintained and distributed via directory services such as NIS or NIS+. This helps ease the administrative burden associated with standardizing the configuration of the filesystem layout on large numbers of NFS client systems.

HP-UX has supported automatic mounting of filesystems for years using the original automount daemon. AutoFS was introduced on HP-UX 10.20 in April 1997 as part of the ACE 2.0 networking bundle. It was subsequently introduced to the 11.0 release on Extension Pack 9808, which shipped in August 1998.

AutoFS ships with HP-UX 11i.



How are Automount and AutoFS different from each other? (part 1)

automount
&
autofs

Automount	AutoFS
supports NFS PV2/UDP only	*supports NFS PV3, TCP, CDFS, CacheFS*
single threaded	*multi-threaded* (in certain places)
pseudo NFS server	*legitimate filesystem*
uses symbolic links to redirect pathname lookup requests to real NFS mountpoints	mounts NFS filesystems *in-place*

July 22, 2002 Copyright 2002 Hewlett-Packard Company Page 69

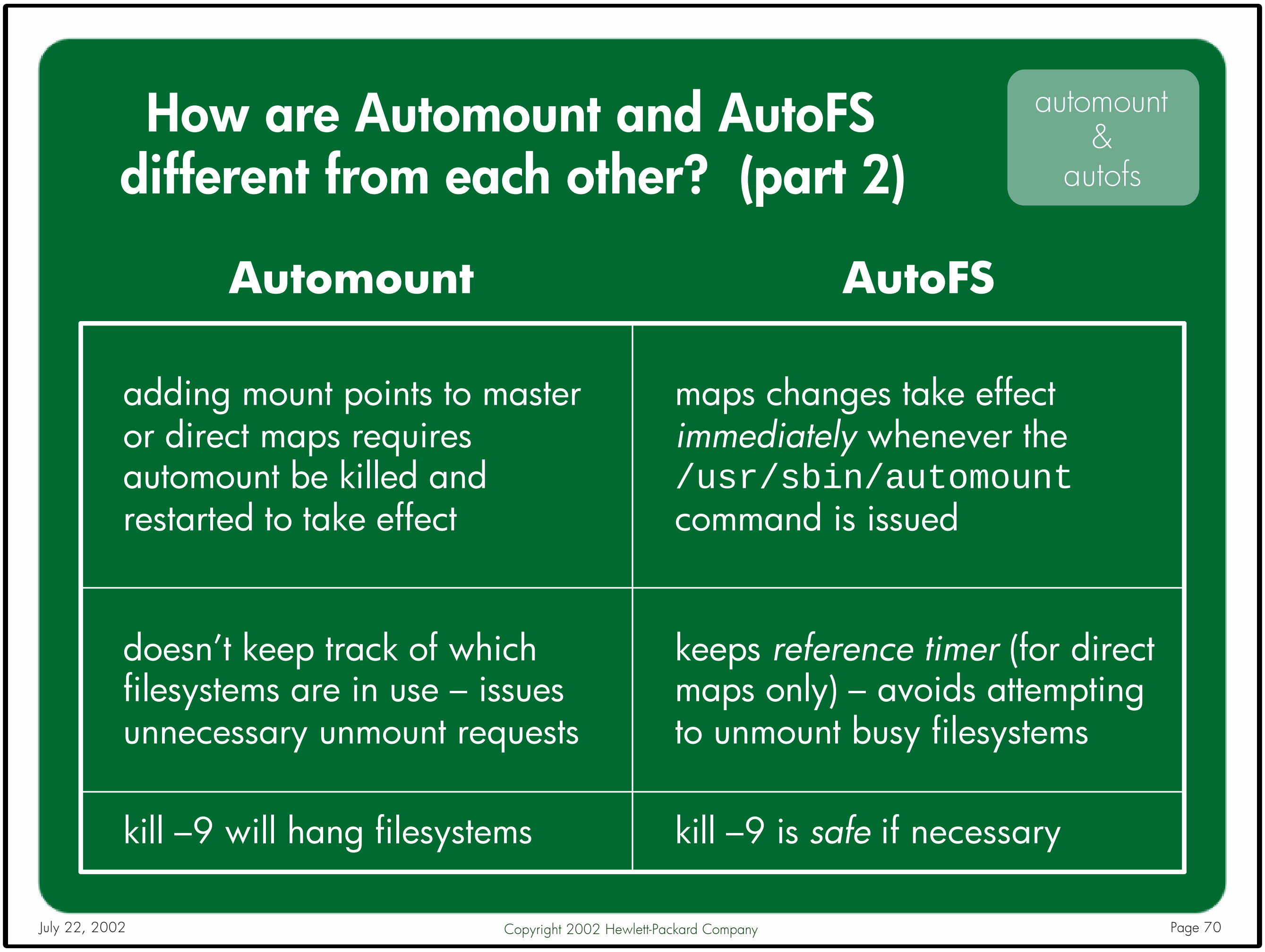
Notes: The original automounter is only capable of managing NFS protocol version 2 mounts using the UDP transport. AutoFS can manage NFS PV2 or PV3 filesystems running on top of either UDP/IP or TCP/IP. Additionally, AutoFS can be configured to manage CacheFS filesystems.

Automount is a single-threaded process that performs its functions of mounting and unmounting filesystems by emulating an NFS server running on the client system. This explains why when automount hangs or is slow to respond the client will print a message similar to one where a remote NFS server stops responding:

```
NFS server (pid570@/net) not responding still trying
```

Automount uses symbolic links to redirect pathname requests to the real NFS mountpoints which it mounts under a holding directory (`/tmp_mnt` by default).

AutoFS is a multi-threaded daemon which implements a legitimate file system, just like HFS, VxFS, CDFS, etc. It mounts the filesystems it manages directly to the requested pathname, avoiding the need for symbolic links.



How are Automount and AutoFS different from each other? (part 2)

automount & autofs

Automount	AutoFS
adding mount points to master or direct maps requires automount be killed and restarted to take effect	maps changes take effect *immediately* whenever the `/usr/sbin/automount` command is issued
doesn't keep track of which filesystems are in use – issues unnecessary unmount requests	keeps *reference timer* (for direct maps only) – avoids attempting to unmount busy filesystems
kill –9 will hang filesystems	kill –9 is *safe* if necessary

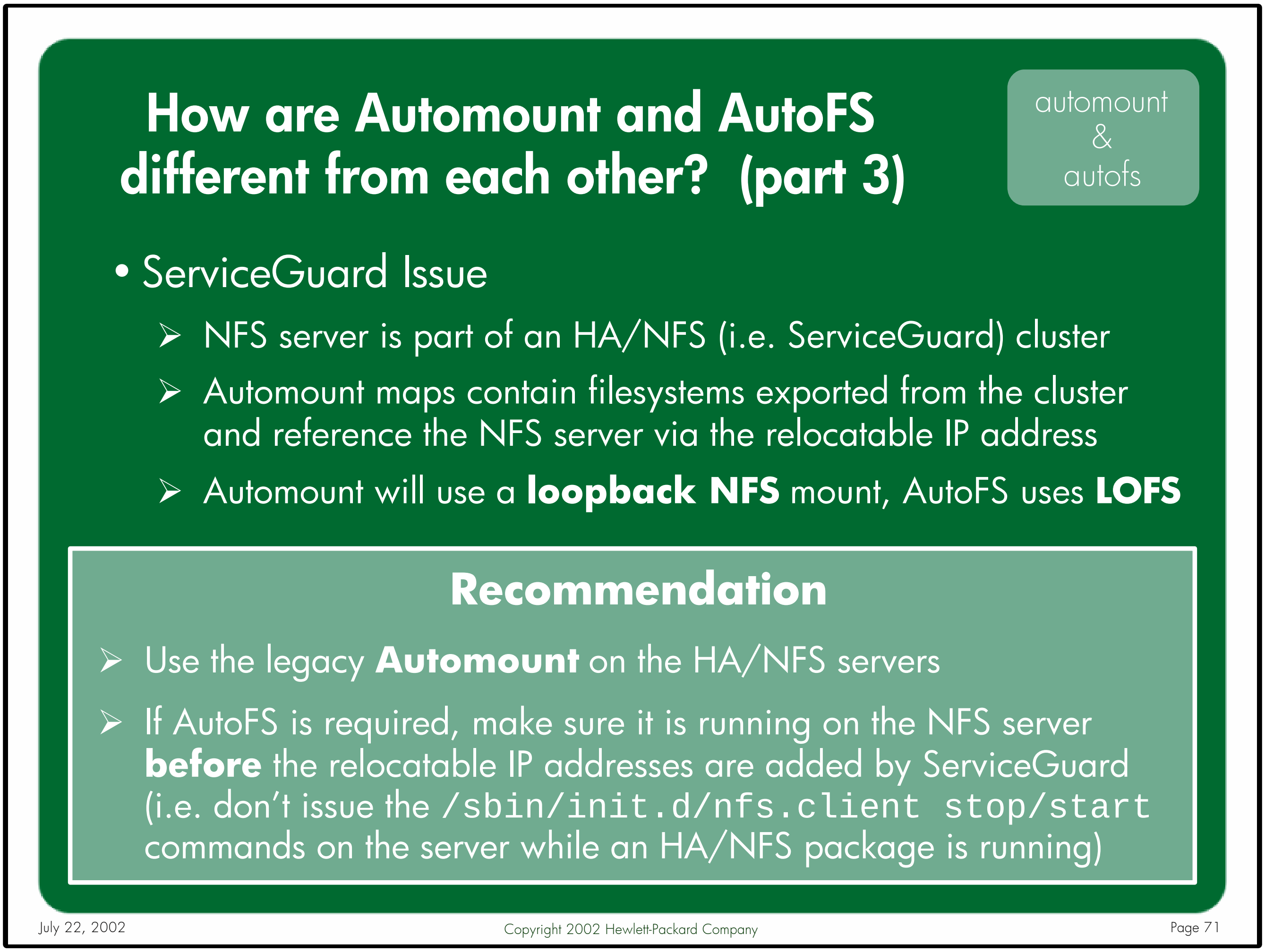
July 22, 2002

Page 70

Notes: Any changes made to the master or direct maps requires automount to be stopped and restarted for the changes to take effect. AutoFS re-processes its maps whenever `/usr/sbin/automount` is issued. Changes take effect immediately.

Automount attempts to unmount filesystems at the specified interval regardless of whether they are in use or not. AutoFS has the ability to track when filesystems were last referenced and only attempts to unmount those filesystems that have been idle for the specified interval. Unfortunately, HP's current implementation of AutoFS only supports this feature for filesystems configured in direct maps. Indirect and hierarchical map entries behave like the old automounter.

Since automount is a user-space daemon that puts placeholders in the VFS layer, if you kill the automount process with a –9 (SIGKILL) it will forcibly exit without first removing the VFS placeholders. Now any access to an automount-managed directory will hang, as the kernel views these entries as belonging to a dead NFS server. The portions of AutoFS that manage the VFS layer entries are in the kernel, which makes it "safe" (although not recommended) to stop the user-space automountd daemon with a SIGKILL if necessary.



How are Automount and AutoFS different from each other? (part 3)

automount
&
autofs

- ServiceGuard Issue
  - ➢ NFS server is part of an HA/NFS (i.e. ServiceGuard) cluster
  - ➢ Automount maps contain filesystems exported from the cluster and reference the NFS server via the relocatable IP address
  - ➢ Automount will use a **loopback NFS** mount, AutoFS uses **LOFS**

Recommendation

- ➢ Use the legacy **Automount** on the HA/NFS servers
- ➢ If AutoFS is required, make sure it is running on the NFS server **before** the relocatable IP addresses are added by ServiceGuard (i.e. don't issue the `/sbin/init.d/nfs.client stop/start` commands on the server while an HA/NFS package is running)

July 22, 2002

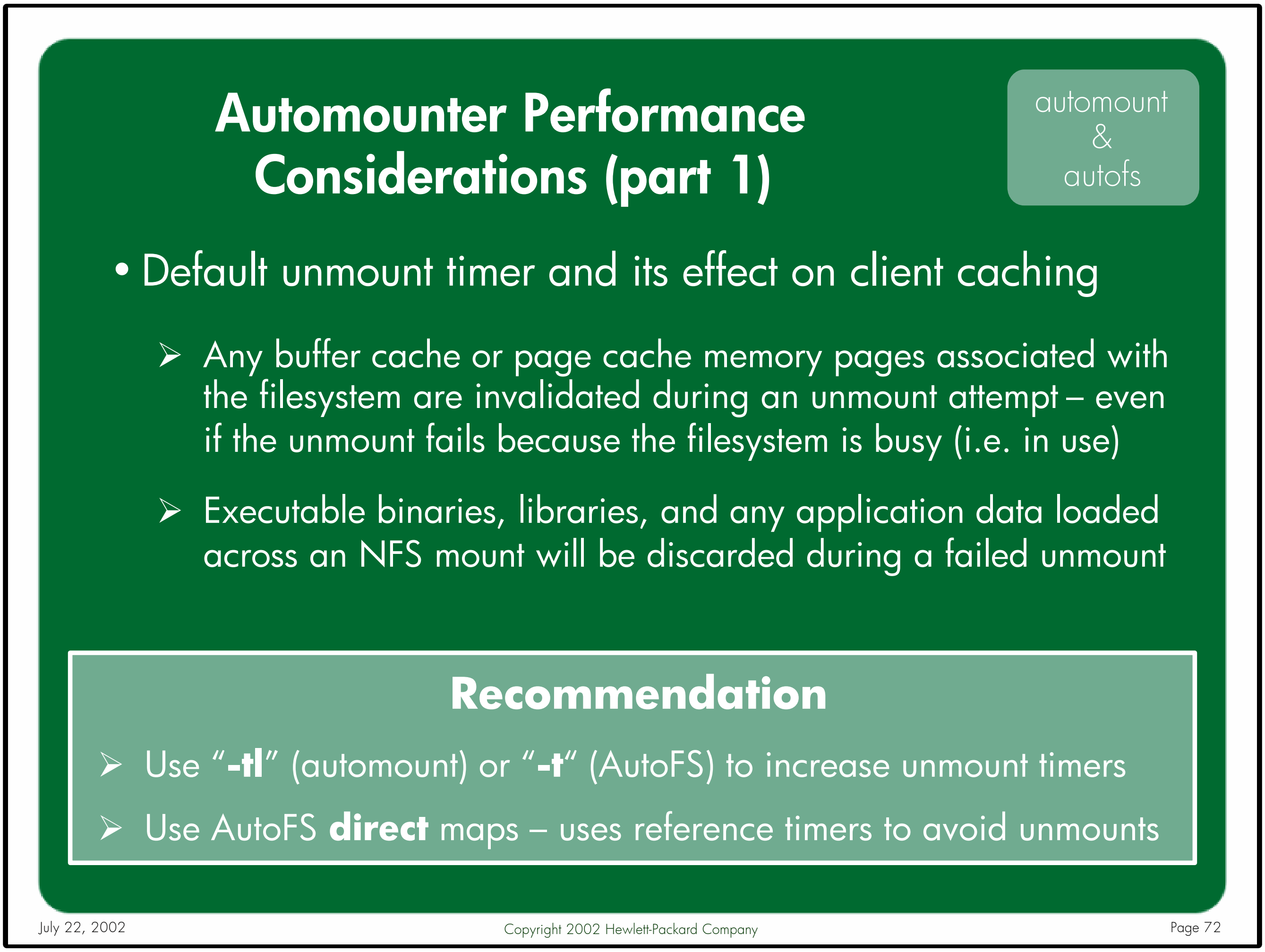
Page 71

Notes: In the above scenario, which is a common configuration for HA/NFS servers, the legacy automounter has no way of detecting that the relocatable IP address is actually a local interface and so it creates a loopback NFS mount.

When AutoFS is started it builds a list of all configured interfaces, including virtual addresses. IP addresses added to the system after AutoFS starts are not added to the list. When processing any mount request, AutoFS compares the IP address of the server with the local list to determine if the server in the map entry is really itself. If the IP address of the server in the map entry matches one of the local interfaces AutoFS will create the mountpoint using an LOFS mount, avoiding NFS entirely.

What happens if an LOFS mount is created and then the NFS package needs to switch to an adoptive node? With the current HA/NFS design, the LOFS filesystem is not unmounted and the NFS package could fail to migrate successfully.

Until this issue is resolved, the recommendation is to use the legacy automounter. In environments where AutoFS is required (i.e. for NFS PV3 support) make sure AutoFS is running on the server when the relocatable IP addresses are added.



Automounter Performance Considerations (part 1)

automount
&
autofs

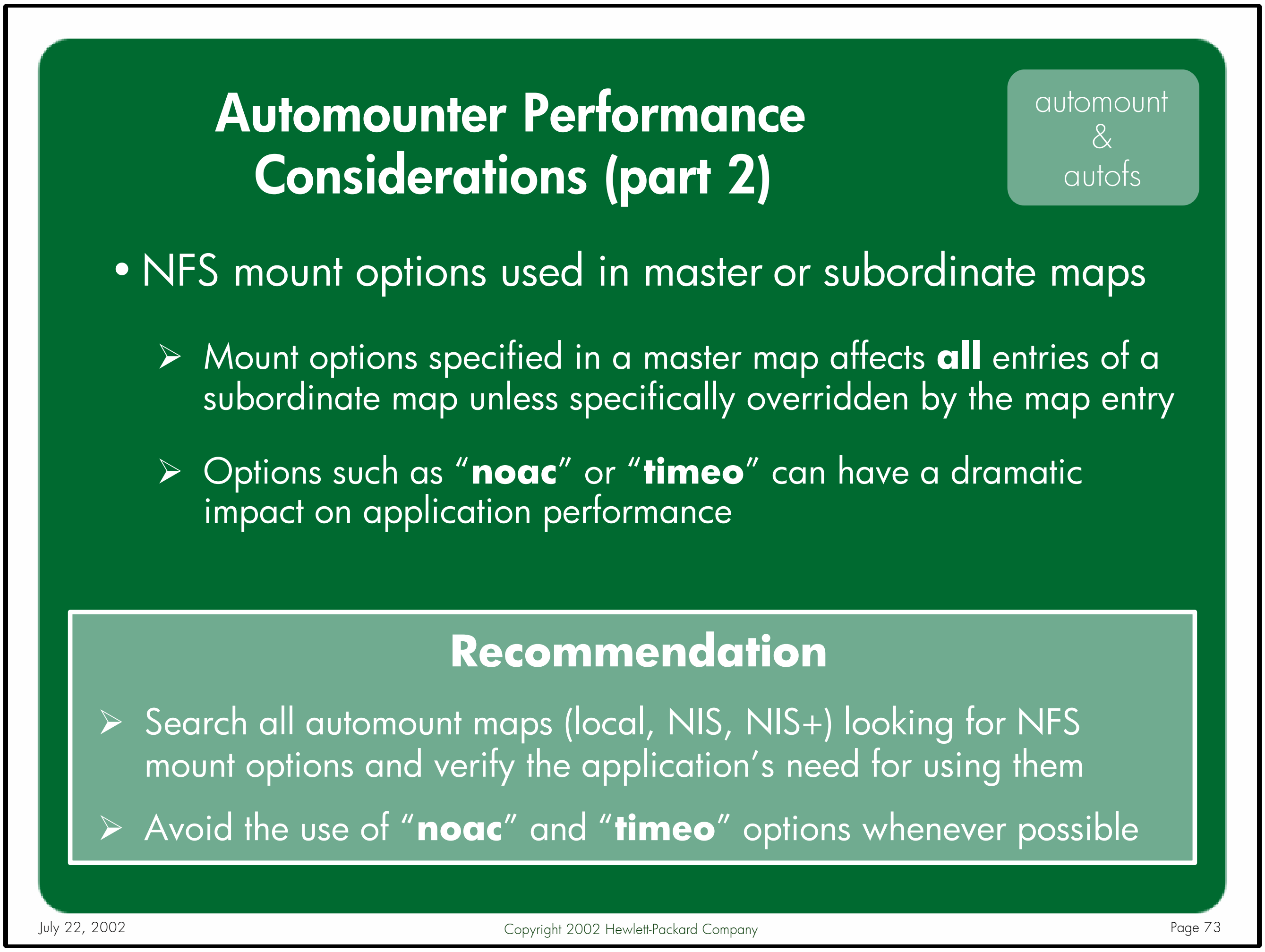
- Default unmount timer and its effect on client caching
  - ➢ Any buffer cache or page cache memory pages associated with the filesystem are invalidated during an unmount attempt – even if the unmount fails because the filesystem is busy (i.e. in use)
  - ➢ Executable binaries, libraries, and any application data loaded across an NFS mount will be discarded during a failed unmount

Recommendation

- ➢ Use "**-tl**" (automount) or "**-t**" (AutoFS) to increase unmount timers
- ➢ Use AutoFS **direct** maps – uses reference timers to avoid unmounts

Notes: A very important reason to consider using a longer automount unmount timeout value is to avoid a nasty side-effect that occurs when unmounting an NFS filesystem on HP-UX systems – all buffer cache and page cache memory pages associated with the filesystem being unmounted are invalidated. This occurs whether the filesystem is in use or not, regardless of whether the unmount attempt is successful or not. This means that both executable binaries and libraries that have been loaded across an NFS mount (page cache), as well as any application data (buffer cache) for an NFS mount will be discarded during an unmount attempt – even if the unmount fails. The NFS client is then required to re-read these pages from the server.

The command-line options used to set the unmount timer value differs between automount and AutoFS – automount uses "-tl" and AutoFS uses the "-t" option. Both options specify the number of seconds between unmount attempts. The appropriate number for this timeout value is largely dependent upon the client's usage patterns for the filesystems it mounts. In some environments a reasonable starting value would be 8 hours, while in other situations 10 minutes would make more sense.



Automounter Performance Considerations (part 2)

automount & autofs

- NFS mount options used in master or subordinate maps
  - ➢ Mount options specified in a master map affects **all** entries of a subordinate map unless specifically overridden by the map entry
  - ➢ Options such as “**noac**” or “**timeo**” can have a dramatic impact on application performance

Recommendation

➢ Search all automount maps (local, NIS, NIS+) looking for NFS mount options and verify the application’s need for using them

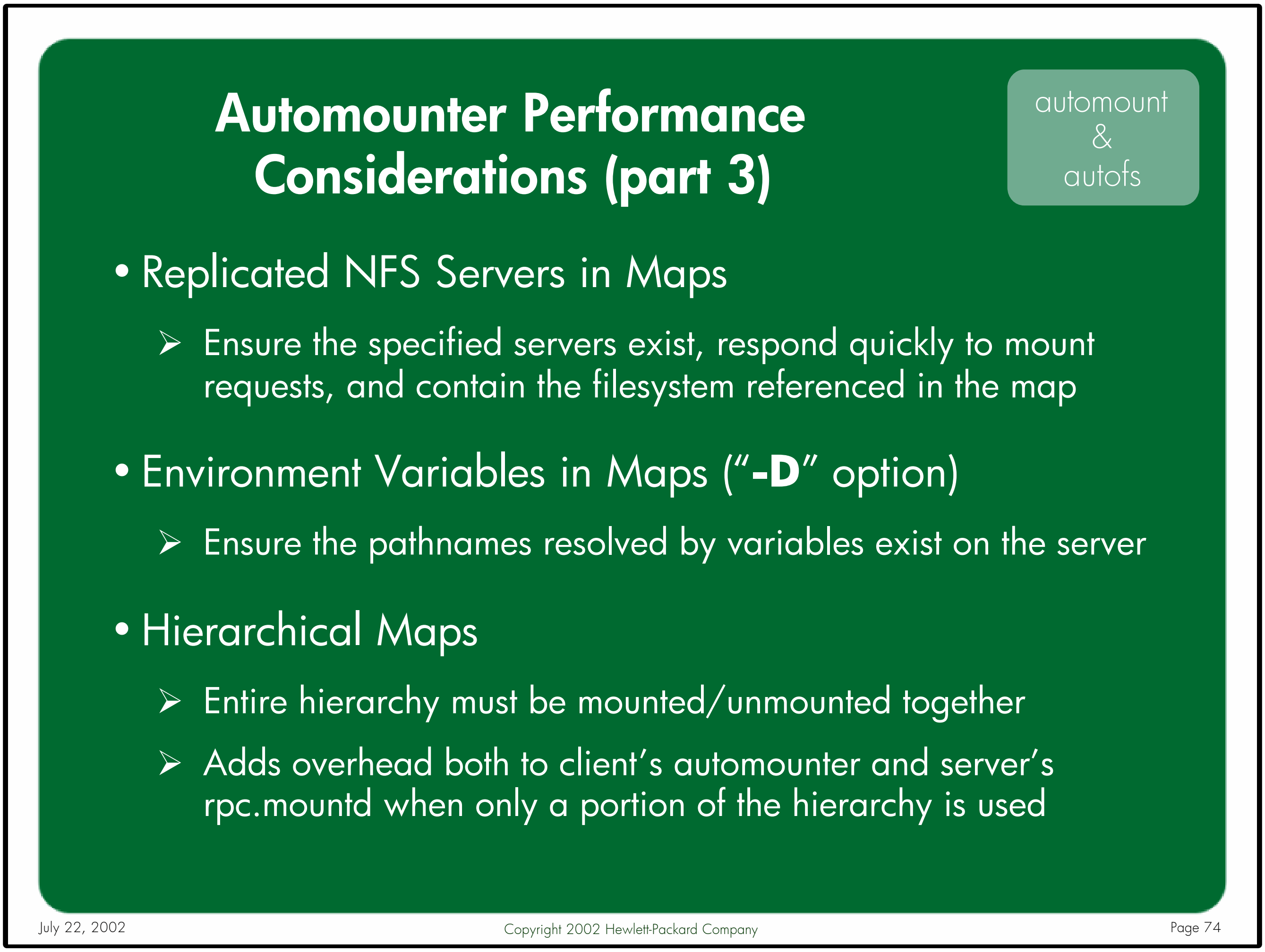
➢ Avoid the use of “**noac**” and “**timeo**” options whenever possible

July 22, 2002 Page 73

Notes: Both automount and AutoFS allow NFS mount options to be specified in the map entries. If these options are included in a filesystem-specific entry within an indirect or direct map then these options affect only that specific mount point. If the mount options are specified in a master map entry then these options affect every mount point within the subordinate direct or indirect map. While modifying the master map in this manner provides a convenient method of forcing all entries within a direct/indirect map to use the same mount options, some care should be taken to ensure that the options specified in the master map are really a ppropriate for every underlying NFS mount it affects.

For example, the “–noac” mount option is used to disable client-side attribute caching for an NFS mount. While this option might be necessary to use on a specific mount point, it is generally not a good idea to disable attribute caching for all NFS mounts on a client. In this case, the specific entry within a direct/indirect map for the mount point that requires the “–noac” option should be modified.

Similarly, the “-timeo” option should be avoided whenever possible. This mount option, and the reasons to avoid using it, are described in detail on page 98.



Automounter Performance Considerations (part 3)

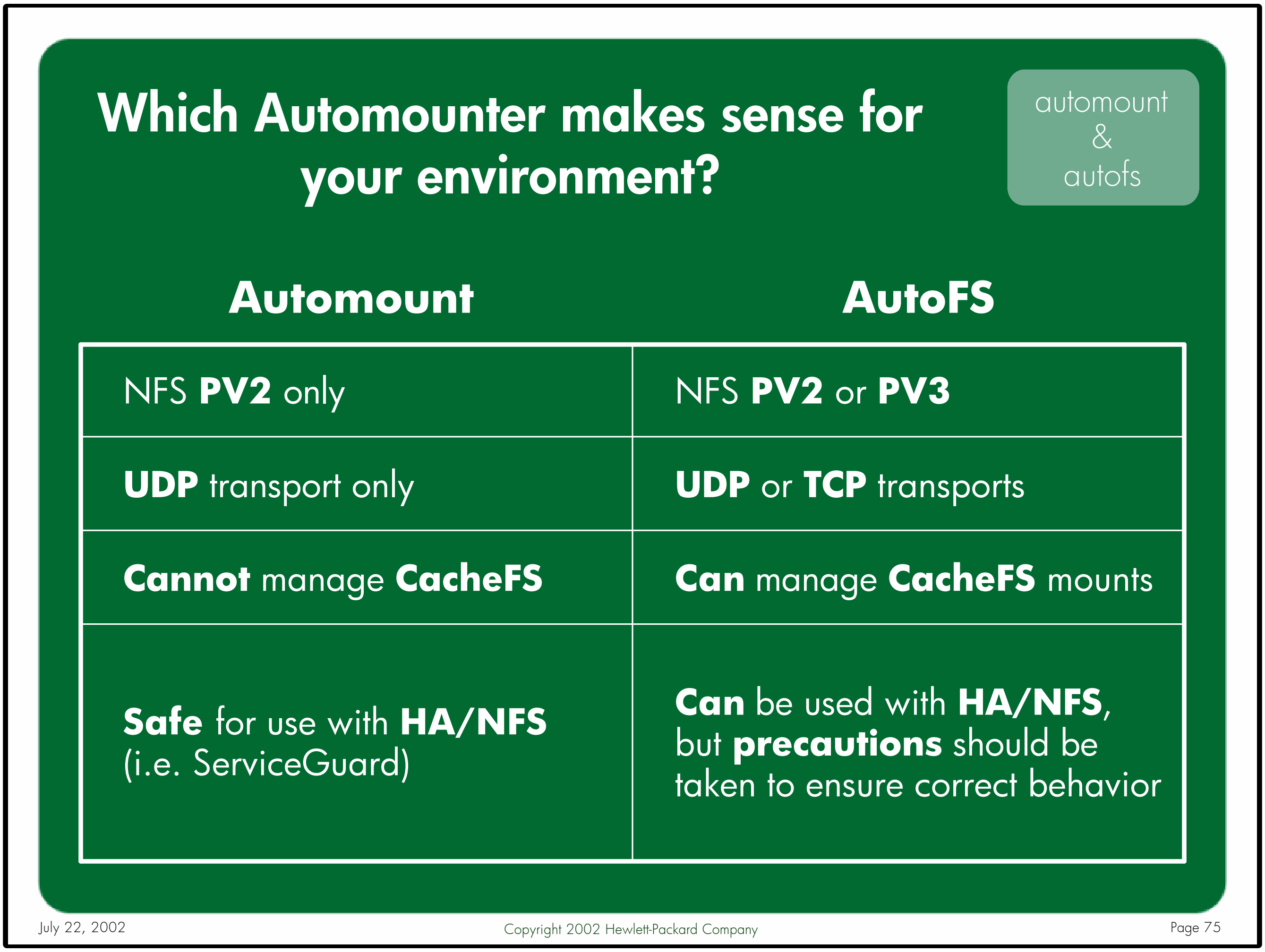
automount
&
autofs

- Replicated NFS Servers in Maps
  - ➢ Ensure the specified servers exist, respond quickly to mount requests, and contain the filesystem referenced in the map
- Environment Variables in Maps ("**-D**" option)
  - ➢ Ensure the pathnames resolved by variables exist on the server
- Hierarchical Maps
  - ➢ Entire hierarchy must be mounted/unmounted together
  - ➢ Adds overhead both to client's automounter and server's rpc.mountd when only a portion of the hierarchy is used

Notes: Both automount and AutoFS allow the administrator to configure multiple replicated NFS servers for a given mount point. If you plan to use this feature be certain to only list NFS servers that truly exist, are responding to mount requests quickly, and contain the filesystem referenced in the map. Otherwise the automounter will waste cycles looking for servers that don't exist, are down, or don't contain the data they need to satisfy the mount request.

Both automounters provide the ability to specify environment variables via the "–D" option and then reference those variables in the maps. This feature also requires a good deal of planning and coordination to ensure that the pathnames resolved via these environment variables actually exist on the target NFS servers.

Automount and AutoFS will attempt to unmount filesystems that are not in use every 5 minutes (default). While this value can add some overhead to the client system by having automounter attempt to unmount filesystems frequently, it can also add a substantial amount of load to the NFS server's rpc.mountd daemon – particularly when hierarchical maps are used, since automount attempts to unmount the entire hierarchy and must re-mount it if any part of the hierarchy is busy.



Which Automounter makes sense for your environment?

automount
&
autofs

Automount	AutoFS
NFS **PV2** only	NFS **PV2** or **PV3**
UDP transport only	**UDP** or **TCP** transports
Cannot manage **CacheFS**	**Can** manage **CacheFS** mounts
Safe for use with **HA/NFS** (i.e. ServiceGuard)	**Can** be used with **HA/NFS**, but **precautions** should be taken to ensure correct behavior

July 22, 2002

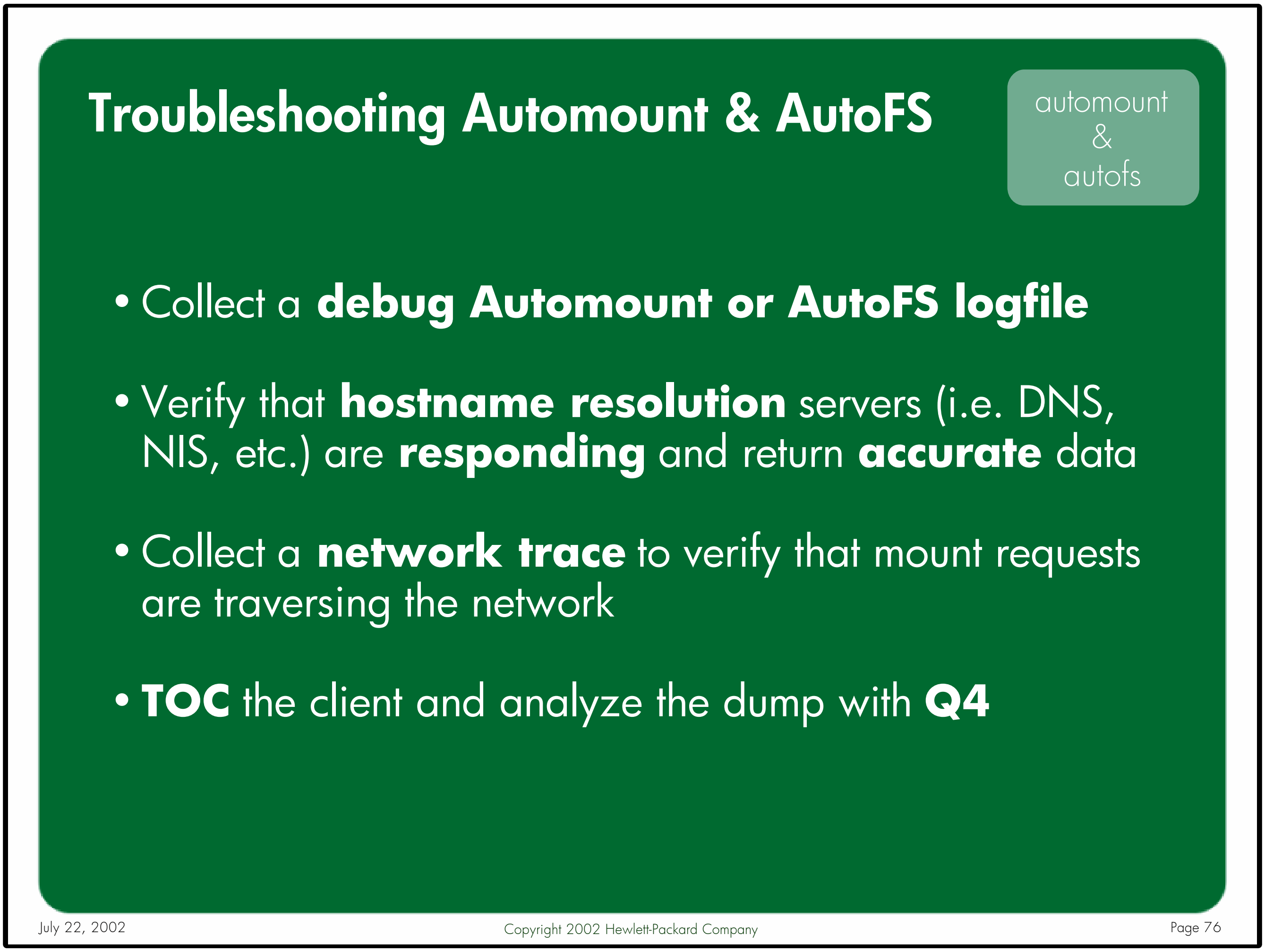
Notes:

Environments where Automount is Recommended

Given the earlier list of differences between automount and AutoFS, there is really only one environment where automount holds an advantage over AutoFS – on NFS servers running Highly Available NFS (i.e. ServiceGuard). Of course, if your environment only requires the use of NFS PV2, the UDP transport, 8KB read/write sizes, and does not need to manage CacheFS mounts, the legacy automounter will work fine.

Environments where AutoFS is Recommended

Since AutoFS includes all of the features provided by the legacy automounter and adds several new features, most NFS environments will benefit from using AutoFS in place of automounter. Any customers needing to integrate support for NFS PV3, NFS/TCP, or CacheFS into their automount environment will need to use AutoFS.



Troubleshooting Automount & AutoFS

automount
&
autofs

- Collect a **debug Automount or AutoFS logfile**
- Verify that **hostname resolution** servers (i.e. DNS, NIS, etc.) are **responding** and return **accurate** data
- Collect a **network trace** to verify that mount requests are traversing the network
- **TOC** the client and analyze the dump with **Q4**

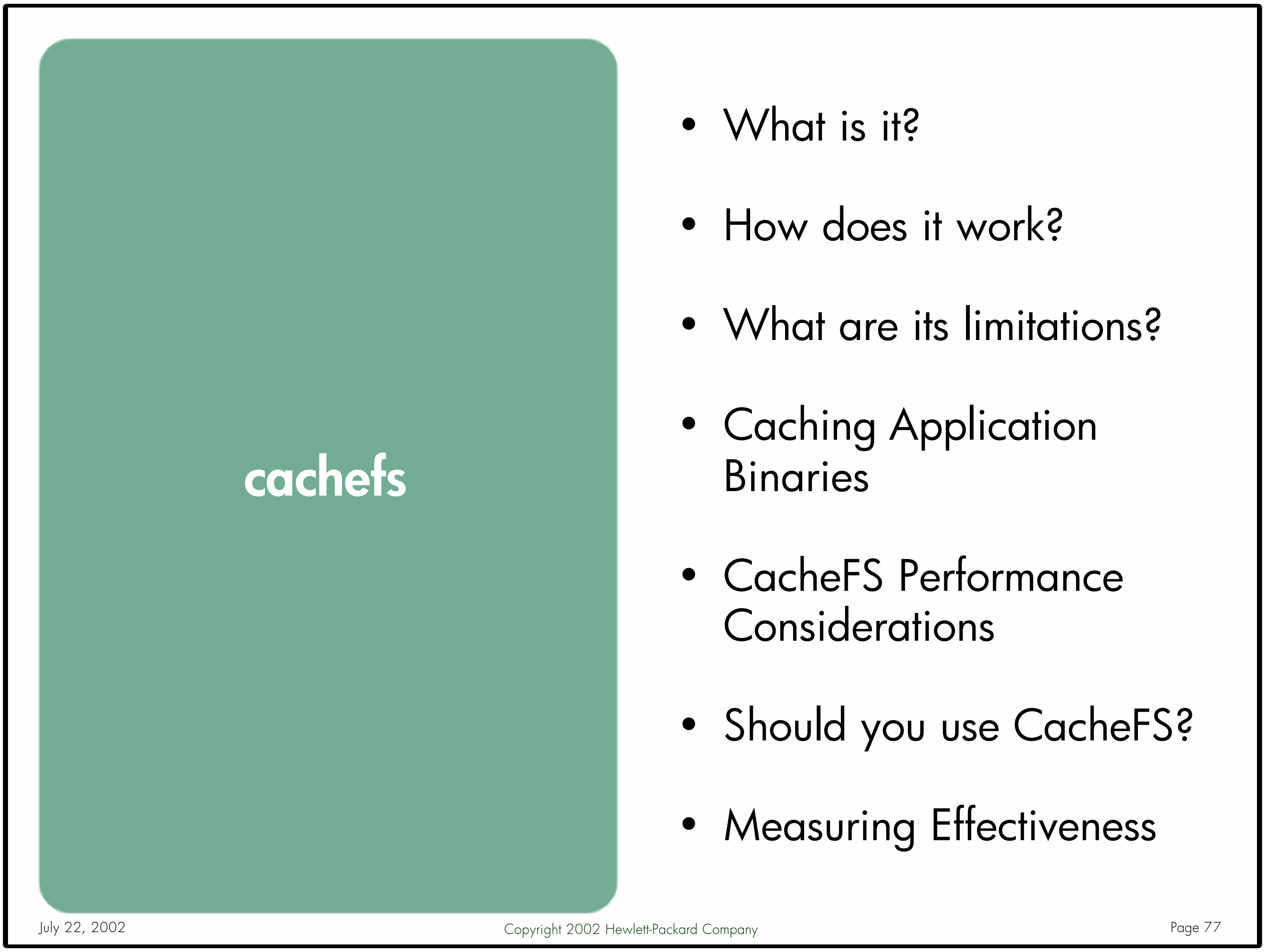
July 22, 2002

Page 76

Notes: Among the best sources of information for troubleshooting automount and AutoFS problems is a debug logfiles collected while reproducing the problem. Debug logging can be toggled on and off by sending the SIGUSR2 signal to the running “automount” (legacy automounter) or “automountd” (AutoFS) process (i.e. kill –17 <pid>). By default, the debug information is logged to the `/var/adm/automount.log` file.

In some cases, a network trace is also needed to fully understand the root cause of an automount problem. The trace can be used to determine whether the mount requests and replies are traversing the network successfully.

Since AutoFS is a multi-threaded process and spends much of its time in the kernel, in some cases a TOC dump must be collected and analyzed with Q4 in order to determine why AutoFS is misbehaving.



cachefs

- What is it?
- How does it work?
- What are its limitations?
- Caching Application Binaries
- CacheFS Performance Considerations
- Should you use CacheFS?
- Measuring Effectiveness

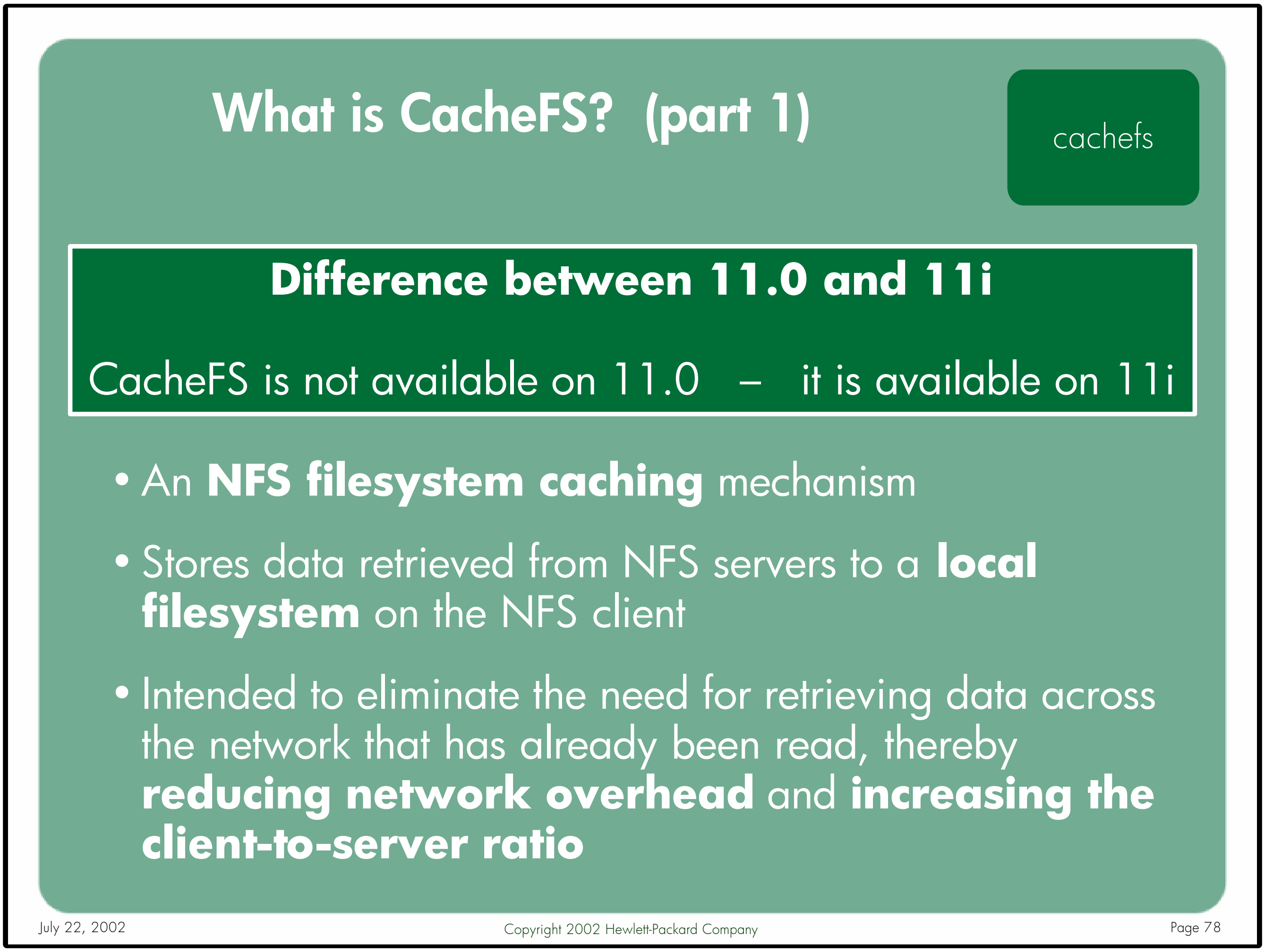
July 22, 2002

Page 77

Notes: This section begins by describing how CacheFS works and the potential benefits it can provide. This is followed by a list of the many limitations of CacheFS, and why it may not provide the promised benefits in every environment.

Next is a discussion of the unique considerations to understand when using CacheFS to cache application binaries, and how to best configure CacheFS when used to front NFS-based applications. This section describes an HP-specific CacheFS enhancement called the *rpages* mount option, and how this mount option influences CacheFS behavior and performance.

Following this is a discussion of the many ways you can configure your CacheFS clients to achieve optimal caching performance. This is followed by a discussion of whether CacheFS should be considered for your environment or not. Finally, for those customers who choose to implement CacheFS, are some guidelines for determining conclusively whether CacheFS is actually providing a benefit in your environment.



What is CacheFS? (part 1)

cachefs

Difference between 11.0 and 11i

CacheFS is not available on 11.0 – it is available on 11i

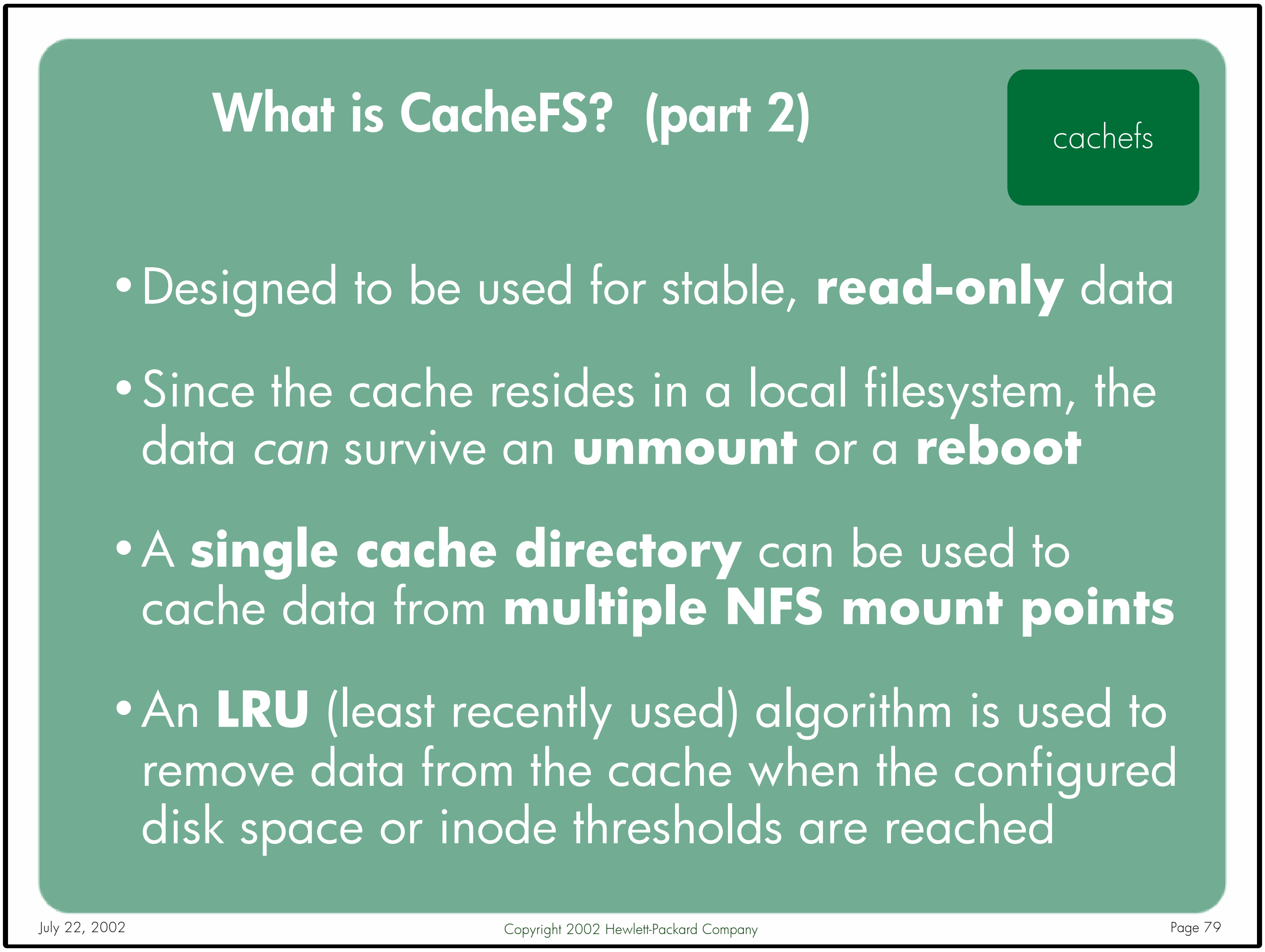
- An **NFS filesystem caching** mechanism
- Stores data retrieved from NFS servers to a **local filesystem** on the NFS client
- Intended to eliminate the need for retrieving data across the network that has already been read, thereby **reducing network overhead** and **increasing the client-to-server ratio**

July 22, 2002

Page 78

Notes: Since NFS is a file access protocol, one of the primary goals of NFS performance tuning is to reduce the amount of network traffic required to satisfy read requests. One method of reducing read traffic is to use client-side buffer cache and page cache to store the most recently accessed data. However, since physical memory is limited in size, the caching mechanisms often have to flush pages and retrieve them again from the server. Also, as more NFS clients are added to a given server, the file access times for each client increases, perpetuating the server performance load and impacting the overall network performance.

CacheFS (Cache File System) attempts to address these needs by keeping copies of data read from NFS servers on the client in the hopes of avoiding the need for future NFS read calls for the same data. CacheFS is a file system caching mechanism that stores data retrieved from NFS servers to a local filesystem on the client. NFS read requests for data residing in the cache directory can be retrieved locally, thereby eliminating the need for a network request and reducing overall server and network load. CacheFS is intended to increase the client-to-server ratio, and improve NFS read performance for clients using slow network links.



What is CacheFS? (part 2)

cachefs

- Designed to be used for stable, **read-only** data
- Since the cache resides in a local filesystem, the data *can* survive an **unmount** or a **reboot**
- A **single cache directory** can be used to cache data from **multiple NFS mount points**
- An **LRU** (least recently used) algorithm is used to remove data from the cache when the configured disk space or inode thresholds are reached

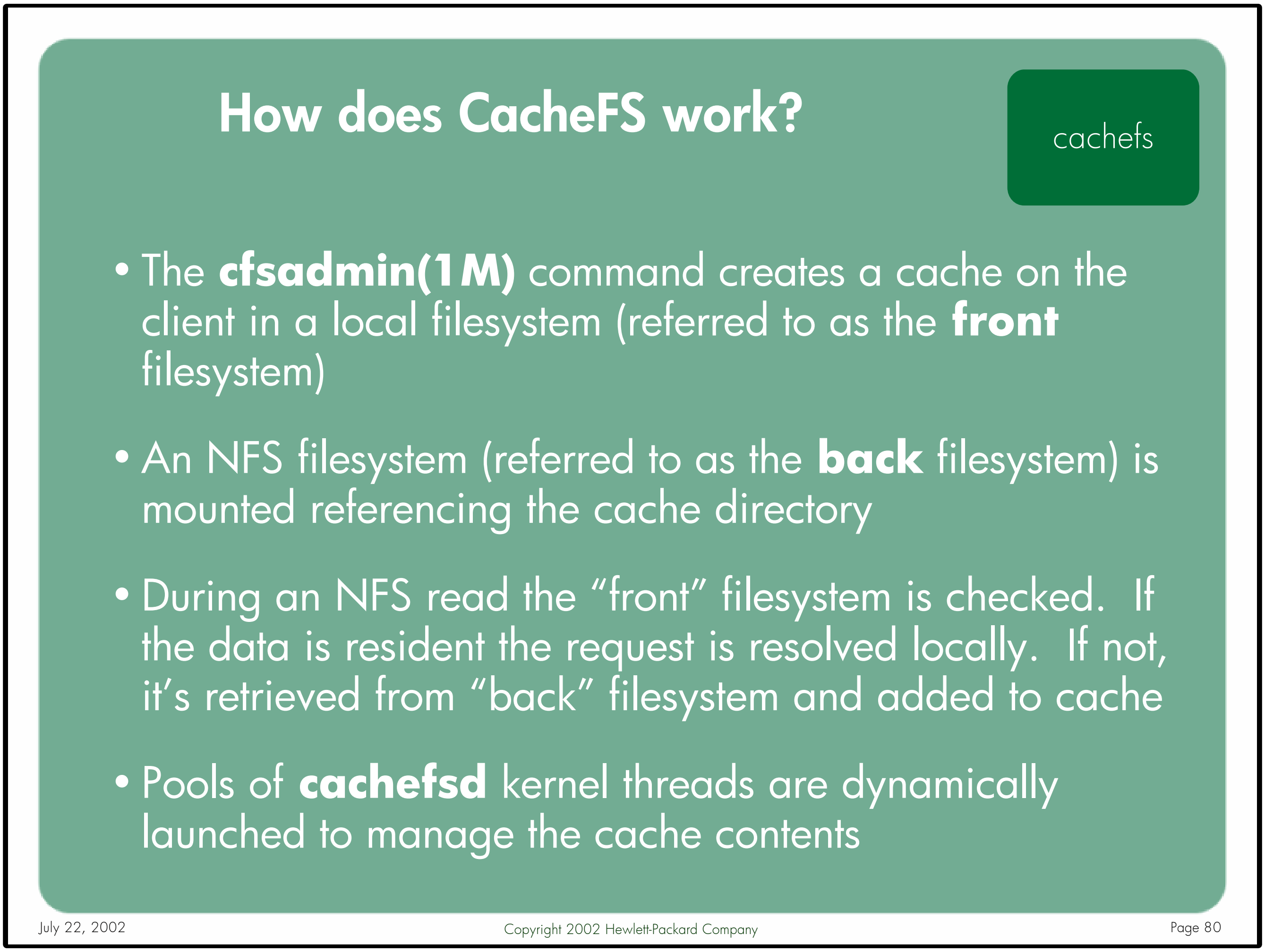
July 22, 2002

Page 79

Notes: CacheFS can supply a performance boost to those systems that spend a great deal of time accessing stable, read-only data over NFS mounts. Since the cache is maintained on a local client filesystem and not in memory, in most cases the cache contents remain even if the CacheFS filesystem is unmounted. In fact, the cache contents remain in the local filesystem even after a client reboot.

A single cache directory can be used to cache data from multiple NFS mounts. Once the configured thresholds of the cache have been reached, CacheFS implements a LRU (least recently used) algorithm for invalidating files from the cache.

Upon mounting a CacheFS filesystem, the kernel launches a number of “cachefsd” kernel threads to manage the cache contents. These threads are associated with a single “cachefsd” process.



How does CacheFS work?

cachefs

- The **cfsadmin(1M)** command creates a cache on the client in a local filesystem (referred to as the **front** filesystem)
- An NFS filesystem (referred to as the **back** filesystem) is mounted referencing the cache directory
- During an NFS read the "front" filesystem is checked. If the data is resident the request is resolved locally. If not, it's retrieved from "back" filesystem and added to cache
- Pools of **cachefsd** kernel threads are dynamically launched to manage the cache contents

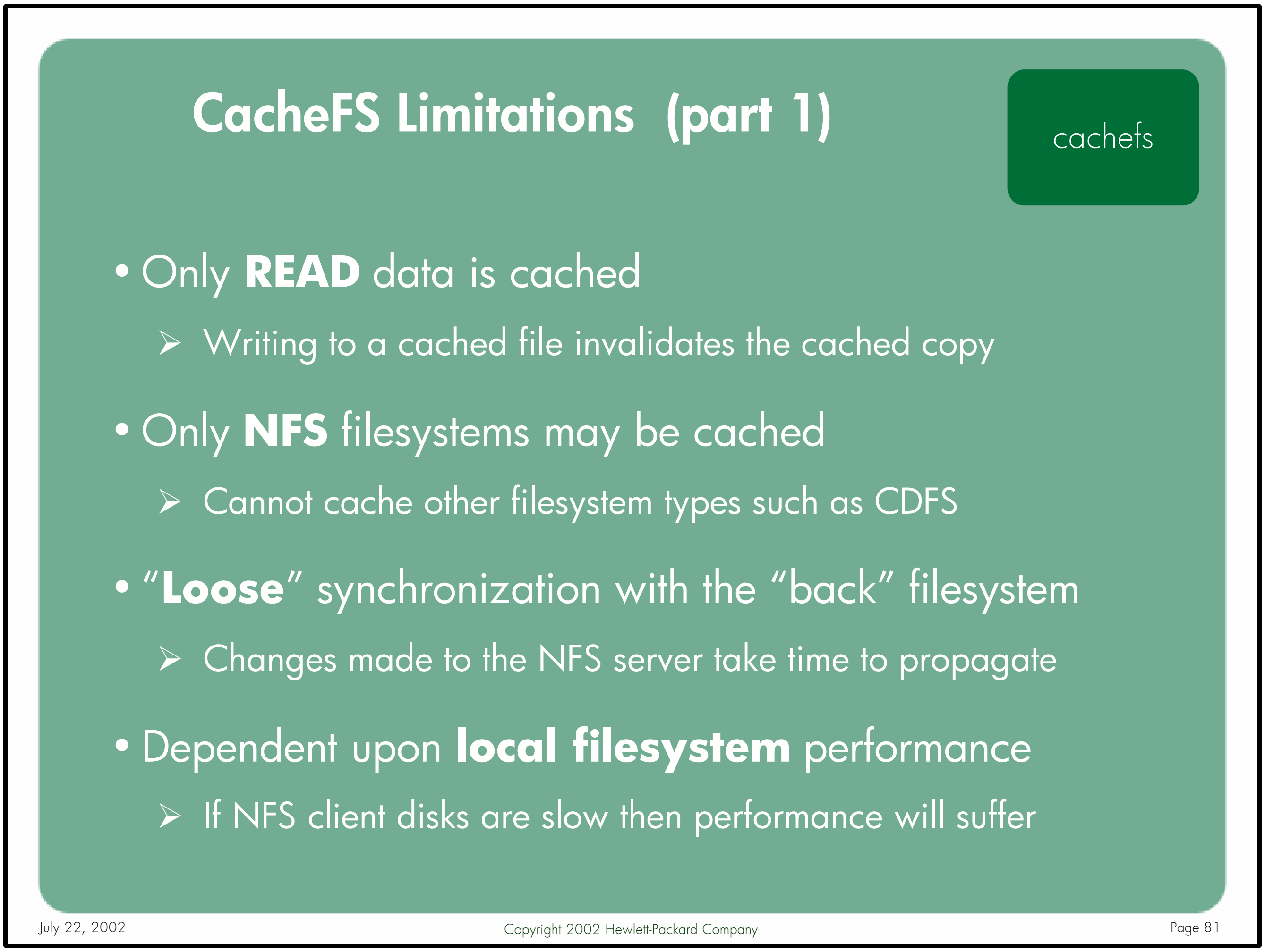
July 22, 2002

Page 80

Notes: The cfsadmin(1M) command is used to create the local cache directory and define the parameters of the cache. By default, the cache is allowed to consume up to 90% of the available disk space of the filesystem in which it resides.

The CacheFS file system caches data read from the NFS filesystem (known as the "back" filesystem) onto another, local filesystem (known as the "front" filesystem).

A separate pool of cachefsd threads is created to service each cache and each pool can have a maximum of 5 cachefsd threads associated with it at any time. The kernel launches new threads as requests for cached data increase.



CacheFS Limitations (part 1)

cachefs

- Only **READ** data is cached
  - ➢ Writing to a cached file invalidates the cached copy
- Only **NFS** filesystems may be cached
  - ➢ Cannot cache other filesystem types such as CDFS
- “**Loose**” synchronization with the “back” filesystem
  - ➢ Changes made to the NFS server take time to propagate
- Dependent upon **local filesystem** performance
  - ➢ If NFS client disks are slow then performance will suffer

July 22, 2002

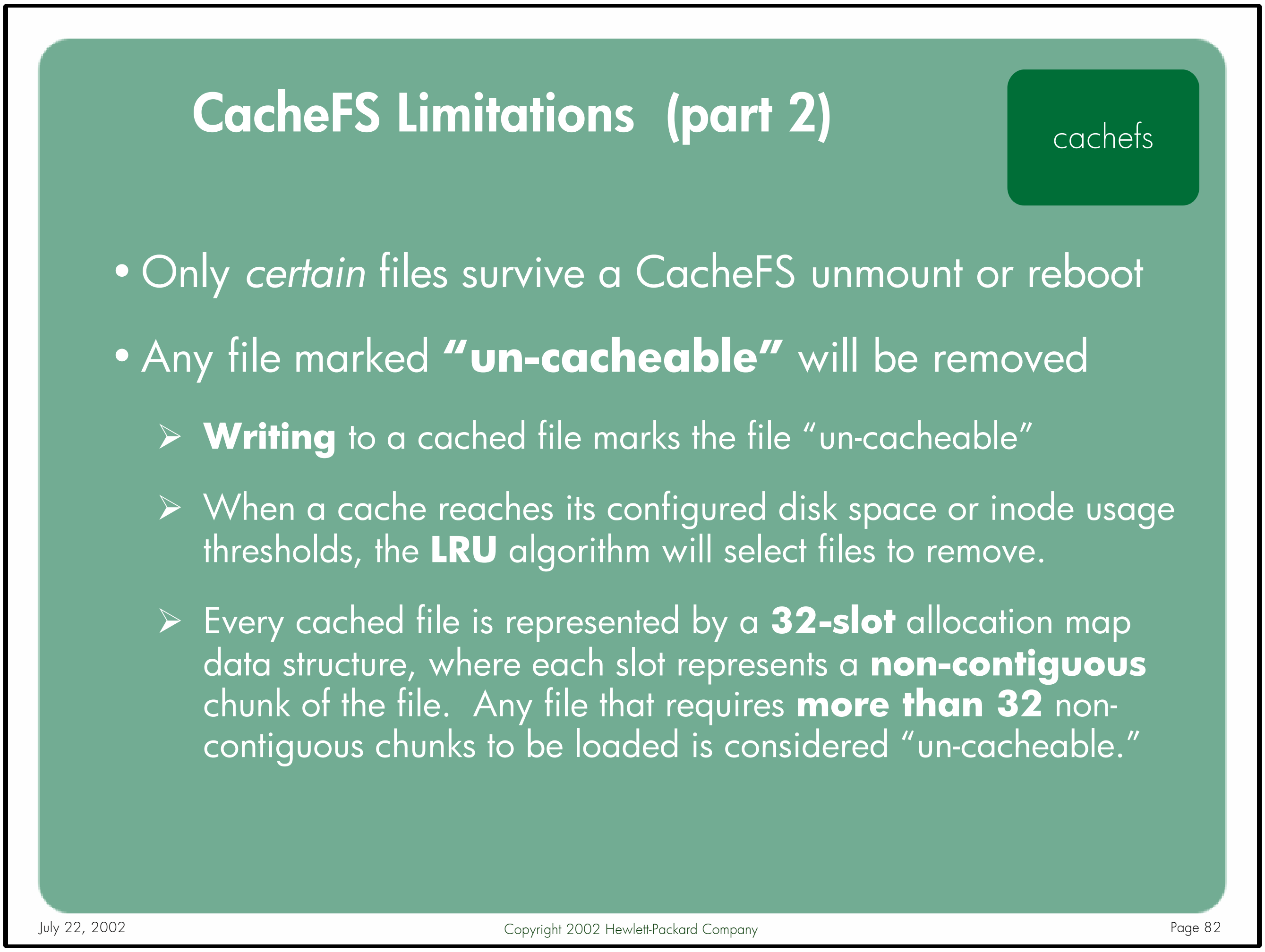
Page 81

Notes: CacheFS does not cache writes. Any writes performed to a file currently residing in the local cache will result in that file being invalidated from the cache. These added steps of invalidating the cache and redirecting the write() call to the back filesystem can add some overhead to the writing operation.

HP’s current implementation of CacheFS only supports NFS as the back filesystem. Other vendors allow CDFS filesystems to be cached as well.

CacheFS does not maintain tight synchronization with the remote NFS server. Attribute or data changes done on the server by one client may take some time to propagate to the caches on other clients. For this reason, it is recommended that CacheFS not be used on mounts where data files are frequently updated.

CacheFS performance is affected by many factors, including client local filesystem performance. On an NFS client with a fast network interface, mounting a filesystem from a very fast NFS server, but using a slow, heavily loaded local filesystem for the cache directory, CacheFS access may actually be slower than accessing the data directly from the NFS server.



CacheFS Limitations (part 2)

cachefs

- Only *certain* files survive a CacheFS unmount or reboot
- Any file marked **"un-cacheable"** will be removed
  - ➢ **Writing** to a cached file marks the file "un-cacheable"
  - ➢ When a cache reaches its configured disk space or inode usage thresholds, the **LRU** algorithm will select files to remove.
  - ➢ Every cached file is represented by a **32-slot** allocation map data structure, where each slot represents a **non-contiguous** chunk of the file. Any file that requires **more than 32** non-contiguous chunks to be loaded is considered "un-cacheable."

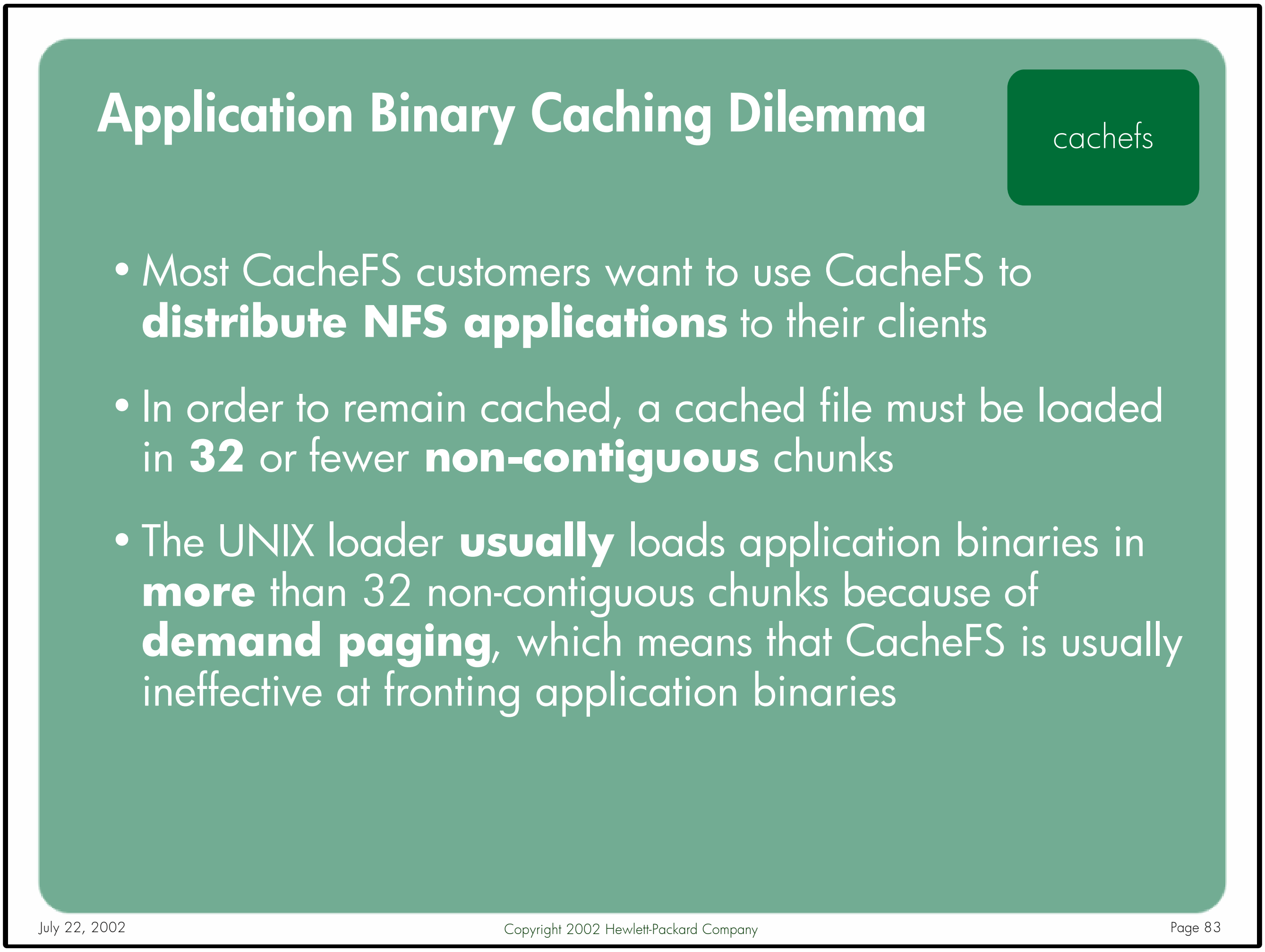
July 22, 2002

Page 82

Notes: There are three main reasons CacheFS will disable caching for a file:

- When a process on the NFS client *writes* to the cached file
- When a cache reaches its configured disk space or inode usage thresholds, the LRU algorithm will select files to remove from the cache
- When 33 or more *non-contiguous* chunks of the cached file are referenced

Once a file is flagged as "un-cacheable," any read requests for this file must be serviced by the back (NFS) filesystem, effectively nullifying any CacheFS benefits for this file. All files that have been marked "un-cacheable" are removed from the cache directory when the CacheFS filesystem is unmounted or the NFS client system is rebooted.



Application Binary Caching Dilemma

cachefs

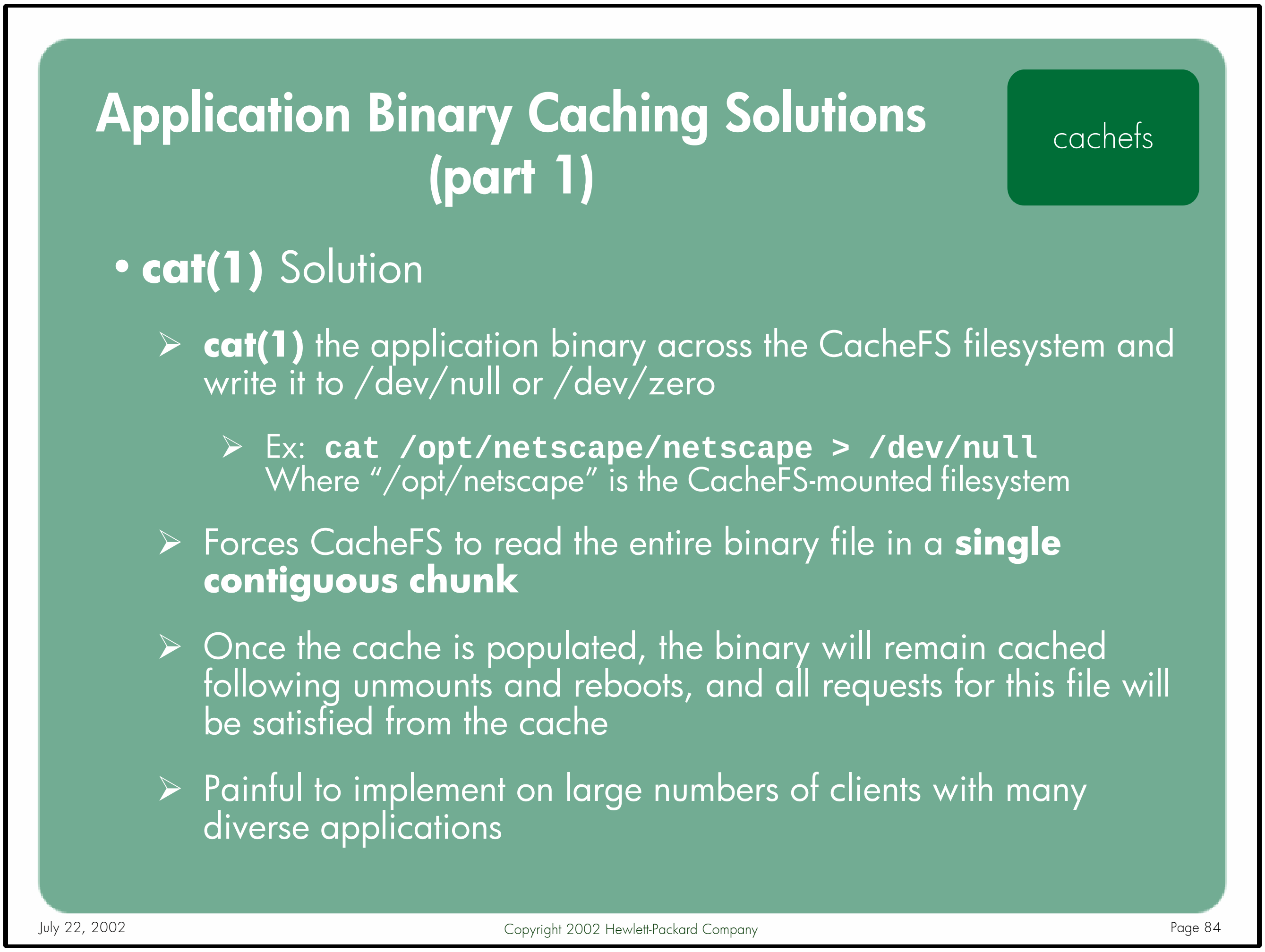
- Most CacheFS customers want to use CacheFS to **distribute NFS applications** to their clients
- In order to remain cached, a cached file must be loaded in **32** or fewer **non-contiguous** chunks
- The UNIX loader **usually** loads application binaries in **more** than 32 non-contiguous chunks because of **demand paging**, which means that CacheFS is usually ineffective at fronting application binaries

July 22, 2002

Page 83

Notes: Since CacheFS is designed to work best with NFS filesystems that contain read-only data that does not change frequently, one of the most common uses for CacheFS is to manage application binaries. The behavior expected by most customers is that when the binary is executed the first time via a CacheFS mount it will be cached on the client, and subsequent accesses will be satisfied from the local cached copy, thus avoiding the need for an NFS read request. Another expectation is that these binaries would remain cached following an unmount or a reboot. Unfortunately, CacheFS rarely behaves this way on either HP-UX or Solaris platforms.

CacheFS uses a 32-slot allocation map to represent every file it caches. Each slot consists of a starting offset and a length value, which means that a cached file can only be represented in 32 non-contiguous chunks. Any attempt to access more than 32 non-contiguous regions of the file causes CacheFS to mark this file as invalid – effectively disabling caching for this file. The “demand paging” algorithm used by the HP-UX (and Solaris) loader typically results in application binaries being loaded in more than 32 non-contiguous regions, which limits CacheFS’ effectiveness when fronting NFS-based applications.



Application Binary Caching Solutions (part 1)

cachefs

- **cat(1)** Solution
  - ➢ **cat(1)** the application binary across the CacheFS filesystem and write it to /dev/null or /dev/zero
    - ➢ Ex: **`cat /opt/netscape/netscape > /dev/null`** Where "/opt/netscape" is the CacheFS-mounted filesystem
  - ➢ Forces CacheFS to read the entire binary file in a **single contiguous chunk**
  - ➢ Once the cache is populated, the binary will remain cached following unmounts and reboots, and all requests for this file will be satisfied from the cache
  - ➢ Painful to implement on large numbers of clients with many diverse applications

July 22, 2002

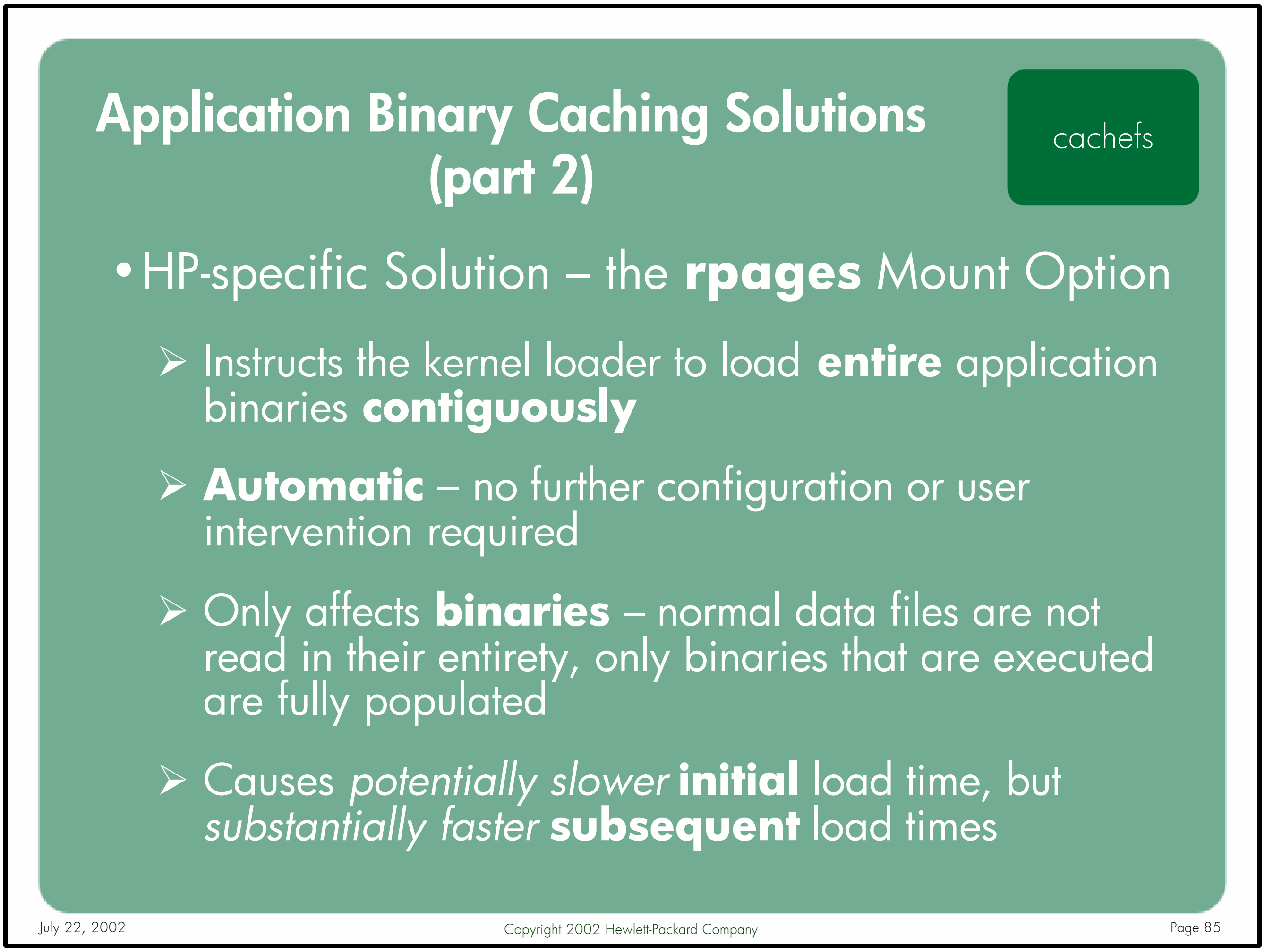
Page 84

Notes: One way to guarantee that CacheFS stores an application binary, or any other type of file, in a single contiguous collection of disk space is to read the entire target file across the CacheFS mount point via the cat(1) command. For example:

```
cat /cfs_mount/myprog > /dev/null
```

Using cat in this manner forces CacheFS to read the entire "myprog" file sequentially, consuming only one slot in the allocation map. Remember, the map entries consist of a starting offset value and a length field. Therefore, if a 20MB file is read sequentially across a CacheFS mount, the file can be represented in one slot (i.e. starting offset = 0, length = 20MB). Once the cat command completes, the client system has a complete copy of the "myprog" binary in its local filesystem. This cached binary will survive a CacheFS filesystem unmount or a client reboot.

This is not a very practical solution in most CacheFS environments because there are typically hundreds, or even thousands, of NFS clients to distribute applications to, and in many cases the clients will run different sets of applications.



Application Binary Caching Solutions (part 2)

cachefs

- HP-specific Solution – the **rpages** Mount Option
  - Instructs the kernel loader to load **entire** application binaries **contiguously**
  - **Automatic** – no further configuration or user intervention required
  - Only affects **binaries** – normal data files are not read in their entirety, only binaries that are executed are fully populated
  - Causes *potentially slower* **initial** load time, but *substantially faster* **subsequent** load times

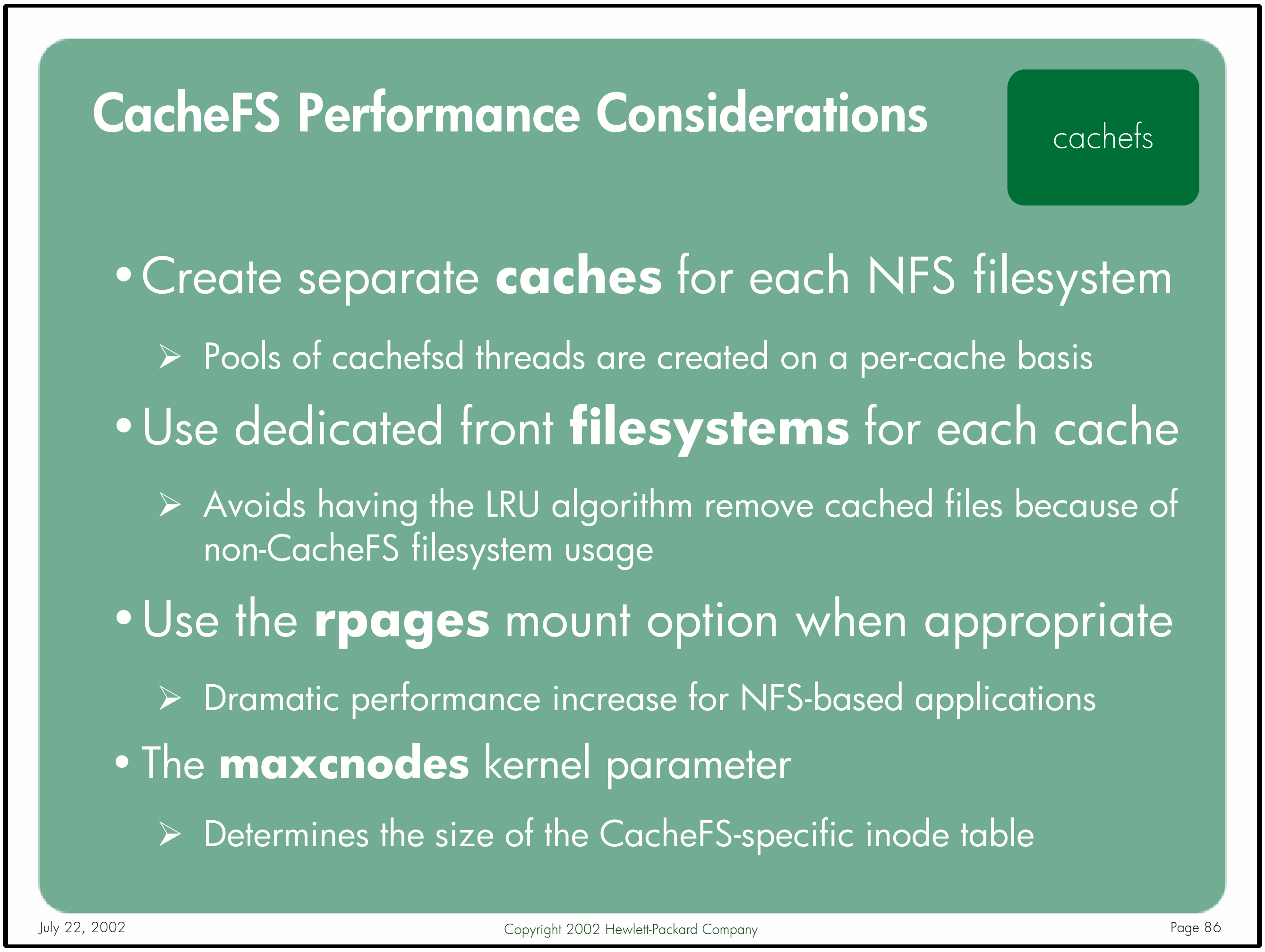
July 22, 2002

Page 85

Notes: The *rpages* option instructs CacheFS to behave as follows: whenever an application residing in a cached NFS filesystem is executed, the kernel checks the front filesystem to see if a *complete* copy of the application binary is present in the local cache; if not, the client will sequentially read the entire application binary from the server and cache a local copy. This automatic caching of complete binaries occurs without any user intervention. Any future requests for this file are satisfied from the front filesystem, and this cached binary will remain intact during a CacheFS filesystem unmount or a client reboot.

An important point to understand about the *rpages* functionality is that it only forces the client to cache complete copies of application files that are *executed*. It does not automatically load entire copies of files that the client merely *reads*.

The *rpages* option may cause initial binary load times to increase (I.e. the first time the application is run on a CacheFS client with an un-populated cache) since the entire binary will be loaded across the network. However, subsequent run times are usually substantially faster.



CacheFS Performance Considerations

cachefs

- Create separate **caches** for each NFS filesystem
  - ➢ Pools of cachefsd threads are created on a per-cache basis
- Use dedicated front **filesystems** for each cache
  - ➢ Avoids having the LRU algorithm remove cached files because of non-CacheFS filesystem usage
- Use the **rpages** mount option when appropriate
  - ➢ Dramatic performance increase for NFS-based applications
- The **maxcnodes** kernel parameter
  - ➢ Determines the size of the CacheFS-specific inode table

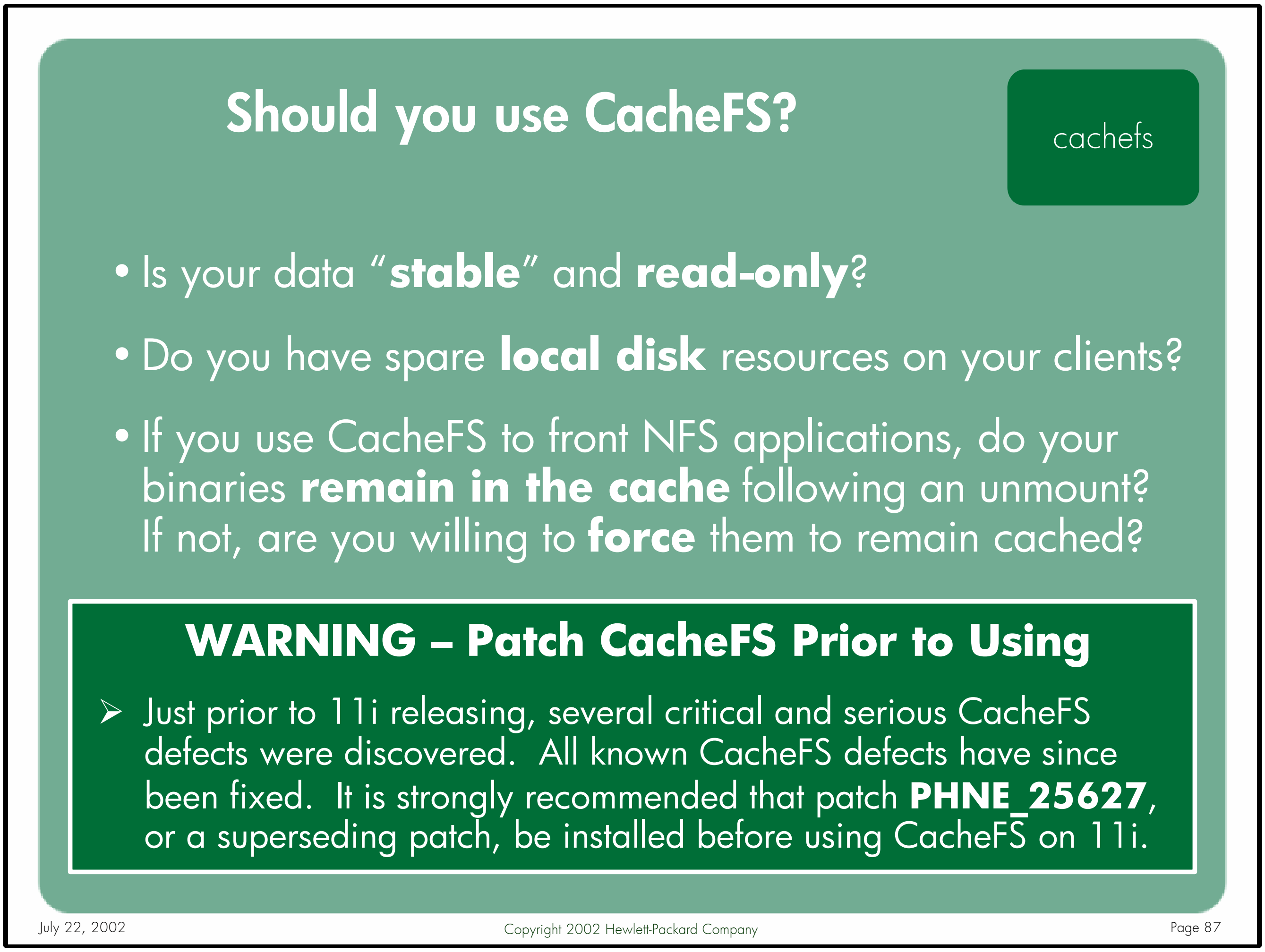
July 22, 2002

Page 86

Notes: The recommendation is to create a separate cache for each CacheFS mount point, thus avoiding the case where data from one NFS filesystem forces CacheFS to remove files cached by another NFS filesystem when the cache disk space or inode resource thresholds are reached.

It is important to understand that the kernel creates cachefsd thread pools on a per-cache basis — not a per-CacheFS mount basis. In other words, if three NFS filesystems share a single cache, all requests for these cached resources will be serviced by a single pool of cachefsd threads. If, however, there are three separate caches configured, and each of the NFS filesystems are mounted using different caches, then each cached filesystem will be serviced by its own pool of threads. By creating separate caches for each CacheFS filesystem, the kernel creates dedicated pools of cachefsd threads to service each CacheFS mount point.

The maxcnodes kernel parameter configures the size of the CacheFS-specific inode table. By default, this parameter is set to the same value as ncsize, which is sized based on ninode. Typically this parameter is tuned by tuning ncsize or ninode.



Should you use CacheFS?

cachefs

- Is your data “**stable**” and **read-only**?
- Do you have spare **local disk** resources on your clients?
- If you use CacheFS to front NFS applications, do your binaries **remain in the cache** following an unmount? If not, are you willing to **force** them to remain cached?

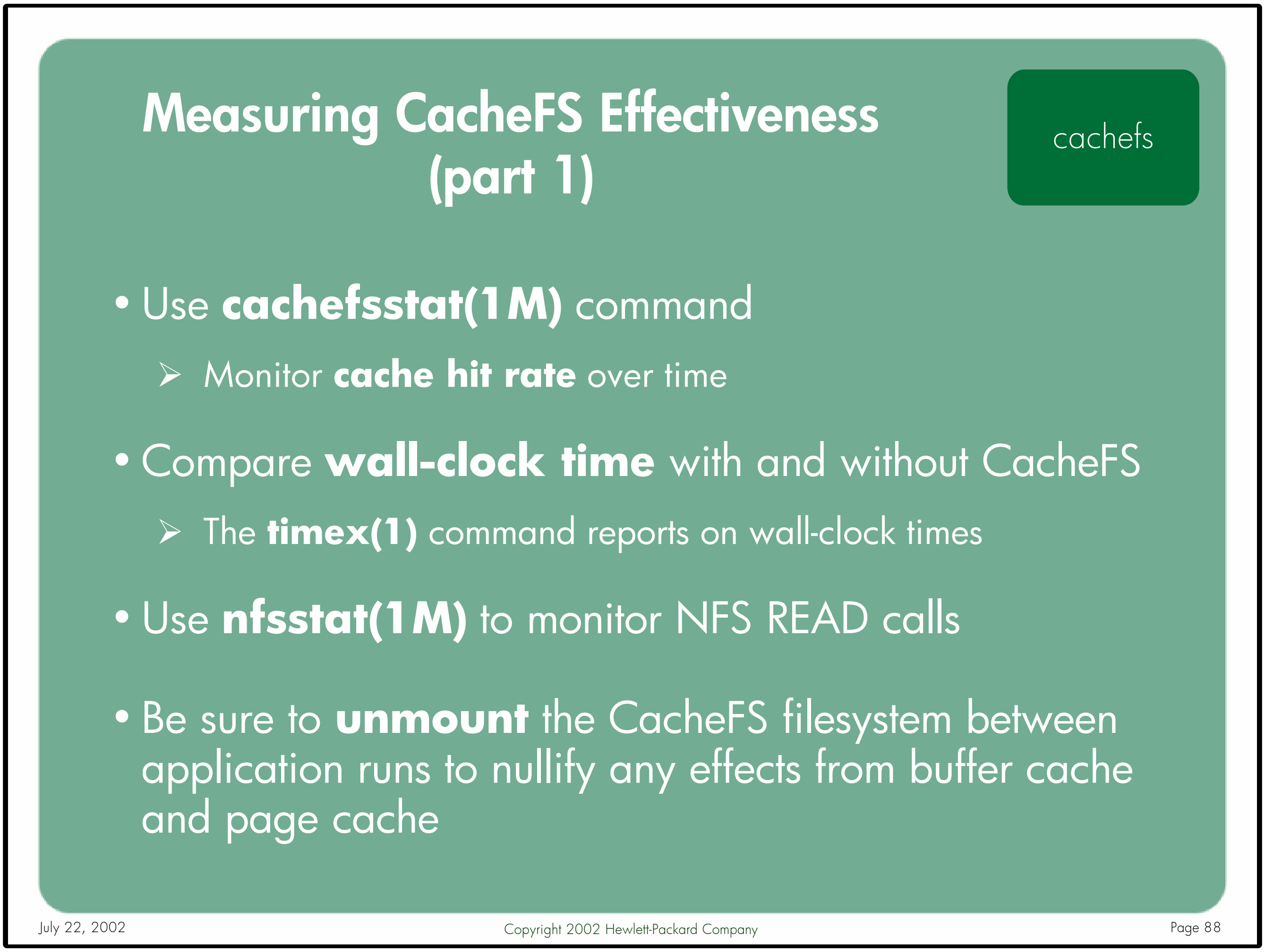
WARNING – Patch CacheFS Prior to Using

➢ Just prior to 11i releasing, several critical and serious CacheFS defects were discovered. All known CacheFS defects have since been fixed. It is strongly recommended that patch **PHNE_25627**, or a superseding patch, be installed before using CacheFS on 11i.

July 22, 2002

Page 87

Notes: As with most system configuration issues, the decision to use CacheFS should be made by an experienced system administrator who is familiar with the applications used by the client.



Measuring CacheFS Effectiveness (part 1)

cachefs

- Use **cachefsstat(1M)** command
  - ➢ Monitor **cache hit rate** over time
- Compare **wall-clock time** with and without CacheFS
  - ➢ The **timex(1)** command reports on wall-clock times
- Use **nfsstat(1M)** to monitor NFS READ calls
- Be sure to **unmount** the CacheFS filesystem between application runs to nullify any effects from buffer cache and page cache

July 22, 2002

Page 88

Notes: There are several methodologies for determining whether CacheFS is having a positive impact on your NFS environment:

1. Use the cachefsstat(1M) command to monitor the cache hit rate over a period of time.

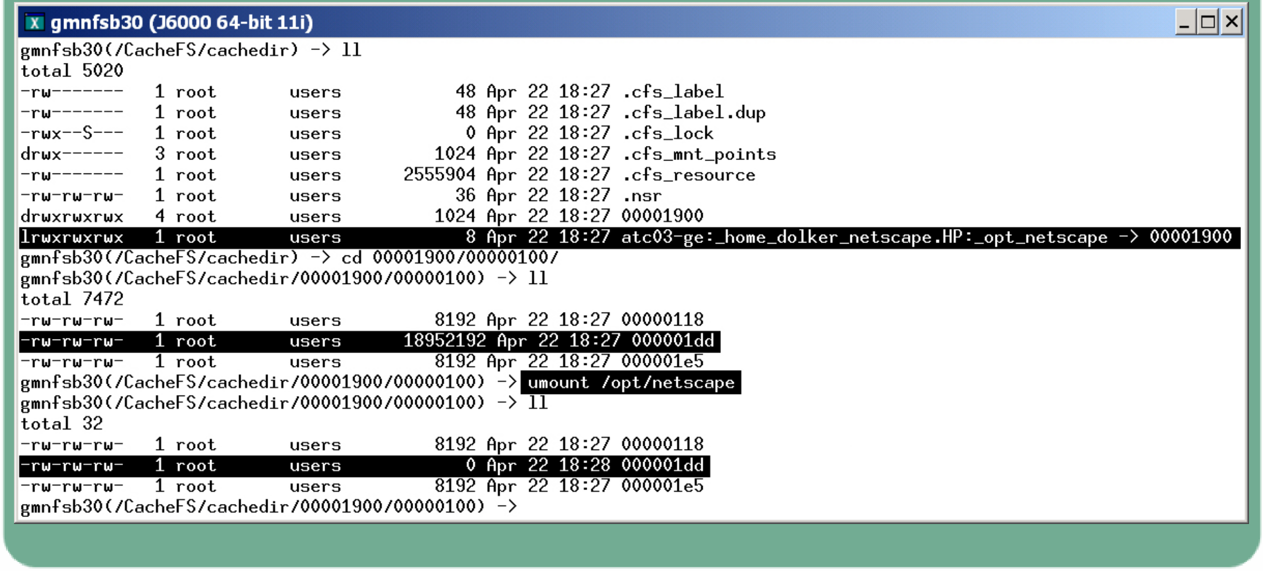
2. Try performing the same commands via the CacheFS mountpoint and then via an NFS mountpoint. Use the timex(1) command to see if any noticeable wall-clock time differences are seen with CacheFS.

3. Perform the same commands via the CacheFS mountpoint several times, but be sure to unmount and remount the filesystem between attempts to nullify any effect from the client’s buffer cache or page cache. As before, timex(1) can be used to compare times between runs. Also, the nfsstat(1M) command can be used to monitor how many NFS READ calls are made during each run. The nfsstat(1M) output should show definitively if CacheFS is eliminating the need for NFS read calls in subsequent runs.

Measuring CacheFS Effectiveness (part 2)

cachefs

- Examine the **contents of the cache** via “ls -l” **before** and **after** unmounting the CacheFS filesystem



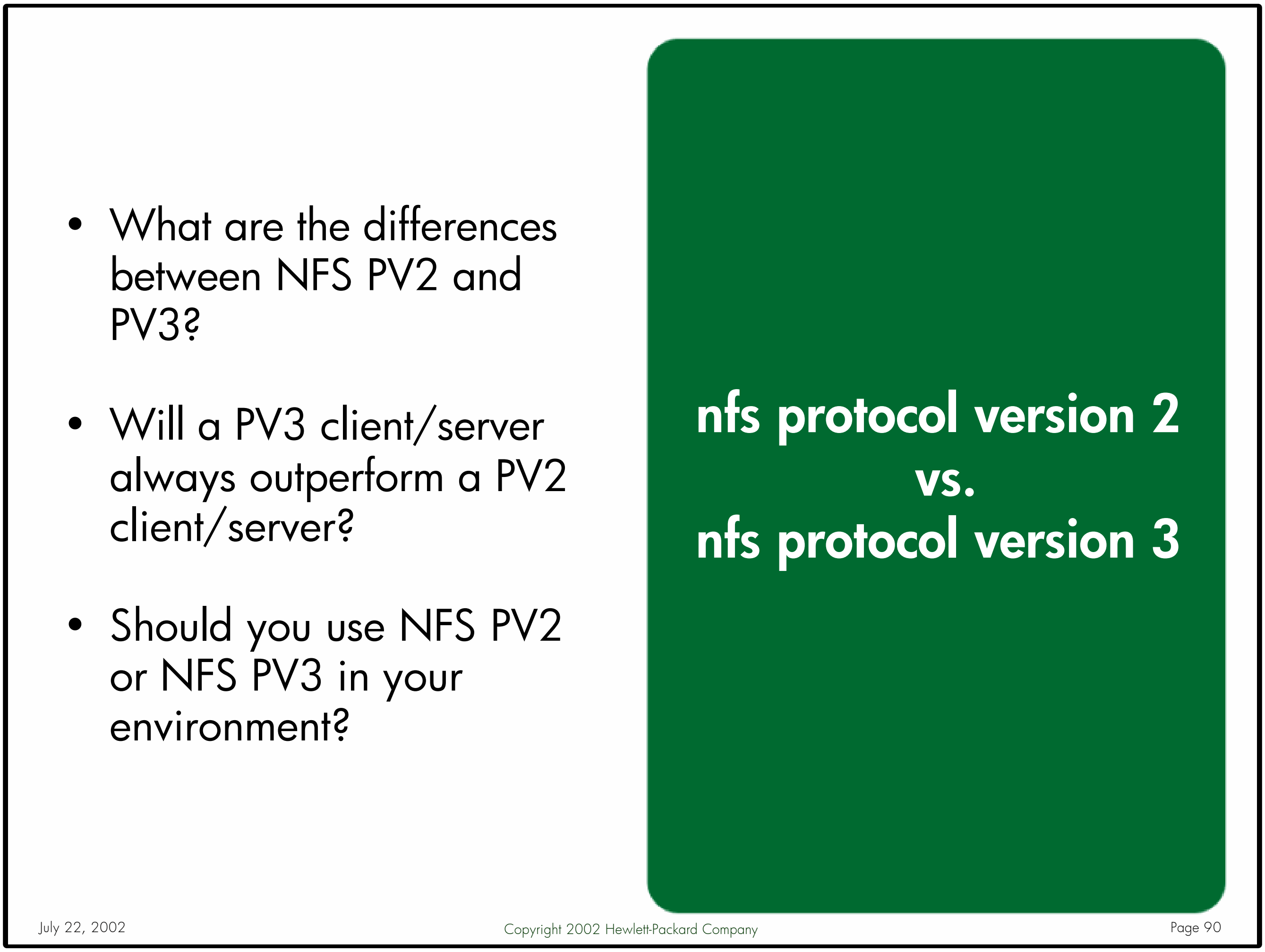
gmnfsb30 (J6000 64-bit 11i)

```
gmnfsb30(/CacheFS/cachedir) -> ll
total 5020
-rw-------   1 root       users           48 Apr 22 18:27 .cfs_label
-rw-------   1 root       users           48 Apr 22 18:27 .cfs_label.dup
-rwx--S---   1 root       users            0 Apr 22 18:27 .cfs_lock
drwx------   3 root       users         1024 Apr 22 18:27 .cfs_mnt_points
-rw-------   1 root       users      2555904 Apr 22 18:27 .cfs_resource
-rw-rw-rw-   1 root       users           36 Apr 22 18:27 .nsr
drwxrwxrwx   4 root       users         1024 Apr 22 18:27 00001900
lrwxrwxrwx   1 root       users            8 Apr 22 18:27 atc03-ge:_home_dolker_netscape.HP:_opt_netscape -> 00001900
gmnfsb30(/CacheFS/cachedir) -> cd 00001900/00000100/
gmnfsb30(/CacheFS/cachedir/00001900/00000100) -> ll
total 7472
-rw-rw-rw-   1 root       users         8192 Apr 22 18:27 00000118
-rw-rw-rw-   1 root       users      18952192 Apr 22 18:27 000001dd
-rw-rw-rw-   1 root       users         8192 Apr 22 18:27 000001e5
gmnfsb30(/CacheFS/cachedir/00001900/00000100) -> umount /opt/netscape
gmnfsb30(/CacheFS/cachedir/00001900/00000100) -> ll
total 32
-rw-rw-rw-   1 root       users         8192 Apr 22 18:27 00000118
-rw-rw-rw-   1 root       users            0 Apr 22 18:28 000001dd
-rw-rw-rw-   1 root       users         8192 Apr 22 18:27 000001e5
gmnfsb30(/CacheFS/cachedir/00001900/00000100) ->
```

Notes: Since the CacheFS cache resides in a local filesystem on the NFS client, the actual contents of the cache can be viewed like any other directory – via the ls(1) command. The names of the files in the cache do not match the names of the files in the back filesystem, so some intuition must be used to determine which file in the front filesystem is the cached equivalent of a specific file in the back filesystem.

In the above example, the `/opt/netscape` directory was mounted via CacheFS to an 11i client without the *rpages* mount option. The Netscape Communicator® application was launched on the client via the CacheFS mount point. At this point, an “ls -l” of the cache directory shows a large file (a cached version of the netscape binary) is resident in the cache. After unmounting the `/opt/netscape` directory, a second “ls -l” shows this same file has been zeroed out – effectively nullifying any CacheFS benefit. As explained previously, the fact that this file was invalidated from the cache at unmount time indicates that the HP-UX loader used more than 32 non-contiguous chunks to load the binary, exceeding the 32-slot map limit.

The before and after “ls -l” outputs of the cache directory provide definitive proof of whether CacheFS is really offering any benefit to your NFS applications or not.



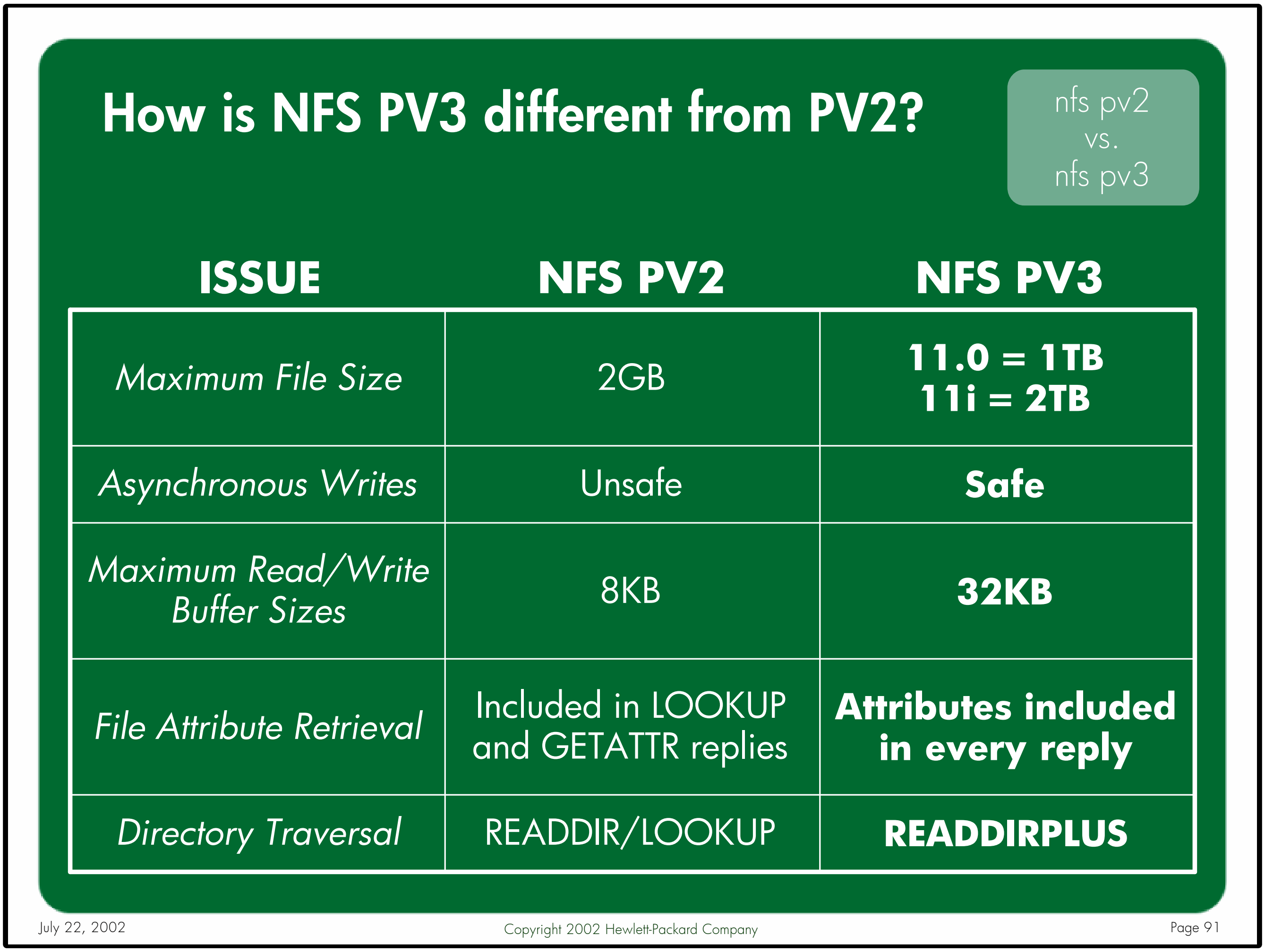
- What are the differences between NFS PV2 and PV3?
- Will a PV3 client/server always outperform a PV2 client/server?
- Should you use NFS PV2 or NFS PV3 in your environment?

nfs protocol version 2
vs.
nfs protocol version 3

July 22, 2002 Copyright 2002 Hewlett-Packard Company Page 90

Notes: NFS, like most protocols, continues to evolve over time. The original version of NFS (Protocol Version 1) existed only within Sun Microsystems and was never released to the public. Protocol Version 2 (PV2) was implemented in 1984 (RFC1094) and it enjoyed a good deal of popularity as a file access protocol. As time went by, some of the limitations imposed by the PV2 design necessitated the development of its replacement – Protocol Version 3 (PV3) – which was introduced in February1994 (RFC1813). This protocol offered many enhancements that made it far superior to its predecessor. Most of these improvements deal specifically with NFS performance.

This section discusses the many differences between NFS PV2 and NFS PV3 with respect to performance. It describes several scenarios where a PV2 mountpoint might actually outperform a PV3 mountpoint. It also explains the criteria for deciding which NFS protocol is more appropriate to use in your environment.



How is NFS PV3 different from PV2?

nfs pv2
vs.
nfs pv3

ISSUE	NFS PV2	NFS PV3
Maximum File Size	2GB	**11.0 = 1TB 11i = 2TB**
Asynchronous Writes	Unsafe	**Safe**
Maximum Read/Write Buffer Sizes	8KB	**32KB**
File Attribute Retrieval	Included in LOOKUP and GETATTR replies	**Attributes included in every reply**
Directory Traversal	READDIR/LOOKUP	**READDIRPLUS**

July 22, 2002

Page 91

Notes: **Maximum File Size Supported by NFS Client**

PV2 uses 32-bit signed integers for file offsets. PV3 uses 64-bit unsigned.

Asynchronous Writes

PV2 originally required all writes to be synchronous. Async writing was later added to PV2, but in an unsafe fashion (i.e. data loss can occur). PV3 writes data asynchronously by default and does so using a "safe" method.

Maximum Read/Write Buffer Sizes

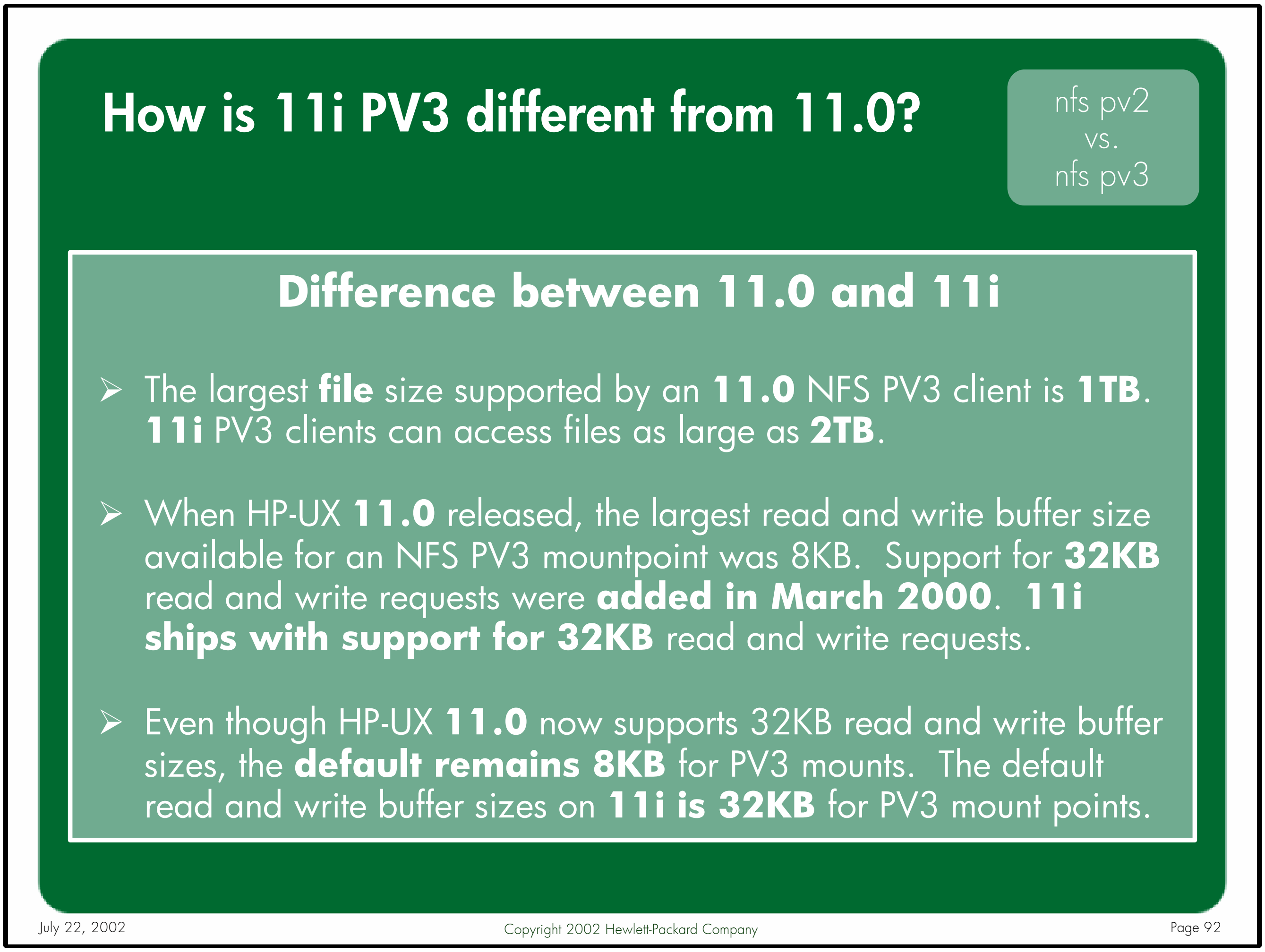
PV2 fixed the largest NFS read or write request at 8KB. HP's PV3 allows 32KB.

File Attribute Retrieval

File and directory attributes are only returned in select replies from a PV2 server. A PV3 server includes attribute information as part of every reply.

Directory Traversal

PV2 uses READDIR to get directory contents and then sends a LOOKUP for each file to get its attributes. PV3's READDIRPLUS call combines these functions.



How is 11i PV3 different from 11.0?

nfs pv2
vs.
nfs pv3

Difference between 11.0 and 11i

- The largest **file** size supported by an **11.0** NFS PV3 client is **1TB**. **11i** PV3 clients can access files as large as **2TB**.
- When HP-UX **11.0** released, the largest read and write buffer size available for an NFS PV3 mountpoint was 8KB. Support for **32KB** read and write requests were **added in March 2000**. **11i ships with support for 32KB** read and write requests.
- Even though HP-UX **11.0** now supports 32KB read and write buffer sizes, the **default remains 8KB** for PV3 mounts. The default read and write buffer sizes on **11i is 32KB** for PV3 mount points.

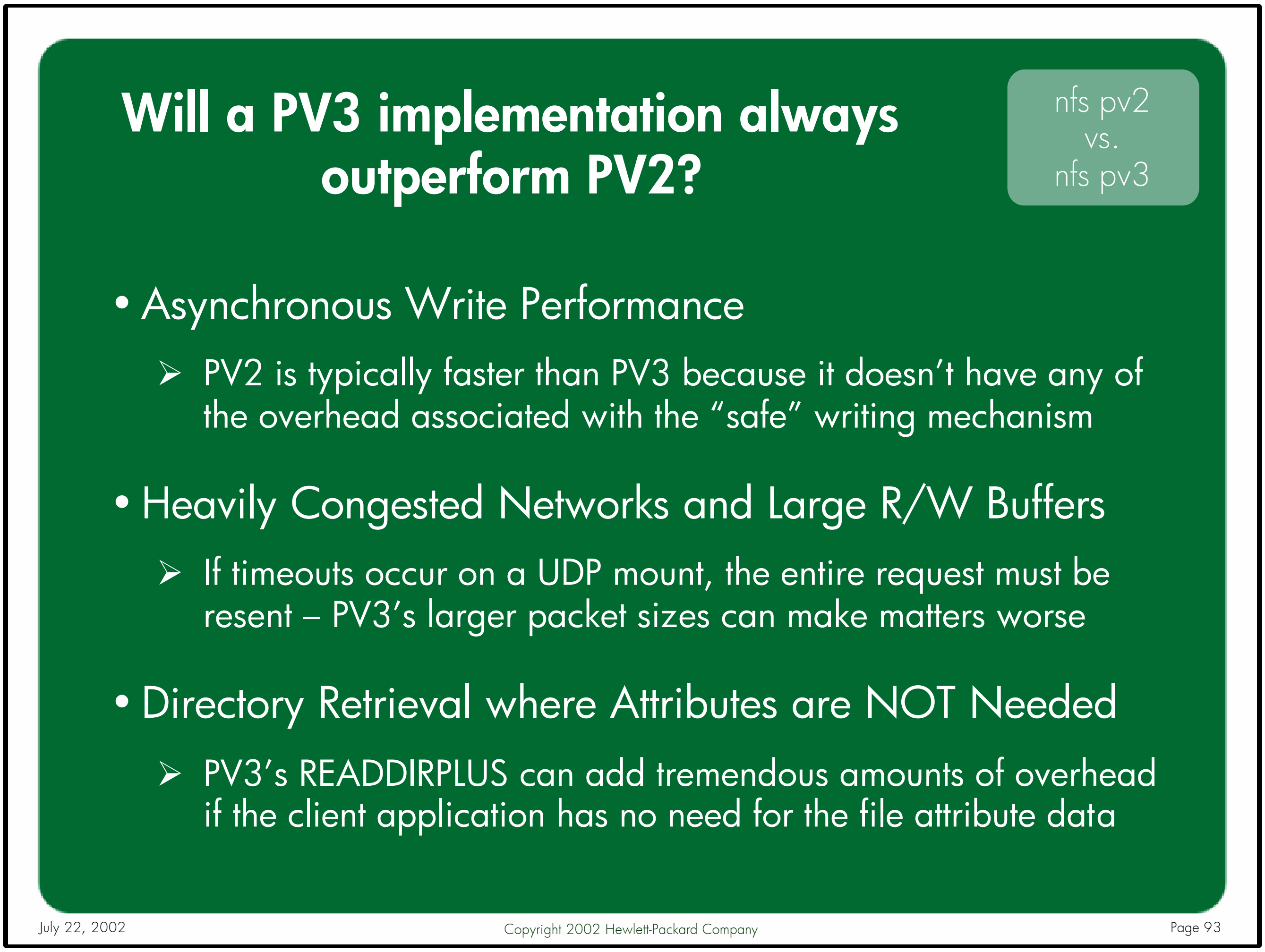
July 22, 2002

Page 92

Notes: The differences between HP-UX 11.0 and 11i that allow an 11i client to access a larger file are not due to variations in the NFS implementations, but rather differences in some of the supporting protocols that NFS is dependent on.

For example, the NLM (Network Lock Manager) protocol shares key kernel variable definitions with the underlying local VxFS filesystem. NLM is therefore subject to the same file size limitations as VxFS. In other words, an HP-UX 11.0 NFS PV3 client may be able to WRITE a file larger than 1TB (assuming it is writing this file to an NFS PV3 server that supports files larger than 1TB), but the client would not be able to set a LOCK on the file at an offset greater than 1TB. Again, this is not a limitation of NFS itself, but of NLM.

Since NFS relies on NLM for file locking semantics, this NLM limitation does impose a limit on the maximum file sizes supported by HP-UX NFS clients.



Will a PV3 implementation always outperform PV2?

nfs pv2
vs.
nfs pv3

- Asynchronous Write Performance
  - ➢ PV2 is typically faster than PV3 because it doesn't have any of the overhead associated with the "safe" writing mechanism
- Heavily Congested Networks and Large R/W Buffers
  - ➢ If timeouts occur on a UDP mount, the entire request must be resent – PV3's larger packet sizes can make matters worse
- Directory Retrieval where Attributes are NOT Needed
  - ➢ PV3's READDIRPLUS can add tremendous amounts of overhead if the client application has no need for the file attribute data

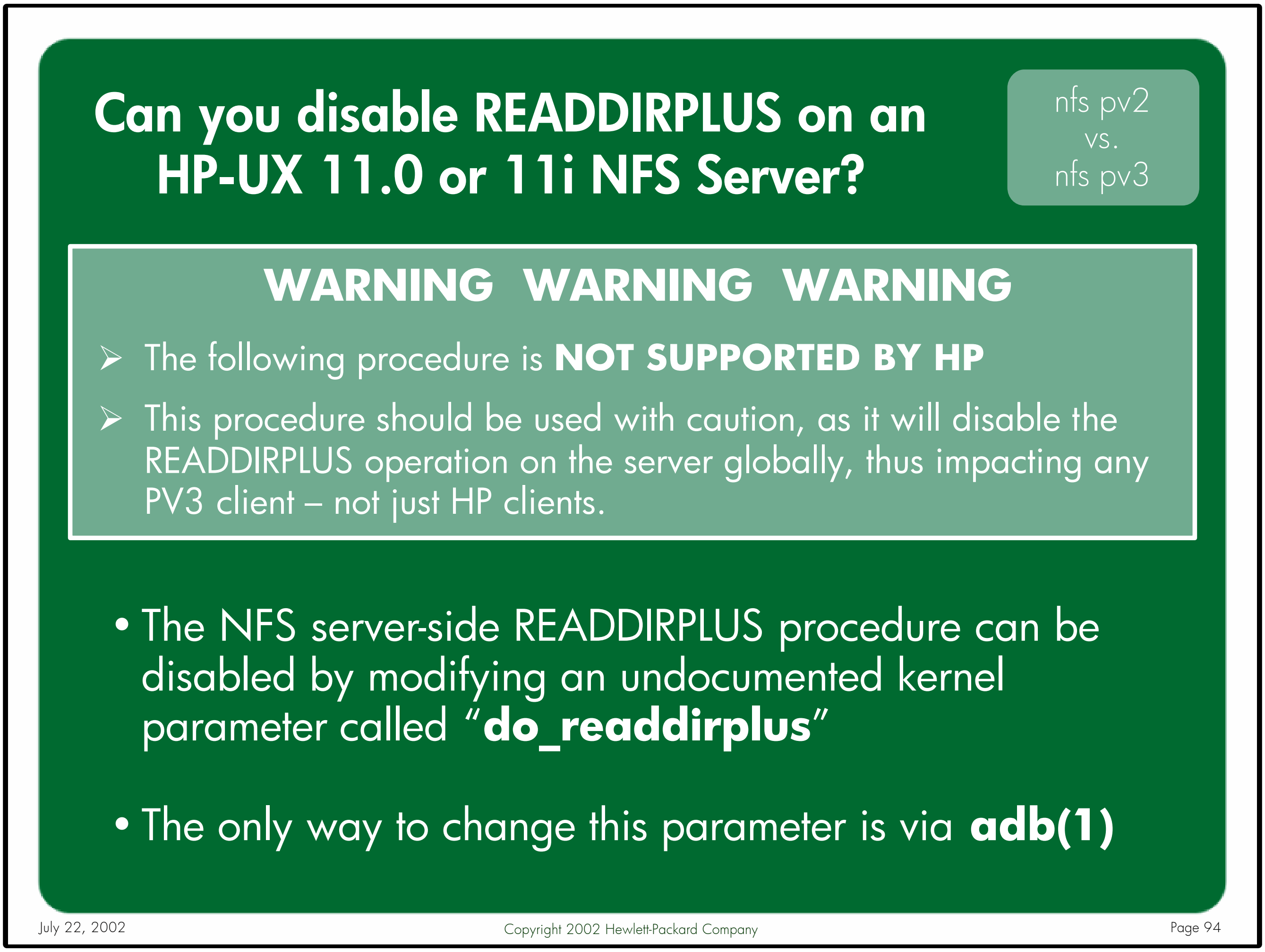
July 22, 2002

Page 93

Notes: While PV3 write performance is comparable to PV2 asynchronous write speed, generally PV2 will be faster simply because it doesn't have any of the overhead associated with the "safe" PV3 write method including: keeping copies of the data in the client's buffer cache, processing write verifiers, etc. However, PV3 write performance is still usually faster than PV2 when larger write buffer sizes are used.

While 32KB read and write buffer sizes generally result in higher NFS performance, there are cases where a smaller buffer size can be beneficial, including: heavily congested networks, wide area networks, and overloaded NFS servers. All three of these cases have the potential for lost packets due to network congestion, network latency, or socket overflows on the NFS server. In UDP environments, larger requests can lead to increased numbers of retransmissions.

READDIRPLUS was added to PV3 to eliminate the need of sending LOOKUP requests for every file in a directory to retrieve its attributes. In large directories this can save hundreds or even thousands of LOOKUP calls. However, if the application doesn't need these attributes, READDIRPLUS adds a tremendous amount of overhead and can cause PV3 clients to perform much worse than PV2 clients.



Can you disable READDIRPLUS on an HP-UX 11.0 or 11i NFS Server?

nfs pv2
vs.
nfs pv3

WARNING WARNING WARNING

- The following procedure is **NOT SUPPORTED BY HP**
- This procedure should be used with caution, as it will disable the READDIRPLUS operation on the server globally, thus impacting any PV3 client – not just HP clients.

- The NFS server-side READDIRPLUS procedure can be disabled by modifying an undocumented kernel parameter called “**do_readdirplus**”
- The only way to change this parameter is via **adb(1)**

July 22, 2002

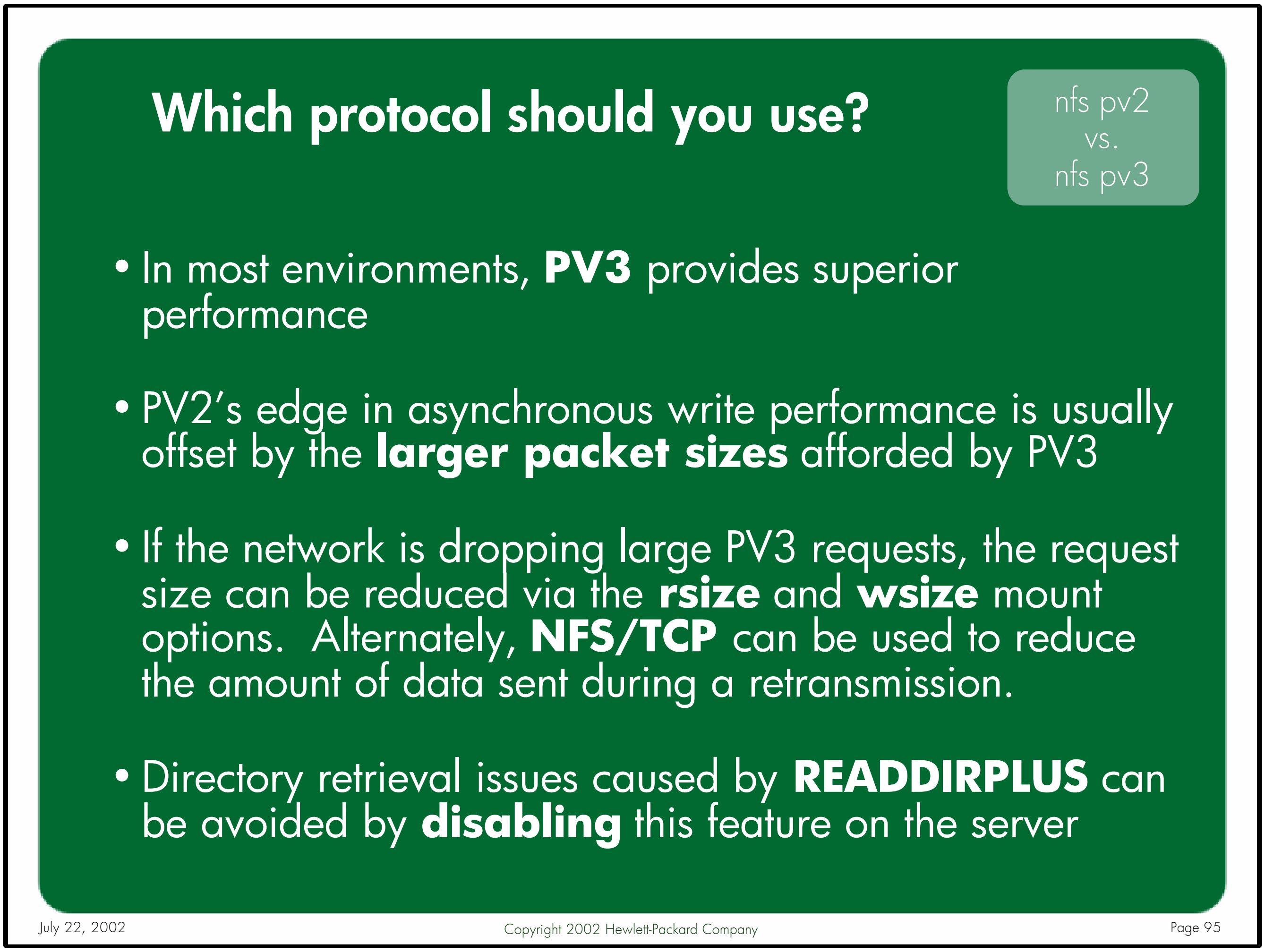
Page 94

Notes: It is important to understand that after using the procedure described below, the 11.0/11i server will respond to any inbound READDIRPLUS request with an error – NFS3ERR_NOTSUPP (operation is not supported). This causes PV3 clients to fall back to the old PV2 method of READDIR/LOOKUP for obtaining directory contents. This procedure is **NOT SUPPORTED BY HP**. Use at your own risk.

To disable READDIRPLUS on an HP-UX 11.0 or 11i server, log into the server as a root user and type:

```
# echo “do_readdirplus/W 0d0” | adb -w /stand/vmunix /dev/kmem
# echo “do_readdirplus?W 0d0” | adb -w /stand/vmunix /dev/kmem
```

The above commands modify both the on-disk kernel file and kernel memory so the change takes effect immediately, and will remain in effect even if the server system is rebooted. These commands would need to be repeated if the kernel is rebuilt, either manually or via a kernel patch installation. To re-enable READDIRPLUS support on the NFS server, substitute “0d1” in place of “0d0” in the above commands.



Which protocol should you use?

nfs pv2
vs.
nfs pv3

- In most environments, **PV3** provides superior performance
- PV2's edge in asynchronous write performance is usually offset by the **larger packet sizes** afforded by PV3
- If the network is dropping large PV3 requests, the request size can be reduced via the **rsize** and **wsize** mount options. Alternately, **NFS/TCP** can be used to reduce the amount of data sent during a retransmission.
- Directory retrieval issues caused by **READDIRPLUS** can be avoided by **disabling** this feature on the server

July 22, 2002

Page 95

Notes: NFS PV3 is a superior protocol to PV2. In nearly every situation a PV3 client/server will outperform a similarly configured PV2 client/server. In those rare occasions where PV2 might have a slight advantage over PV3 (such as asynchronous writing or directory traversal where file attributes are not needed) the PV3 implementation on HP-UX systems can be modified to work around the issue.

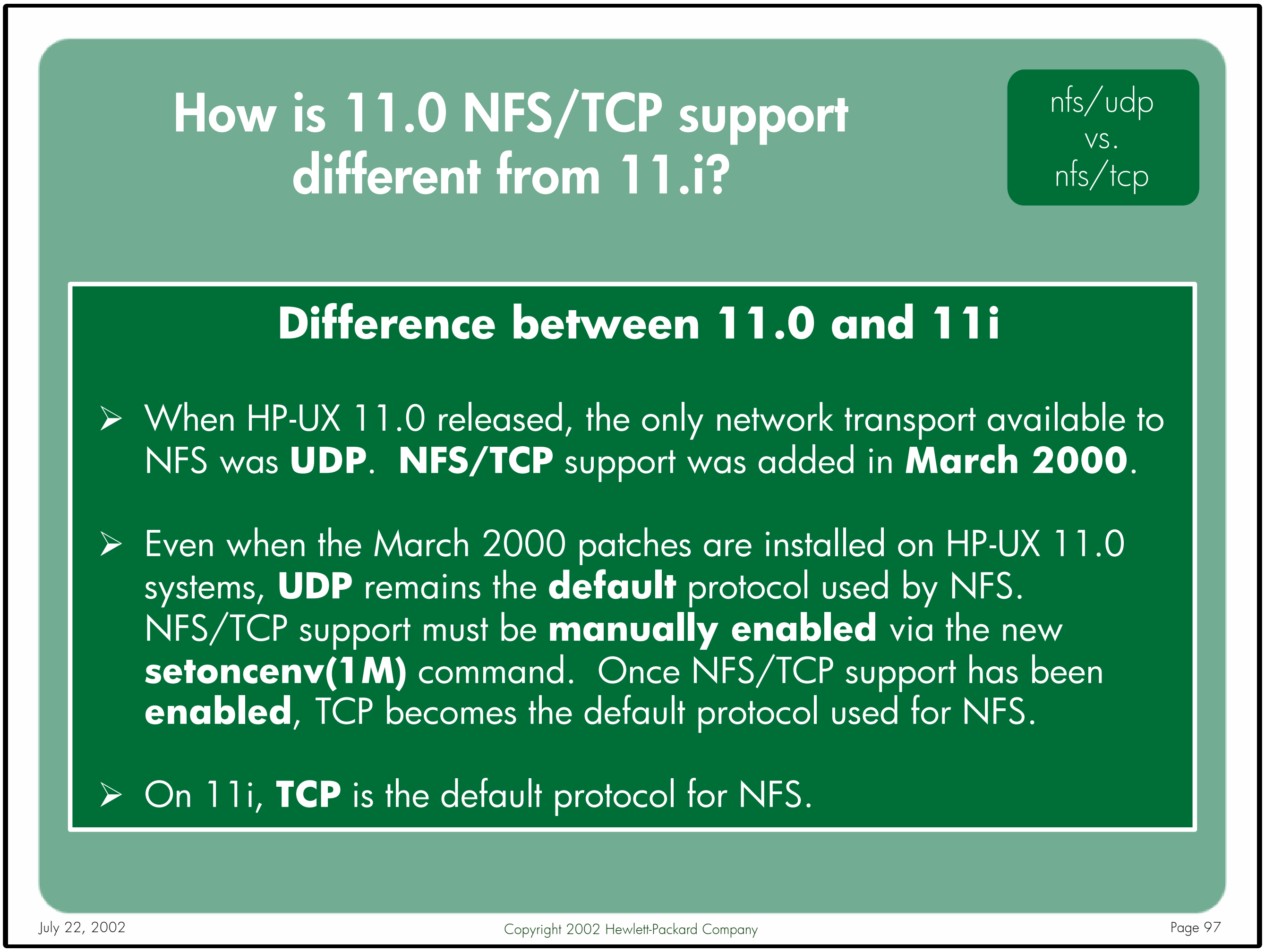
nfs/udp vs. nfs/tcp

- Protocol-Induced Overhead
- Retransmissions and Timeouts
- Network Switch Buffering Considerations
- Should you use NFS/UDP or NFS/TCP in your environment?

Notes: One of the design goals of the original version of NFS, and every version since, has been to minimize the amount of network latency involved in accessing remote files. For this reason NFS was originally designed to run exclusively over the UDP network transport, as this transport provided a lightweight delivery mechanism.

Over time, as clients and servers became geographically dispersed across wide area networks, it became necessary to provide guaranteed data delivery and improved handling of network timeouts and retransmissions – even if the added overhead meant slower performance in local area networks. The TCP/IP protocol provided these benefits and was already widely in use by other networking applications. The developers of NFS therefore decided to modify the NFS protocol to allow it to run over either TCP or UDP and allow the system administrators to decide which transport mechanism better met their needs on a per-mount basis.

Now that both UDP and TCP are supported protocols, the question is 'Which one should I use?' To answer this question, a system administrator needs to understand the differences between the two transport mechanisms and how these differences can benefit or hinder NFS performance in their environment.



How is 11.0 NFS/TCP support different from 11.i?

nfs/udp
vs.
nfs/tcp

Difference between 11.0 and 11i

- When HP-UX 11.0 released, the only network transport available to NFS was **UDP**. **NFS/TCP** support was added in **March 2000**.
- Even when the March 2000 patches are installed on HP-UX 11.0 systems, **UDP** remains the **default** protocol used by NFS. NFS/TCP support must be **manually enabled** via the new **setoncenv(1M)** command. Once NFS/TCP support has been **enabled**, TCP becomes the default protocol used for NFS.
- On 11i, **TCP** is the default protocol for NFS.

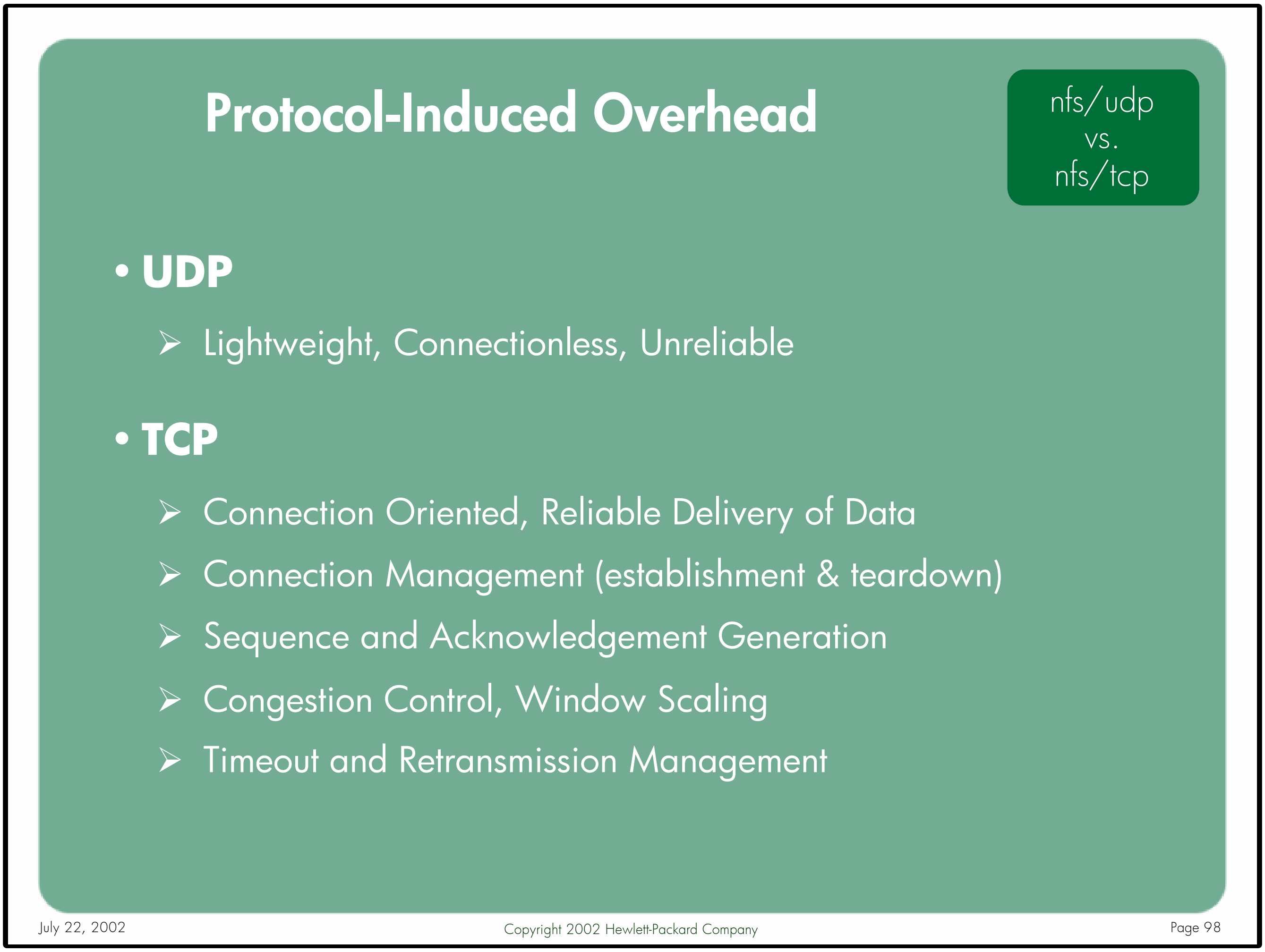
July 22, 2002

Page 97

Notes: When HP-UX 11.0 shipped it only supported NFS running over UDP. HP added support for NFS/TCP via the kernel and user-space NFS patches released in March 2000. Any HP-UX 11.0 NFS patches released after March 2000 also contains this functionality. Since new NFS patches are released periodically, be sure to consult HP Support or the IT Resource Center web site – **http://itrc.hp.com** – to obtain information about currently available NFS patches.

After installing these patches on both the client and server (assuming both systems run HP-UX 11.0), the system administrator must enable NFS/TCP functionality manually via a new command called setoncenv(1M). Again, this must be done on both the NFS client and the server. Once enabled, TCP becomes the default network transport for new NFS mounts.

On HP-UX 11i, TCP is the default transport used for NFS mounts.



Protocol-Induced Overhead

nfs/udp
vs.
nfs/tcp

- **UDP**
  - Lightweight, Connectionless, Unreliable
- **TCP**
  - Connection Oriented, Reliable Delivery of Data
  - Connection Management (establishment & teardown)
  - Sequence and Acknowledgement Generation
  - Congestion Control, Window Scaling
  - Timeout and Retransmission Management

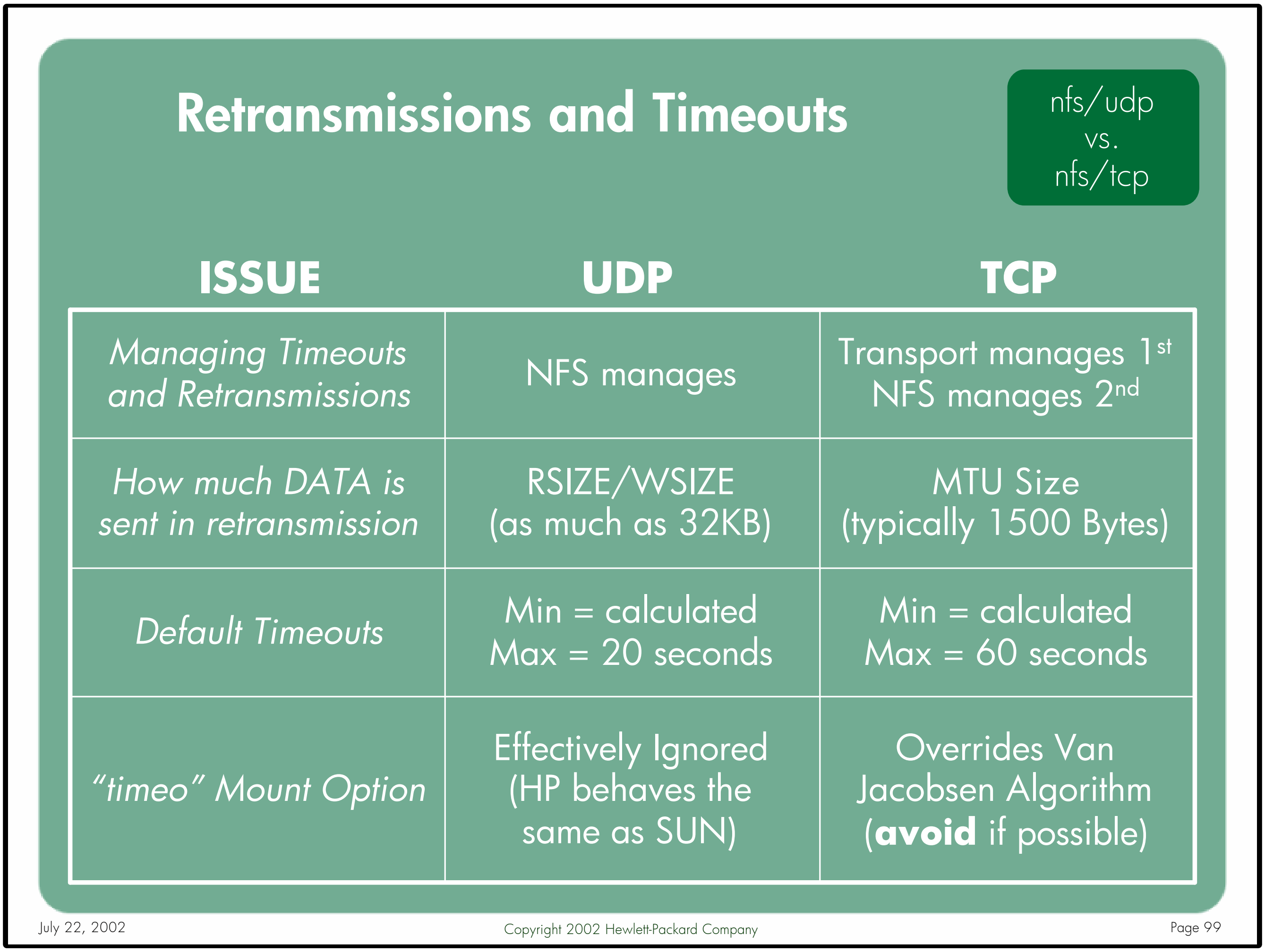
July 22, 2002

Page 98

Notes: UDP is commonly described as a connectionless or unreliable transport protocol. It is a very lightweight protocol, which is designed to quickly and efficiently deliver IP datagrams between systems. By contrast, TCP is a connection-oriented transport, which maintains established connections between systems and guarantees reliable delivery of data.

The reliability provided by TCP does not come without some overhead. There is connection management overhead (i.e. connection establishment and teardown), tracking of sequence and acknowledgement information, congestion control, error recovery, etc. In LAN environments, where latency is low and retransmissions are few, the overhead imposed by TCP can hurt overall NFS performance.

How much does the overhead of TCP affect NFS performance? As with most performance related questions the answer is 'It depends'. Many factors can influence the behavior and performance of UDP and TCP in a given network. It is therefore difficult to predict how much impact converting from UDP to TCP will have in an existing NFS installation. Thorough testing with both protocols is highly recommended.



Retransmissions and Timeouts

nfs/udp
vs.
nfs/tcp

ISSUE	UDP	TCP
Managing Timeouts and Retransmissions	NFS manages	Transport manages 1st NFS manages 2nd
How much DATA is sent in retransmission	RSIZE/WSIZE (as much as 32KB)	MTU Size (typically 1500 Bytes)

Default Timeouts	Min = calculated
	Max = 20 seconds
		Min = calculated
		Max = 60 seconds

"timeo" Mount Option	Effectively Ignored (HP behaves the same as SUN)	Overrides Van Jacobsen Algorithm (**avoid** if possible)

Notes:

Managing Timeouts and Retransmissions

Since the UDP transport is unreliable, it is up to the application (NFS) to manage retransmitting packets that are not replied to. TCP/IP has built in mechanisms for managing the delivery of data. Only in extreme situations does NFS get involved in retransmitting data on a TCP mount.

How much DATA is sent in retransmission

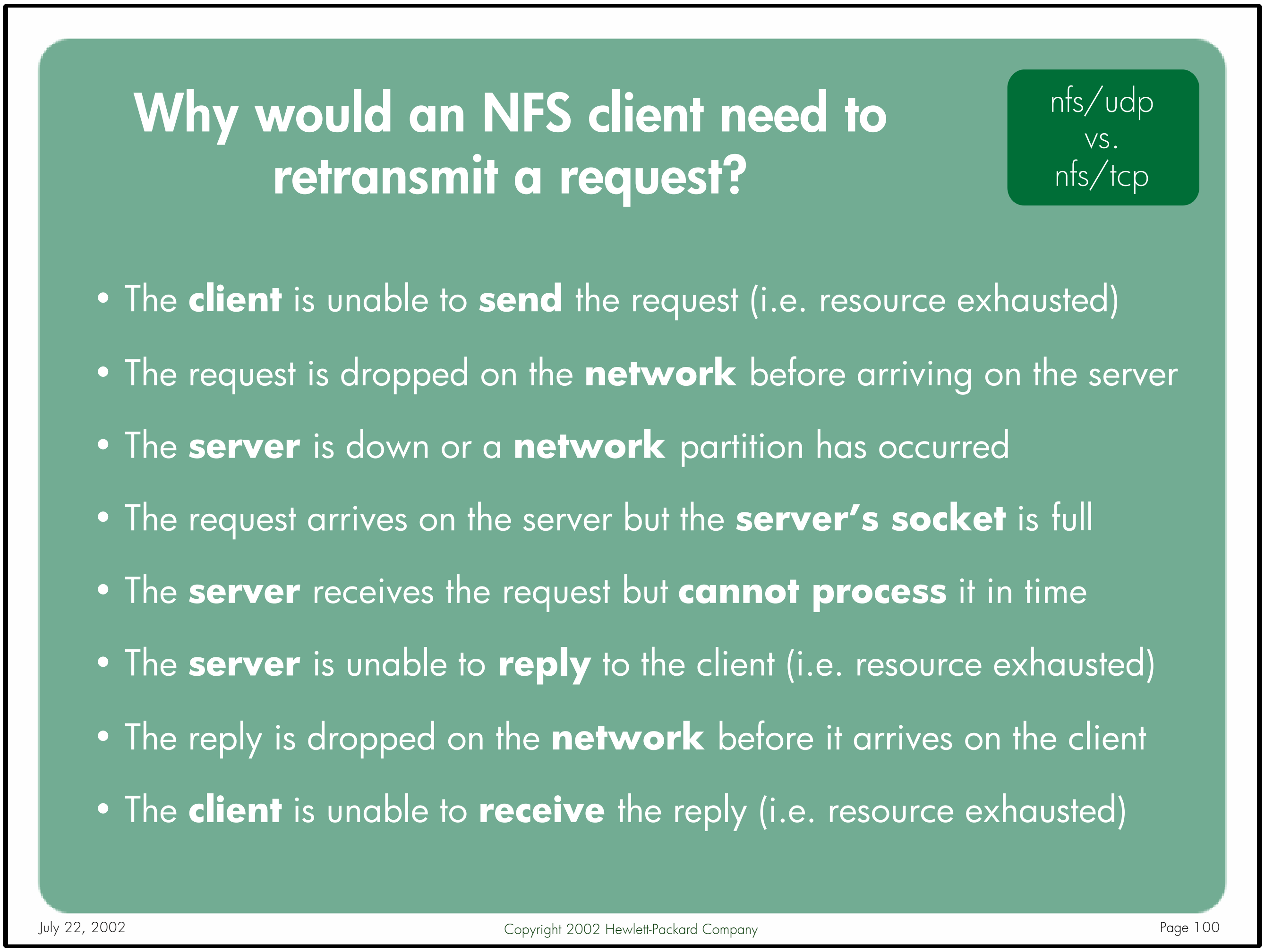
When a retransmission occurs on a UDP mount NFS must resend the entire request, which could be as much as 32KB of data – even if only a single MTU of data was lost. TCP keeps track of how much data has been sent by the clients and received by the servers. Therefore, if a single MTU-sized packet of data is lost on the network, only that portion must be retransmitted.

Default Timeouts

Both UDP and TCP use the Van Jacobsen algorithm to calculate timeout values based on the smooth round trip timers (the current srtt values can be displayed on a per-mount basis by issuing the "nfsstat –m" command). UDP has a maximum timeout value of 20 seconds, while NFS allows TCP to retry for 60 seconds before it forces a retransmission of the entire request.

"timeo" Mount Option

The "timeo" mount option is effectively ignored on UDP mounts. The use of this option on TCP-based filesystems is highly discouraged as it overrides the Van Jacobsen algorithm, potentially resulting in very poor performance, and causing "`NFS server not responding`" errors.



Why would an NFS client need to retransmit a request?

nfs/udp
vs.
nfs/tcp

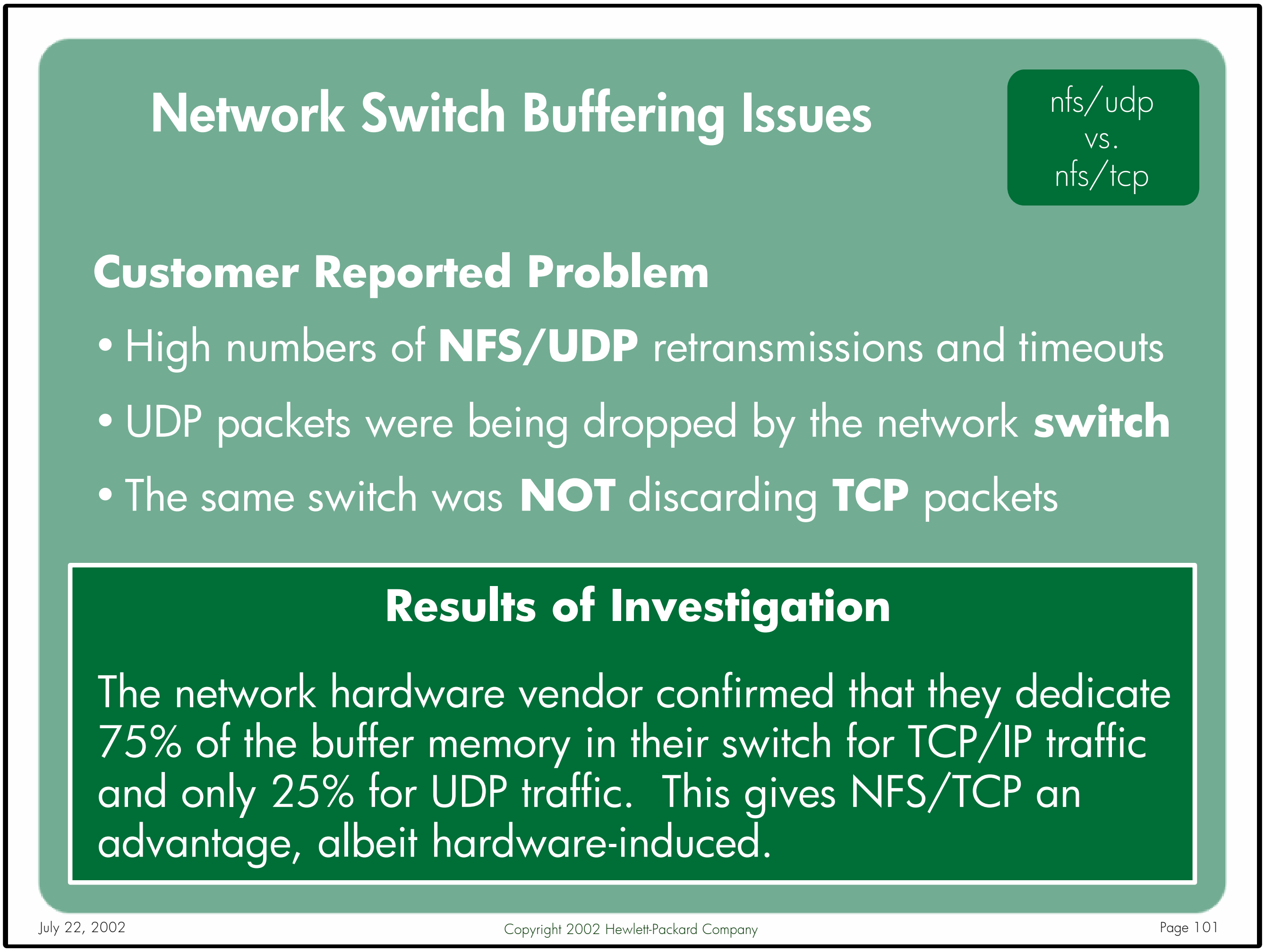
- The **client** is unable to **send** the request (i.e. resource exhausted)
- The request is dropped on the **network** before arriving on the server
- The **server** is down or a **network** partition has occurred
- The request arrives on the server but the **server's socket** is full
- The **server** receives the request but **cannot process** it in time
- The **server** is unable to **reply** to the client (i.e. resource exhausted)
- The reply is dropped on the **network** before it arrives on the client
- The **client** is unable to **receive** the reply (i.e. resource exhausted)

July 22, 2002

Page 100

Notes: Because NFS was originally designed to only use UDP as a network transport, and because of the semantics of UDP (i.e. unreliable, connectionless), NFS was designed with its own built-in mechanisms to guarantee data integrity. The client keeps track of the requests it sends and waits for a response from the server before generating new requests. If the client does not receive a response from the server within a given time period, it considers the original request lost and reissues the request. The number of times the client will retry the request depends upon whether the NFS mount uses hard-mount or soft-mount semantics, where hard mounts will continually retry the request until a response is received, and soft mounts will eventually time out the request and return control (and any error) to the application.

There are many reasons an NFS client would be forced to resend data. The most common scenarios are listed above. The point of this list is to illustrate the number of places in a typical NFS client/server relationship where a request or reply can be lost, necessitating a retransmission.



Network Switch Buffering Issues

nfs/udp
vs.
nfs/tcp

Customer Reported Problem

- High numbers of **NFS/UDP** retransmissions and timeouts
- UDP packets were being dropped by the network **switch**
- The same switch was **NOT** discarding **TCP** packets

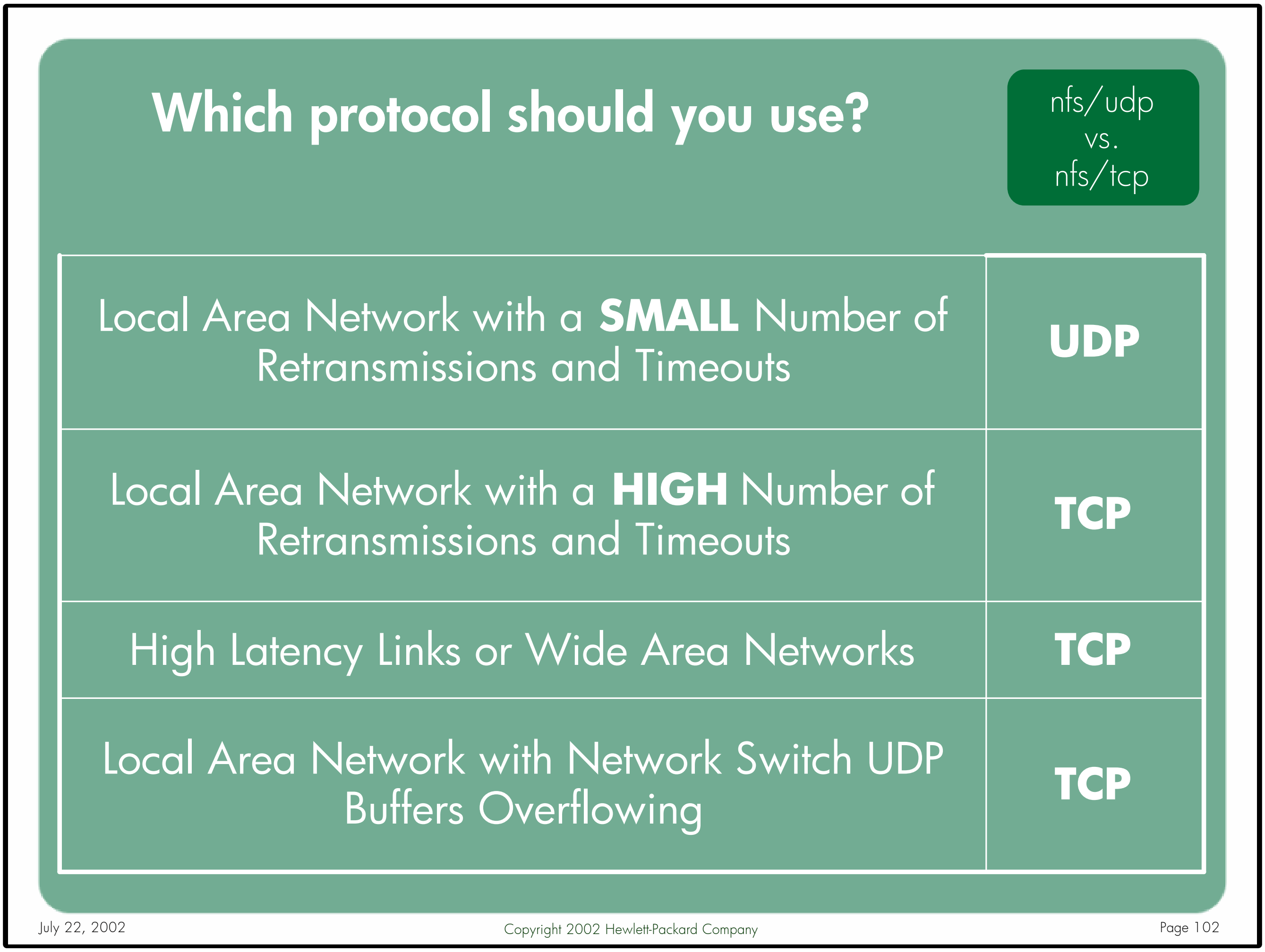
Results of Investigation

The network hardware vendor confirmed that they dedicate 75% of the buffer memory in their switch for TCP/IP traffic and only 25% for UDP traffic. This gives NFS/TCP an advantage, albeit hardware-induced.

Notes: In some of HP's largest NFS customer installations, we've seen cases where high numbers of NFS/UDP retransmissions and timeouts were occurring, resulting in relatively poor read and write performance.

After much investigation, it was determined that UDP packets were being discarded by the network switch used by the customer. When the switch vendor investigated the log files on the switch they discovered that the UDP buffers in the switch were overflowing which led to the dropped packets. When asked why the switch was not also dropping TCP packets the vendor explained that they design their switches with 75% of the buffer memory dedicated for TCP/IP traffic and only 25% for UDP. The switch therefore had sufficient memory to keep up with the TCP traffic but not enough to handle the UDP load.

It is likely that many network hardware vendors configure their switch buffer memory in this manner, so this problem is not confined to large-scale implementations using a specific vendor's switch. In these environments NFS/TCP has an advantage over UDP, albeit hardware-induced.



Which protocol should you use?

nfs/udp
vs.
nfs/tcp

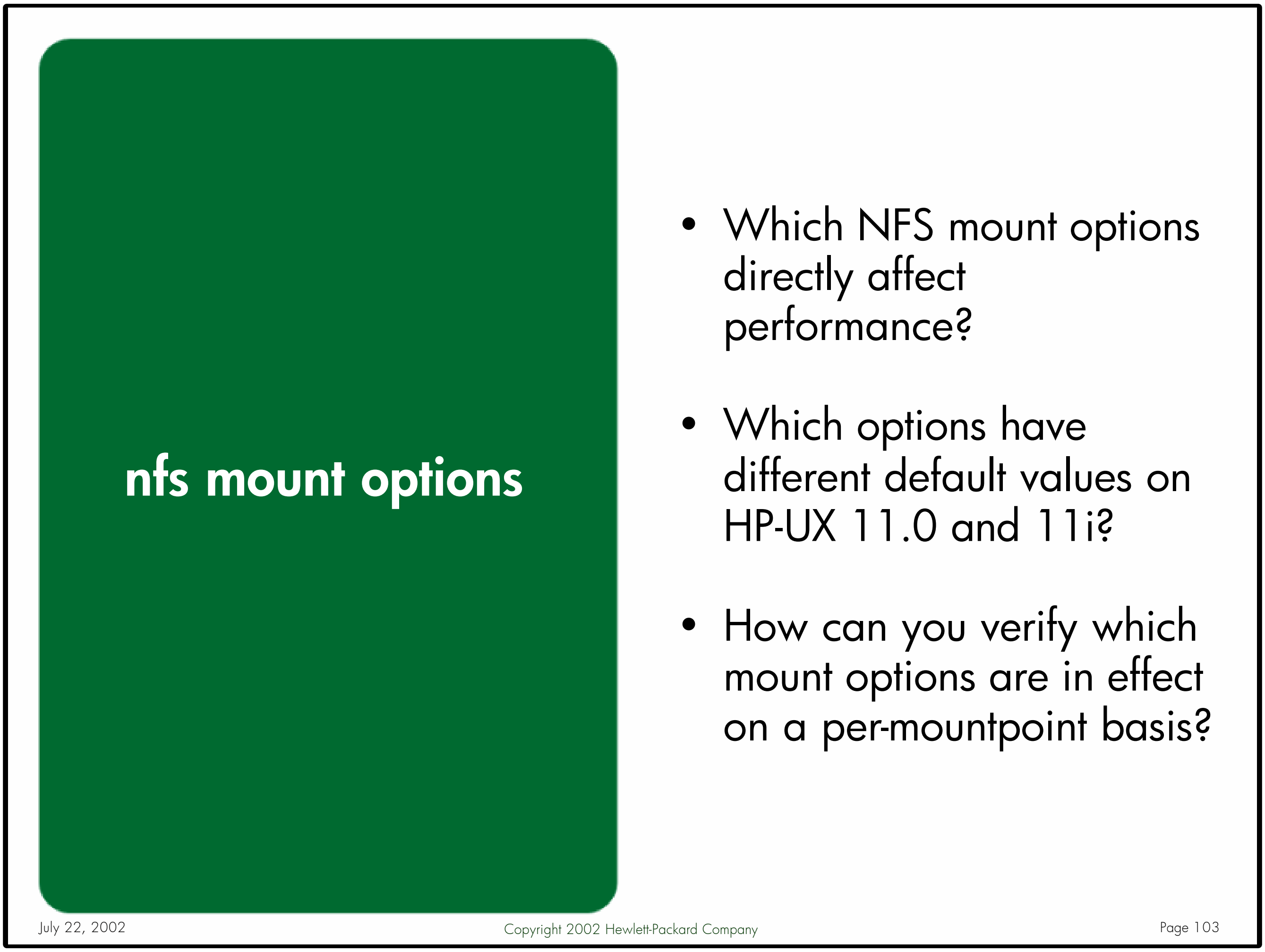
Local Area Network with a **SMALL** Number of Retransmissions and Timeouts	**UDP**
Local Area Network with a **HIGH** Number of Retransmissions and Timeouts	**TCP**
High Latency Links or Wide Area Networks	**TCP**
Local Area Network with Network Switch UDP Buffers Overflowing	**TCP**

July 22, 2002

Page 102

Notes: Traditionally the decision to use UDP or TCP was based solely on geography (i.e. LAN=UDP, WAN=TCP). However, there are situations where even a LAN environment could benefit from TCP.

As with most every performance-related recommendation, your mileage may vary. Even in the scenarios listed above, where one protocol would seem to have a clear advantage over the other, there is no guarantee that the recommendation will hold true in every environment. If possible, both protocols should be tested and evaluated before making a final decision.



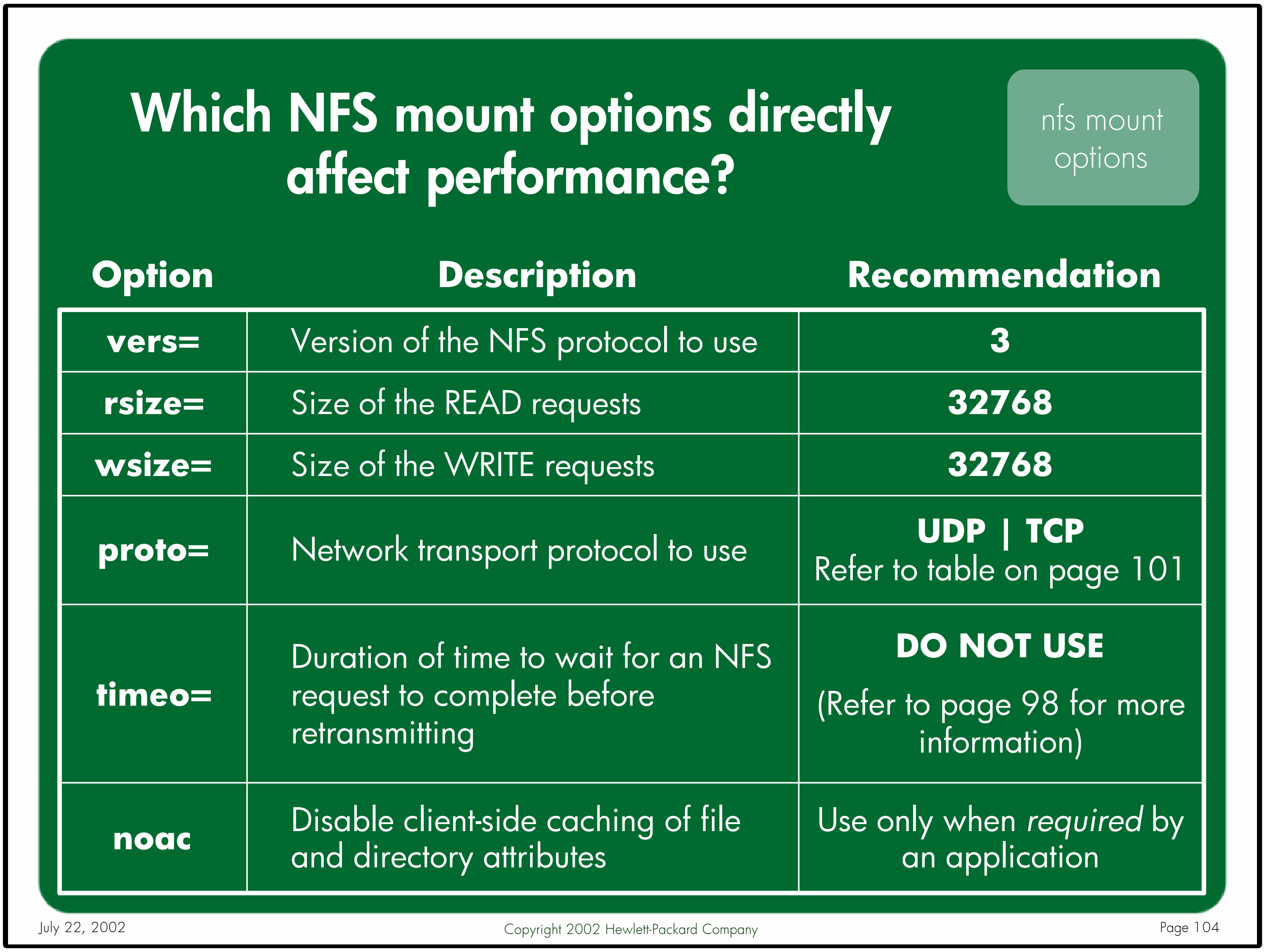
nfs mount options

- Which NFS mount options directly affect performance?
- Which options have different default values on HP-UX 11.0 and 11i?
- How can you verify which mount options are in effect on a per-mountpoint basis?

July 22, 2002

Page 103

Notes: There are many NFS-specific mount options available. Some of these options can have a positive impact on performance, while others can have a dramatically negative effect. It is important to know which options to use and which to avoid.



Which NFS mount options directly affect performance?

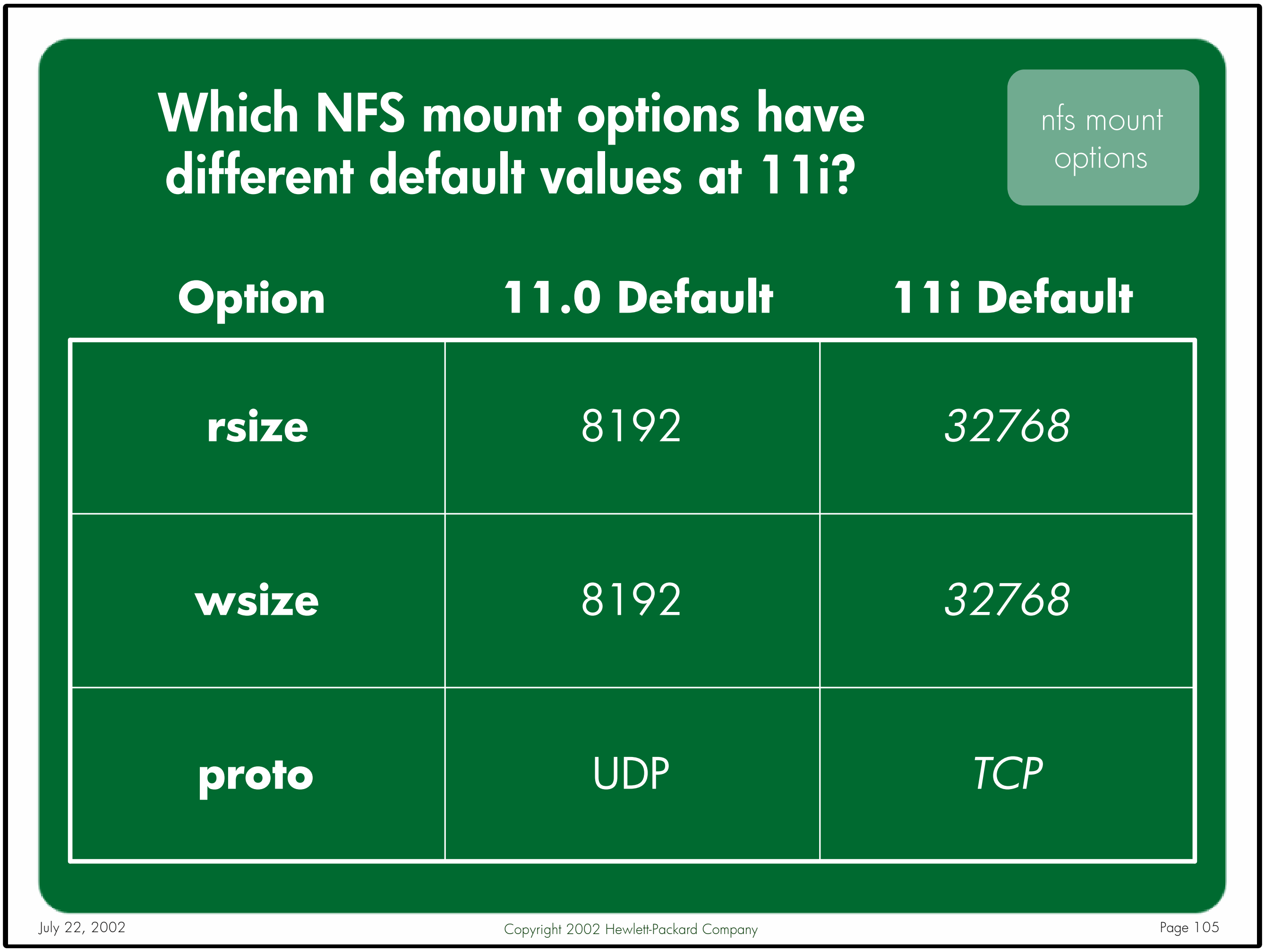
nfs mount options

Option	Description	Recommendation
vers=	Version of the NFS protocol to use	**3**
rsize=	Size of the READ requests	**32768**
wsize=	Size of the WRITE requests	**32768**
proto=	Network transport protocol to use	**UDP | TCP** Refer to table on page 101
timeo=	Duration of time to wait for an NFS request to complete before retransmitting	**DO NOT USE** (Refer to page 98 for more information)
noac	Disable client-side caching of file and directory attributes	Use only when *required* by an application

July 22, 2002

Page 104

Notes: While the above table is by no means an exhaustive list of the available options to the NFS mount command, these are the most commonly used options that can have a significant impact on performance. For more information about the available NFS mount options refer to the mount_nfs(1M) man page.



Which NFS mount options have different default values at 11i?

nfs mount options

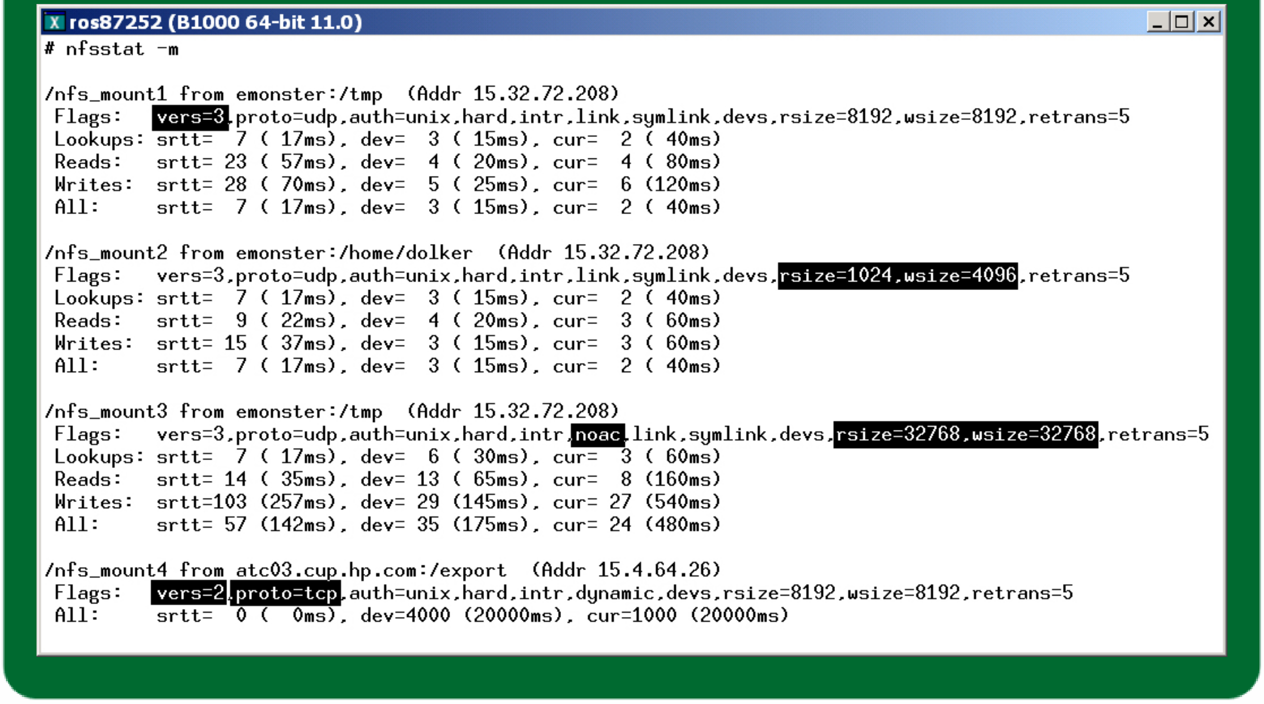
Option	11.0 Default	11i Default
rsize	8192	*32768*
wsize	8192	*32768*
proto	UDP	*TCP*

July 22, 2002

Notes: The default values of many NFS mount options have changed in 11i. It is therefore important to understand which options have changed to know how a default NFS mount (i.e. a mount where no options are specified) will behave on both 11.0 and 11i clients. Some of the defaults were changed for performance reasons, and others to make HP's NFS implementation more compatible with Sun Microsystems' NFS implementation.

How can you verify which NFS mount options are being used?

nfs mount options

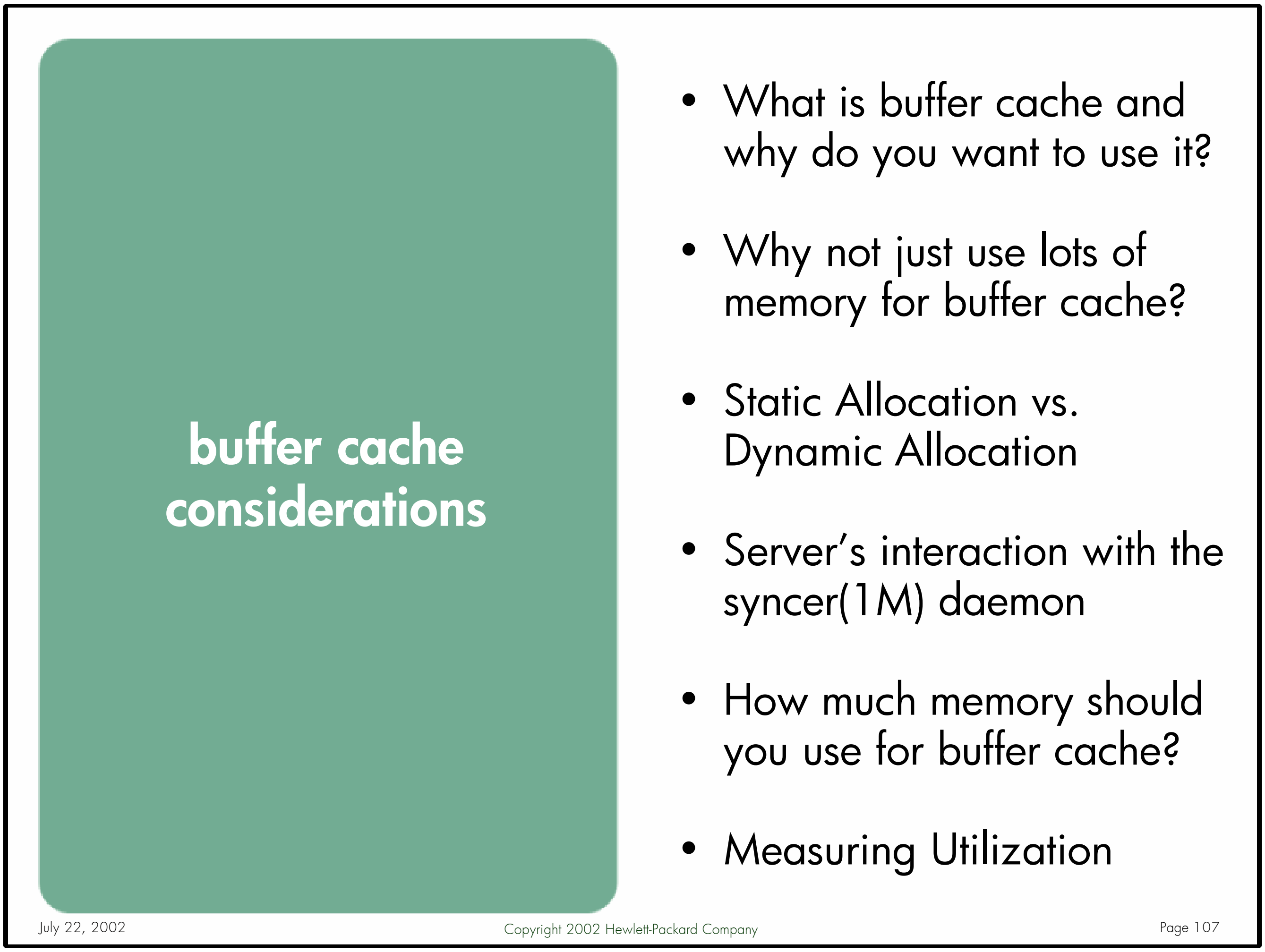


```
# nfsstat -m

/nfs_mount1 from emonster:/tmp  (Addr 15.32.72.208)
 Flags:   vers=3,proto=udp,auth=unix,hard,intr,link,symlink,devs,rsize=8192,wsize=8192,retrans=5
 Lookups: srtt=  7 ( 17ms), dev=  3 ( 15ms), cur=  2 ( 40ms)
 Reads:   srtt= 23 ( 57ms), dev=  4 ( 20ms), cur=  4 ( 80ms)
 Writes:  srtt= 28 ( 70ms), dev=  5 ( 25ms), cur=  6 (120ms)
 All:     srtt=  7 ( 17ms), dev=  3 ( 15ms), cur=  2 ( 40ms)

/nfs_mount2 from emonster:/home/dolker  (Addr 15.32.72.208)
 Flags:   vers=3,proto=udp,auth=unix,hard,intr,link,symlink,devs,rsize=1024,wsize=4096,retrans=5
 Lookups: srtt=  7 ( 17ms), dev=  3 ( 15ms), cur=  2 ( 40ms)
 Reads:   srtt=  9 ( 22ms), dev=  4 ( 20ms), cur=  3 ( 60ms)
 Writes:  srtt= 15 ( 37ms), dev=  3 ( 15ms), cur=  3 ( 60ms)
 All:     srtt=  7 ( 17ms), dev=  3 ( 15ms), cur=  2 ( 40ms)

/nfs_mount3 from emonster:/tmp  (Addr 15.32.72.208)
 Flags:   vers=3,proto=udp,auth=unix,hard,intr,noac,link,symlink,devs,rsize=32768,wsize=32768,retrans=5
 Lookups: srtt=  7 ( 17ms), dev=  6 ( 30ms), cur=  3 ( 60ms)
 Reads:   srtt= 14 ( 35ms), dev= 13 ( 65ms), cur=  8 (160ms)
 Writes:  srtt=103 (257ms), dev= 29 (145ms), cur= 27 (540ms)
 All:     srtt= 57 (142ms), dev= 35 (175ms), cur= 24 (480ms)

/nfs_mount4 from atc03.cup.hp.com:/export  (Addr 15.4.64.26)
 Flags:   vers=2,proto=tcp,auth=unix,hard,intr,dynamic,devs,rsize=8192,wsize=8192,retrans=5
 All:     srtt=  0 (  0ms), dev=4000 (20000ms), cur=1000 (20000ms)
```

Notes: The easiest and most accurate way to determine which NFS mount options are in effect on a per-mountpoint basis is to use the "nfsstat -m" command.

Looking at the above screenshot, we can determine several things about the way this client has mounted its filesystems. For example, three of the four filesystems are mounted using NFS PV3 – only `/nfs_mount4` is using PV2. Also, all filesystems but `/nfs_mount4` are mounted from NFS server "emonster". We can see that `/nfs_mount3` is mounted with the "noac" option and is using read and write buffer sizes of 32KB. We see that `/nfs_mount2` is using a read size of 1KB and a write size of 4KB. Also, notice that 3 of the 4 filesystems use UDP as their underlying transport, only `/nfs_mount4` uses TCP. Finally, we see that `/nfs_mount1` and `/nfs_mount3` reference the same remote filesystem from the same NFS server, but they are mounted on the client with different options.

Notice that the "nfsstat –m" output also includes the smooth round trip timers maintained for the various NFS call types (Lookups, Reads, Writes) on NFS/UDP mounts. This information can be very useful, especially when trying to determine why one NFS mounted filesystem is performing better or worse than another.



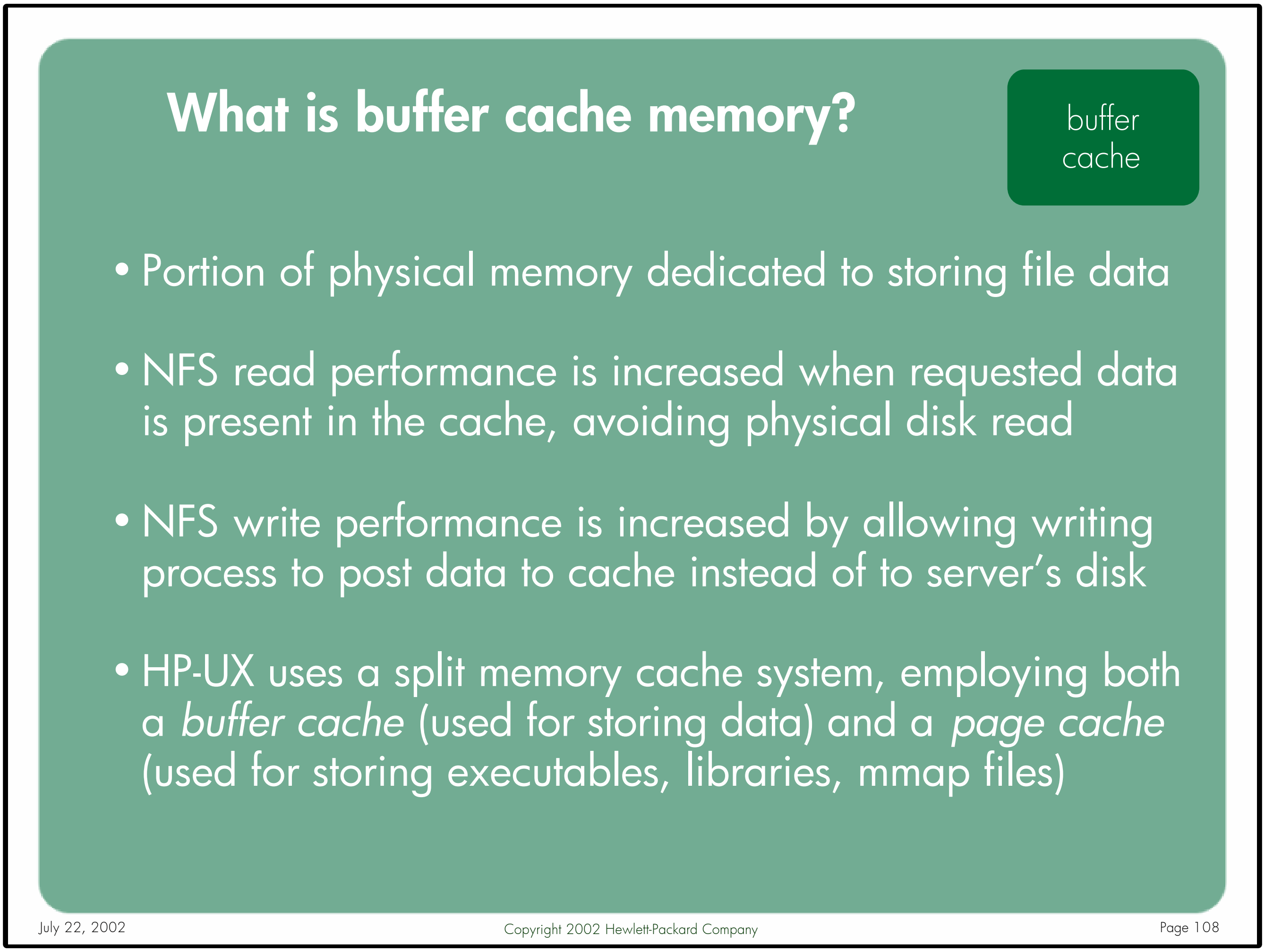
buffer cache considerations

- What is buffer cache and why do you want to use it?
- Why not just use lots of memory for buffer cache?
- Static Allocation vs. Dynamic Allocation
- Server's interaction with the syncer(1M) daemon
- How much memory should you use for buffer cache?
- Measuring Utilization

July 22, 2002 Copyright 2002 Hewlett-Packard Company Page 107

Notes: Sizing buffer cache correctly on NFS clients and servers can be a time consuming endeavor, but one that can dramatically affect both NFS and overall system performance. To understand the intricacies and factors involved in correctly sizing buffer cache, some understanding of what buffer cache memory is, how it is managed, and how it is searched is needed.

This section describes what buffer cache memory is and why it is so important to NFS performance. The differences between static and dynamic buffer cache allocation methods are explained, along with the reasons for selecting one method over the other. The NFS server's interaction with the syncer(1M) daemon is explained, followed by recommendations for determining the appropriate amount of buffer cache memory to use on a given NFS client or server. Finally, this section describes the tools available for measuring buffer cache utilization.



What is buffer cache memory?

buffer cache

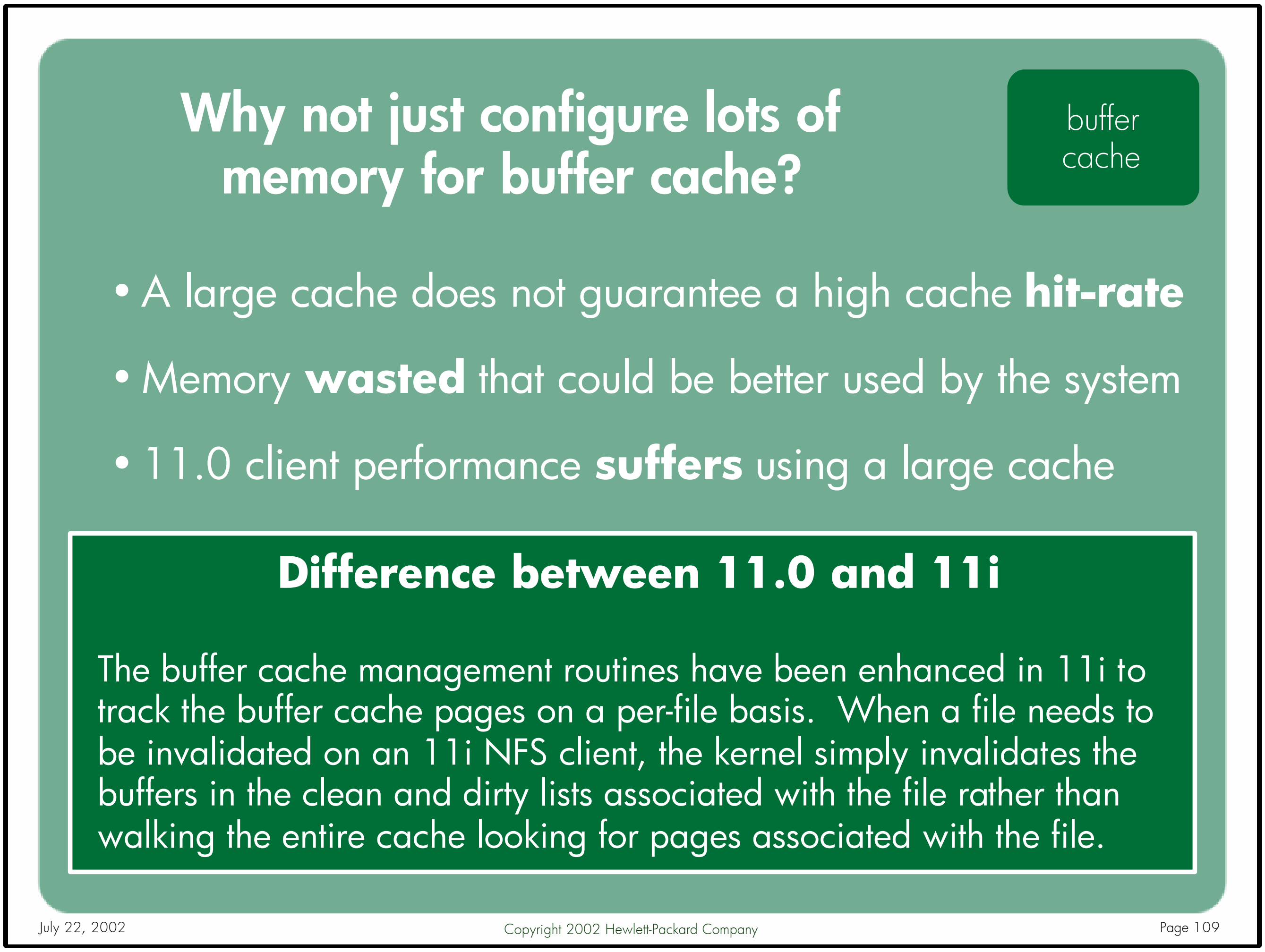
- Portion of physical memory dedicated to storing file data
- NFS read performance is increased when requested data is present in the cache, avoiding physical disk read
- NFS write performance is increased by allowing writing process to post data to cache instead of to server's disk
- HP-UX uses a split memory cache system, employing both a *buffer cache* (used for storing data) and a *page cache* (used for storing executables, libraries, mmap files)

July 22, 2002

Page 108

Notes: Buffer cache is a portion of physical memory that is allocated for storing blocks of file data. HP-UX uses this memory to speed up file operations such as read() and write(). Since memory access times are so much faster than disk access times, generally the more file system requests that can be satisfied by buffer cache the better overall I/O performance will be. NFS read operations can be satisfied without waiting for a physical disk I/O if the requested data is already residing in the cache. NFS write performance can be dramatically increased by allowing a writing process to post the data to local buffer cache and continue, letting the system migrate the pages to the server's physical disk in the background.

Unlike most vendor's UNIX implementations, HP-UX currently uses a split memory cache system where some memory pages are cached in the buffer cache and others in the page cache. While the buffer cache is used to store data pages, the page cache is used when pages are brought into or pushed out from memory using the paging interface, either explicitly by mapping a file into memory through a call to mmap(2), or implicitly for objects such as executable programs or shared libraries which are mapped into memory on behalf of a running program.



Why not just configure lots of memory for buffer cache?

buffer cache

- A large cache does not guarantee a high cache **hit-rate**
- Memory **wasted** that could be better used by the system
- 11.0 client performance **suffers** using a large cache

Difference between 11.0 and 11i

The buffer cache management routines have been enhanced in 11i to track the buffer cache pages on a per-file basis. When a file needs to be invalidated on an 11i NFS client, the kernel simply invalidates the buffers in the clean and dirty lists associated with the file rather than walking the entire cache looking for pages associated with the file.

July 22, 2002

Page 109

Notes: **A large cache does not guarantee a high cache hit rate**

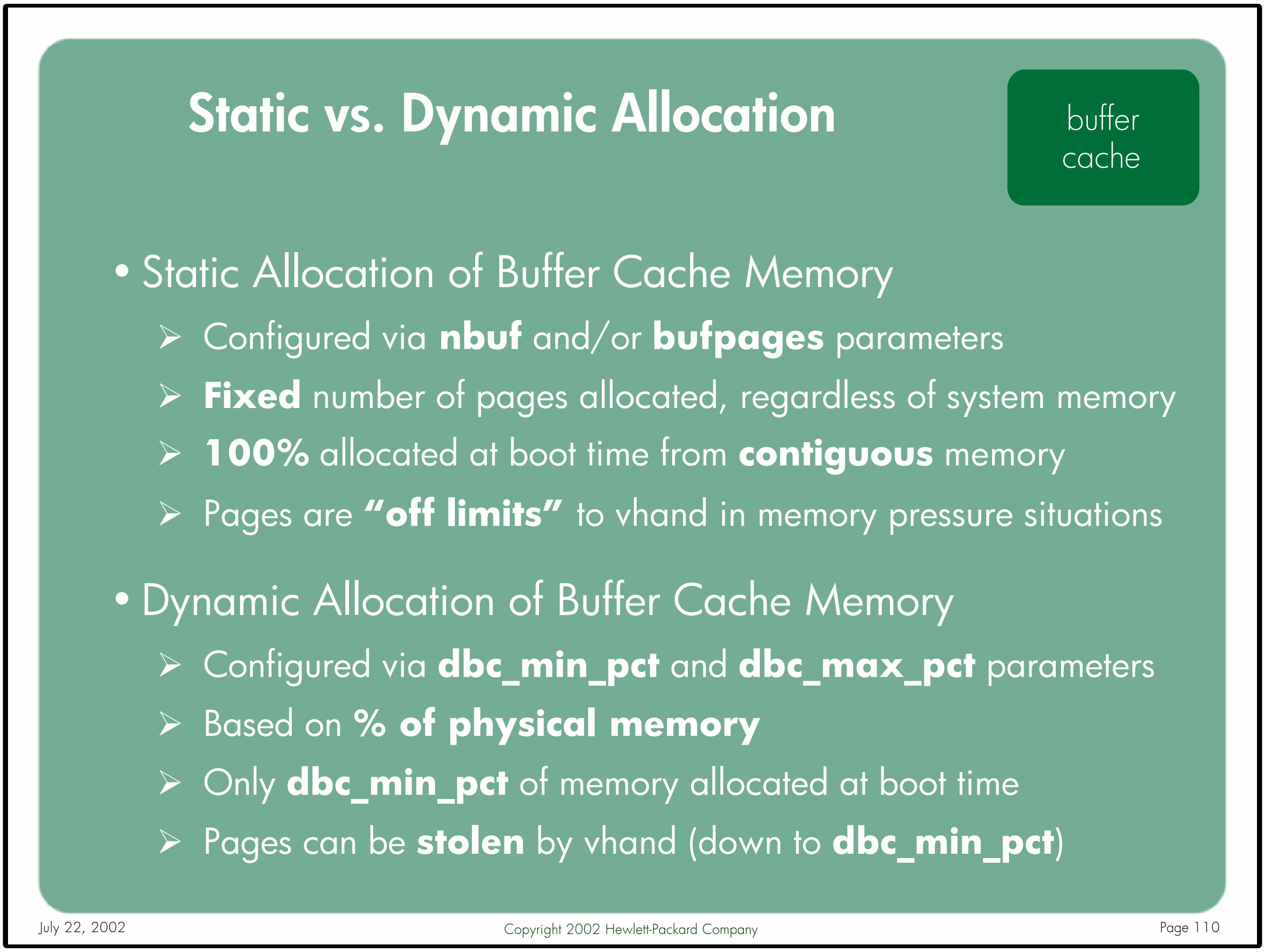
The buffer cache can only satisfy an NFS read request if the data is present in the cache, which would imply that it had been retrieved from the server prior to the client process requesting it. Since biods pre-fetch sequential data, if the client is reading non-sequentially there is a strong probability that the data will not be resident in the cache, regardless of how big the buffer cache is.

Memory wasted that could be better used by the system

By reserving physical memory for buffer cache (particularly in the static configuration, where the memory is reserved at boot time), you are effectively removing this memory from the pool available for user-space processes.

11.0 client performance suffers using a large buffer cache

The most compelling reason to not use a large buffer cache on an 11.0 NFS client is that doing so will actually result in much worse performance than using a smaller cache. This is because buffer cache pages are not tracked on a file-by-file basis in 11.0, forcing the kernel to perform a linear search of the entire cache when it needs to invalidate a file. An example is provided on page 14.



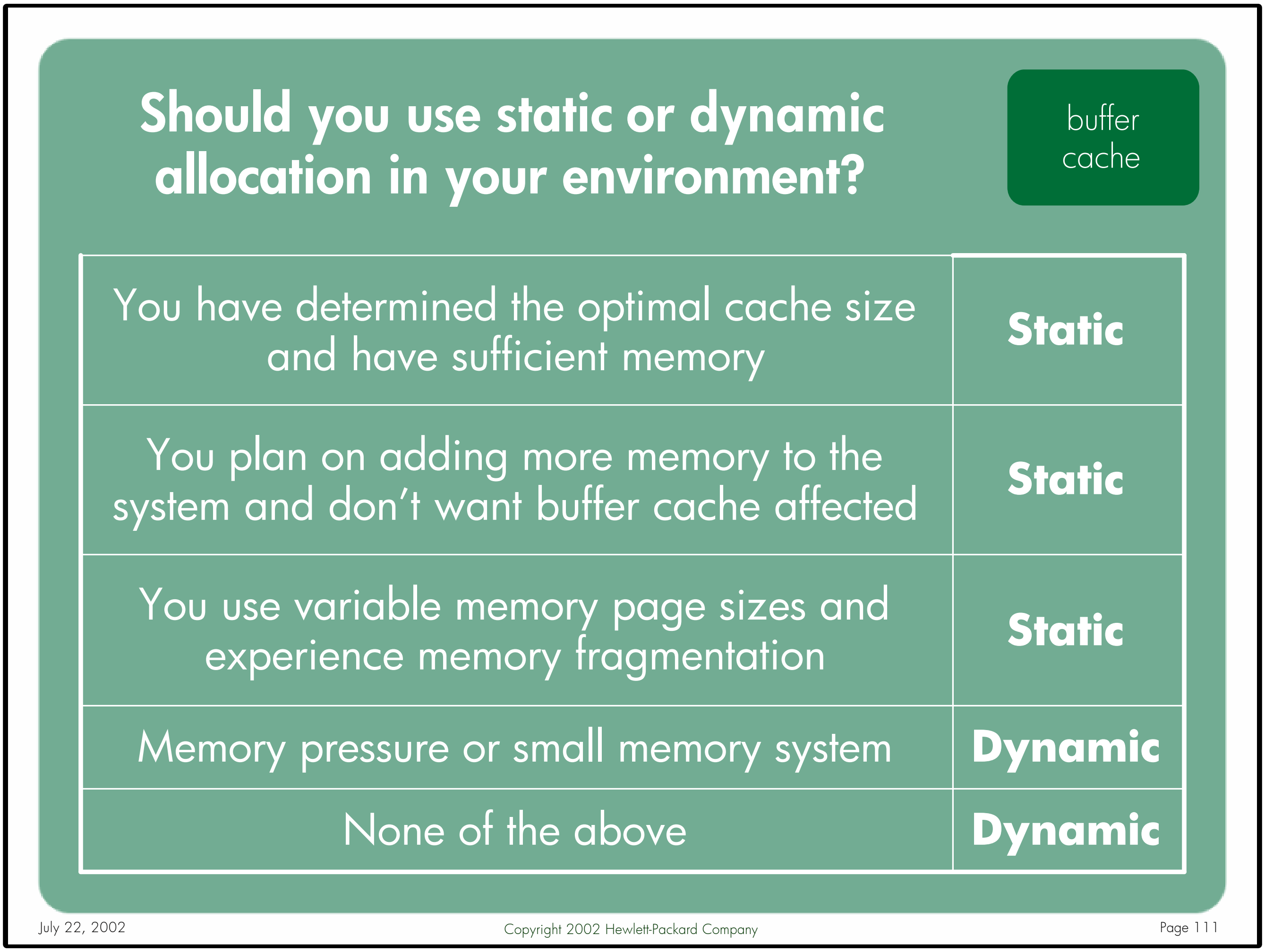
Static vs. Dynamic Allocation

buffer cache

- Static Allocation of Buffer Cache Memory
  - ➢ Configured via **nbuf** and/or **bufpages** parameters
  - ➢ **Fixed** number of pages allocated, regardless of system memory
  - ➢ **100%** allocated at boot time from **contiguous** memory
  - ➢ Pages are **"off limits"** to vhand in memory pressure situations
- Dynamic Allocation of Buffer Cache Memory
  - ➢ Configured via **dbc_min_pct** and **dbc_max_pct** parameters
  - ➢ Based on **% of physical memory**
  - ➢ Only **dbc_min_pct** of memory allocated at boot time
  - ➢ Pages can be **stolen** by vhand (down to **dbc_min_pct**)

Notes: Two methods of configuring buffer cache are supported: static and dynamic. The static method allocates a fixed number of 4KB buffers (configured via the bufpages kernel variable) and buffer header structures (configured via the nbuf kernel variable) at system boot time. The dynamic buffer cache method allocates buffer space and supporting data structures as they are needed, using defined minimum (dbc_min_pct kernel variable) and maximum (dbc_max_pct kernel variable) values to establish overall buffer cache size limits.

When a system experiences memory pressure, the system process vhand is invoked to try to reclaim memory pages from areas where it is not immediately needed so that processes which are memory starved can use it. One of the first places vhand looks for memory resources is dynamic buffer cache. Under severe memory pressure conditions, vhand will steal pages from dynamic buffer cache 3 times more often than anywhere else on the system (i.e. ordinary processes, shared memory, shared libraries, or memory-mapped regions). If no buffer cache pages are available (i.e. dbc_min_pct is reached or static buffer cache is configured) then vhand may need to deactivate processes to reclaim their memory pages.



Should you use static or dynamic allocation in your environment?

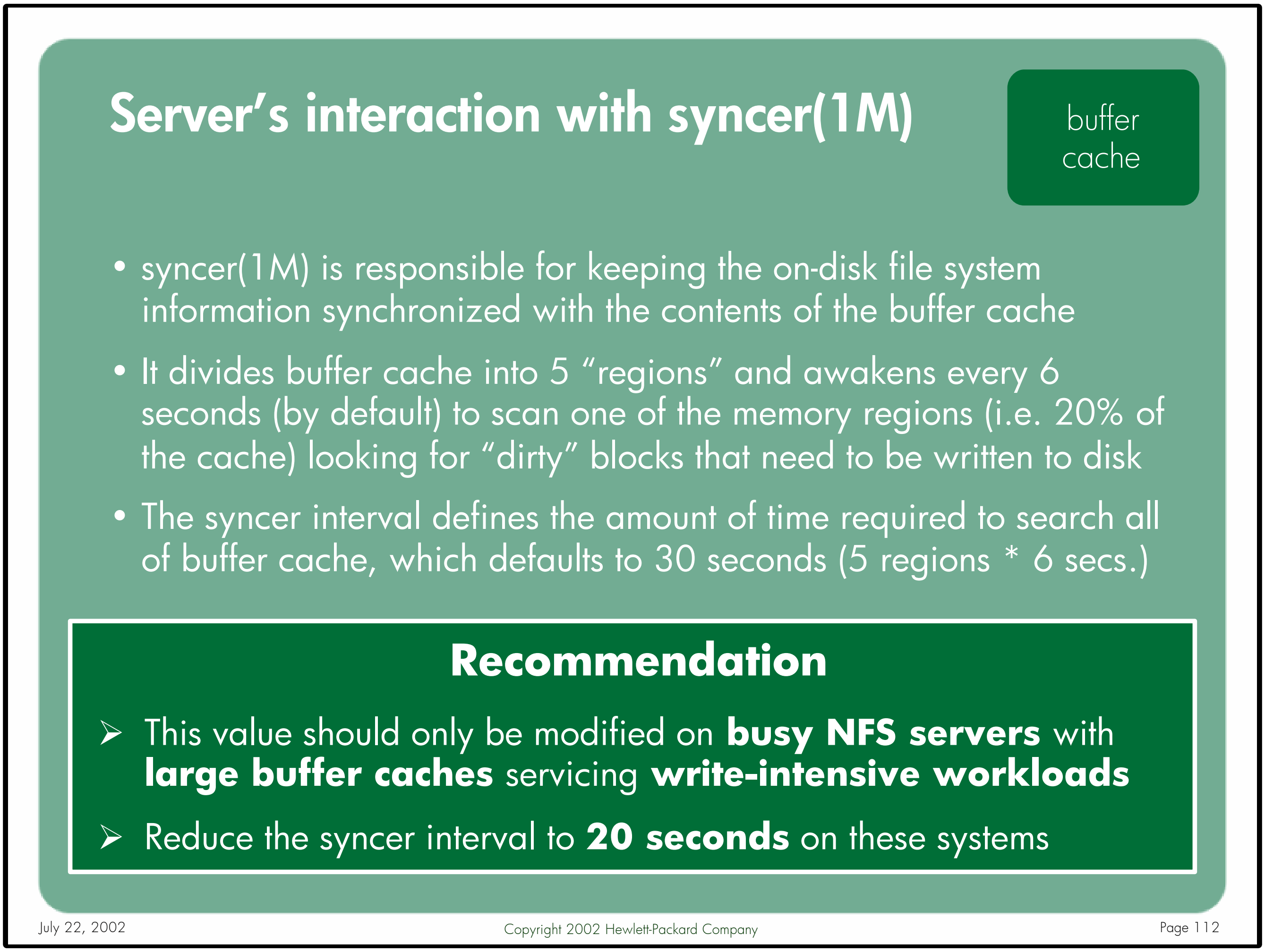
buffer cache

You have determined the optimal cache size and have sufficient memory	**Static**
You plan on adding more memory to the system and don't want buffer cache affected	**Static**
You use variable memory page sizes and experience memory fragmentation	**Static**
Memory pressure or small memory system	**Dynamic**
None of the above	**Dynamic**

Notes: Under most circumstances, the dynamic allocation method is the recommended method of allocating system memory to buffer cache. However, there are some environments where the static allocation method might offer some benefits.

If you have experimented with different amounts of buffer cache on a given client and have determined the optimal size of the cache (i.e. provides the best performance for your applications) then the easiest solution is to fix buffer cache at that size. Also, once you've determined the optimal size of the cache the last thing you want is for this size to inadvertently change without your consent. In the case where you need to add more physical memory to the system, perhaps due to growing application requirements, this will affect your total buffer cache size if you use dynamic allocation since it calculates the cache size as a percentage of memory.

In rare cases, customers using variable memory page sizes and dynamic buffer cache have reported problems of memory fragmentation and resulting poor performance. Static buffer cache avoids this issue by allocating all of its memory resources at system boot time in contiguous space.



Server's interaction with syncer(1M)

buffer cache

- syncer(1M) is responsible for keeping the on-disk file system information synchronized with the contents of the buffer cache
- It divides buffer cache into 5 "regions" and awakens every 6 seconds (by default) to scan one of the memory regions (i.e. 20% of the cache) looking for "dirty" blocks that need to be written to disk
- The syncer interval defines the amount of time required to search all of buffer cache, which defaults to 30 seconds (5 regions * 6 secs.)

Recommendation

- ➢ This value should only be modified on **busy NFS servers** with **large buffer caches** servicing **write-intensive workloads**
- ➢ Reduce the syncer interval to **20 seconds** on these systems

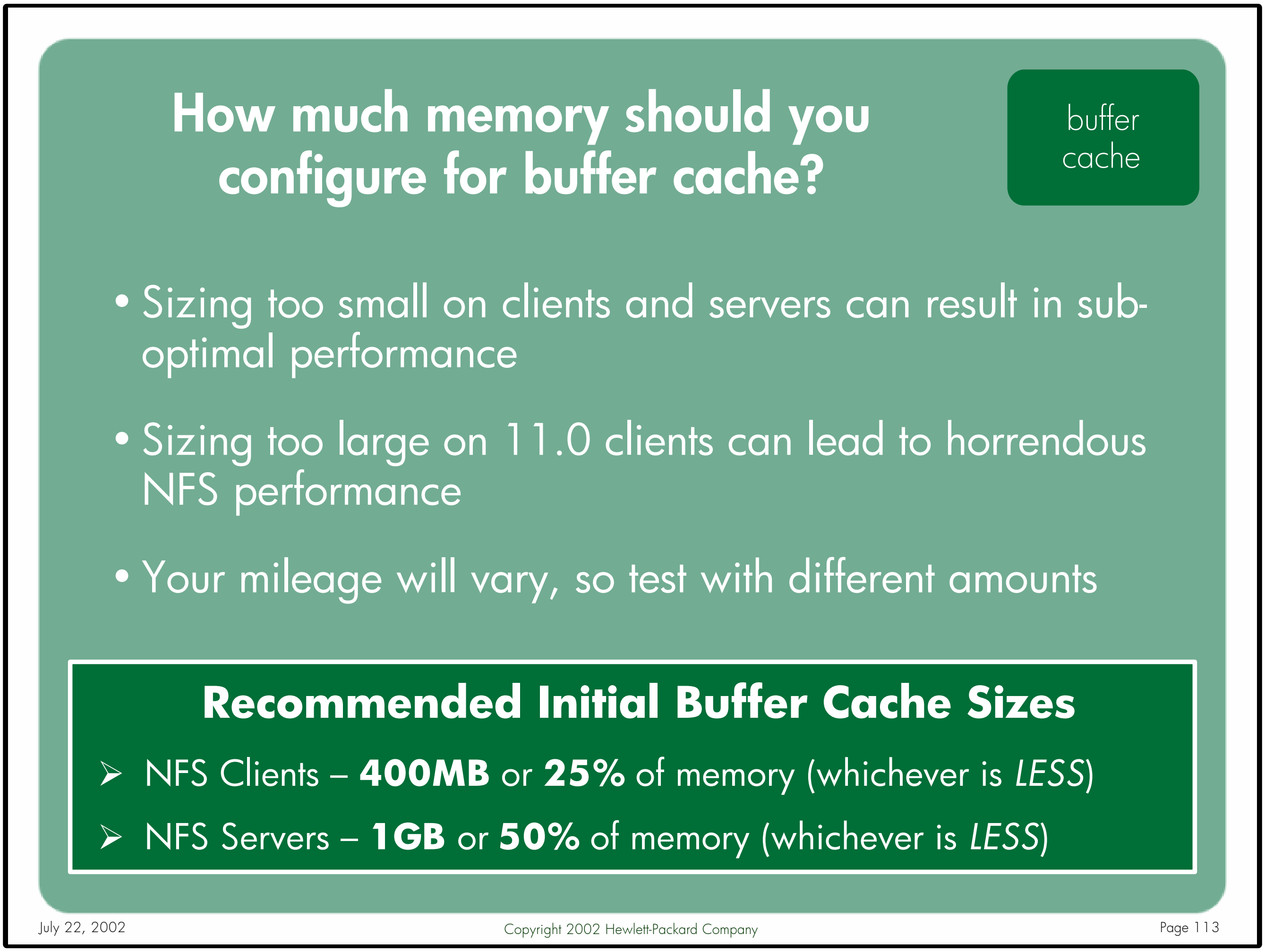
July 22, 2002

Page 112

Notes: The syncer(1M) daemon is responsible for keeping the on-disk file system information synchronized with the contents of the buffer cache. Syncer divides buffer cache into 5 "regions" each containing 20% of the total cache. Each region is processed once per interval. The default interval is 30 seconds, which tells the syncer to awaken every 6 seconds and process 20% of the cache.

NFS PV3 write data is placed in the server's buffer cache and marked for delayed writes semantics. Since syncer needs 30 seconds to process all of buffer cache (i.e. 6 seconds * 5 regions), this gives a busy NFS server a relatively long period of time to queue up write requests that need to be flushed. When the syncer flushes these buffers to disk, they transition to the disk's queue where they have the potential of getting in the way of synchronous transfers such as read() requests. Consequently there have been cases reported where read() calls take many seconds to complete because they are blocked behind all of the delayed writes being flushed from the server's buffer cache. This can make the NFS clients appear to "hang" for short periods of time.

The syncer interval should be changed in the `/sbin/init.d/syncer` script itself.



How much memory should you configure for buffer cache?

buffer cache

- Sizing too small on clients and servers can result in sub-optimal performance
- Sizing too large on 11.0 clients can lead to horrendous NFS performance
- Your mileage will vary, so test with different amounts

Recommended Initial Buffer Cache Sizes

- ➢ NFS Clients – **400MB** or **25%** of memory (whichever is *LESS*)
- ➢ NFS Servers – **1GB** or **50%** of memory (whichever is *LESS*)

July 22, 2002

Notes: Once you've decided whether to use static or dynamic buffer cache (based on the requirements of your applications and environment), the question becomes how much memory to assign to buffer cache.

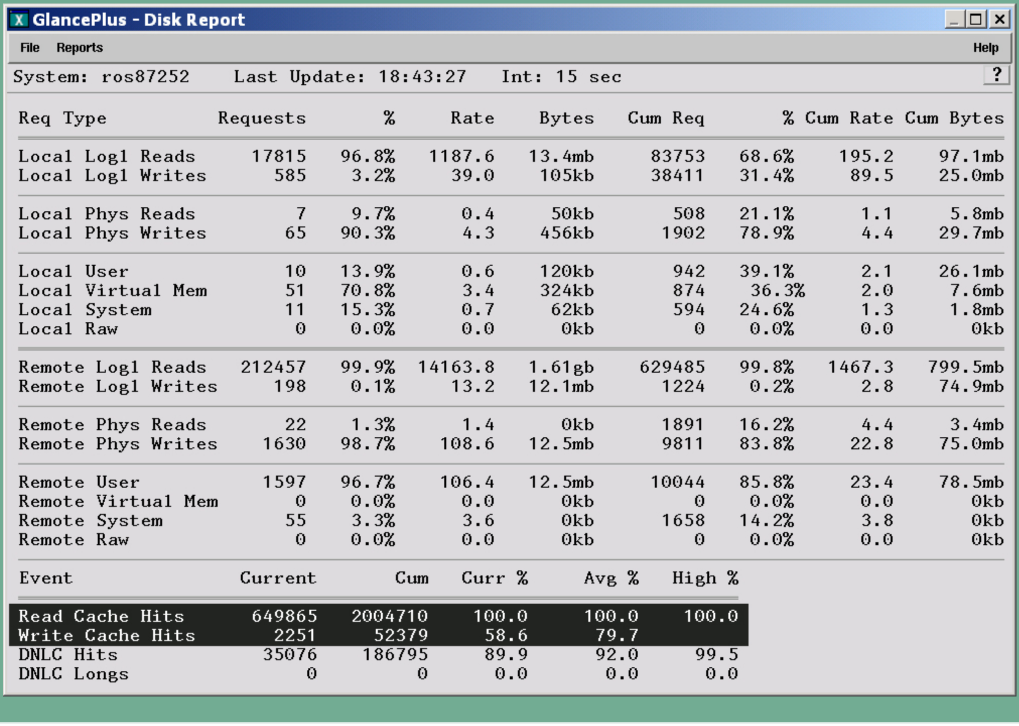
As stated earlier, configuring a large buffer cache on 11.0 clients can result in very poor NFS performance. Although the changes made to the buffer cache code in 11i should allow you to use a larger buffer cache without paying the performance penalties seen on 11.0, the question is 'Should you?' This question can only be accurately answered through sufficient testing in your own environment, using your own applications.

There is no one right answer for every NFS client, however a good starting point for most clients is 400MB or 25% of physical memory – whichever is LESS.

Similarly, there is no one right answer for every NFS server, but a good starting point for most servers is 1GB or 50% of physical memory – whichever is LESS.

Measuring Buffer Cache Utilization

buffer cache



```
System: ros87252    Last Update: 18:43:27    Int: 15 sec

Req Type             Requests       %     Rate    Bytes   Cum Req       % Cum Rate Cum Bytes

Local Logl Reads        17815   96.8%   1187.6   13.4mb     83753   68.6%    195.2    97.1mb
Local Logl Writes         585    3.2%     39.0    105kb     38411   31.4%     89.5    25.0mb

Local Phys Reads            7    9.7%      0.4     50kb       508   21.1%      1.1     5.8mb
Local Phys Writes          65   90.3%      4.3    456kb      1902   78.9%      4.4    29.7mb

Local User                 10   13.9%      0.6    120kb       942   39.1%      2.1    26.1mb
Local Virtual Mem          51   70.8%      3.4    324kb       874    36.3%     2.0     7.6mb
Local System               11   15.3%      0.7     62kb       594   24.6%      1.3     1.8mb
Local Raw                   0    0.0%      0.0      0kb         0    0.0%      0.0       0kb

Remote Logl Reads      212457   99.9%  14163.8   1.61gb    629485   99.8%   1467.3   799.5mb
Remote Logl Writes        198    0.1%     13.2   12.1mb      1224    0.2%      2.8    74.9mb

Remote Phys Reads          22    1.3%      1.4      0kb      1891   16.2%      4.4     3.4mb
Remote Phys Writes       1630   98.7%    108.6   12.5mb      9811   83.8%     22.8    75.0mb

Remote User              1597   96.7%    106.4   12.5mb     10044   85.8%     23.4    78.5mb
Remote Virtual Mem          0    0.0%      0.0      0kb         0    0.0%      0.0       0kb
Remote System              55    3.3%      3.6      0kb      1658   14.2%      3.8       0kb
Remote Raw                  0    0.0%      0.0      0kb         0    0.0%      0.0       0kb

Event                   Current         Cum   Curr %     Avg %    High %

Read Cache Hits          649865     2004710    100.0     100.0     100.0
Write Cache Hits           2251       52379     58.6      79.7
DNLC Hits                 35076      186795     89.9      92.0      99.5
DNLC Longs                    0           0      0.0       0.0       0.0
```

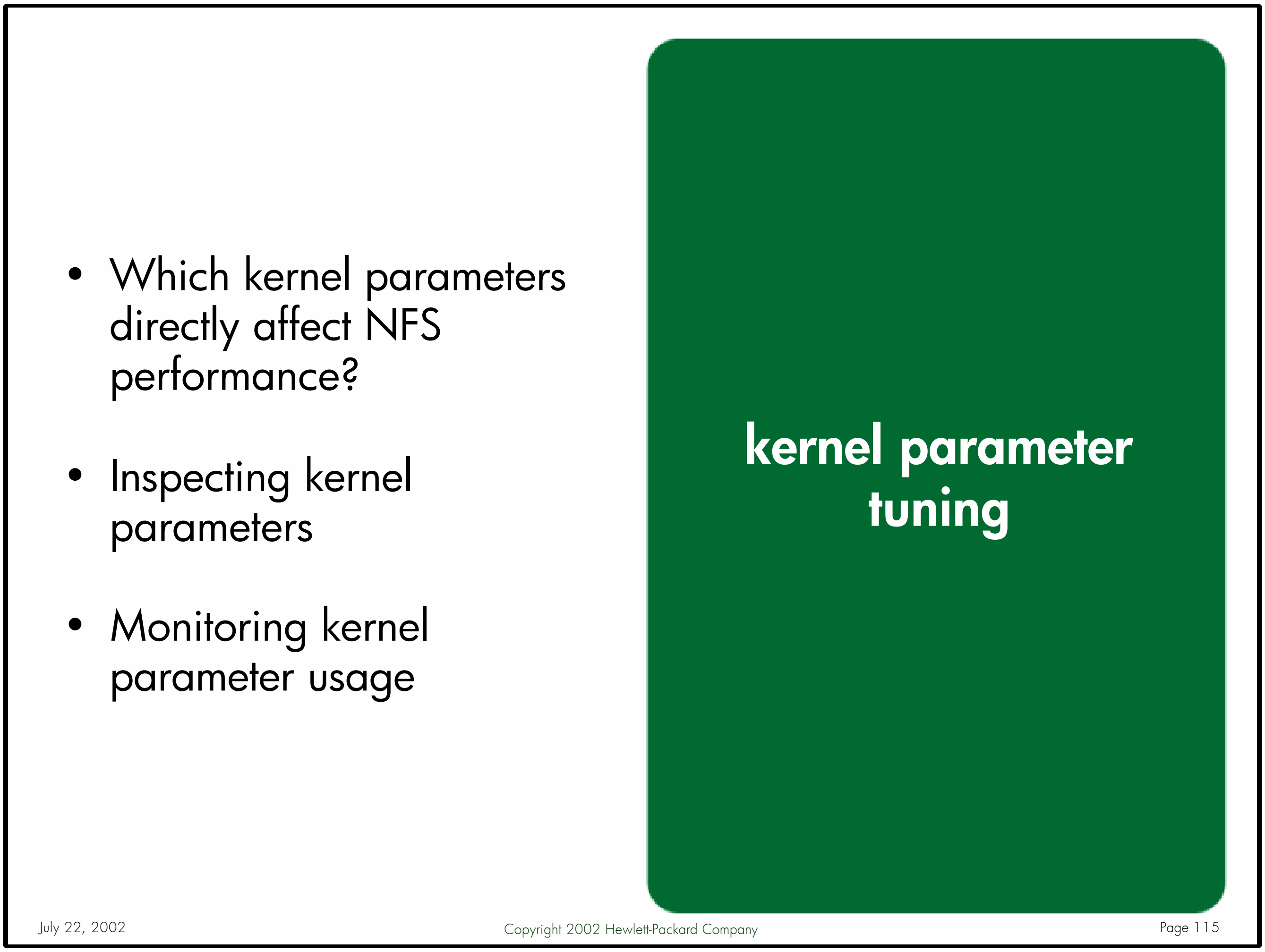
July 22, 2002

Copyright 2002 Hewlett-Packard Company

Page 114

Notes: After configuring your system with reasonable sized buffer cache, the next step is to run your applications and evaluate the performance with these buffer cache settings.

You can either time the performance of your application using the time(1) or timex(1) commands, or simply use a stopwatch. You can also monitor the buffer cache read and write hit rates using a performance measurement tool such as Glance Plus (shown above), or via the sar(1M) command using the "–b" option.



- Which kernel parameters directly affect NFS performance?
- Inspecting kernel parameters
- Monitoring kernel parameter usage

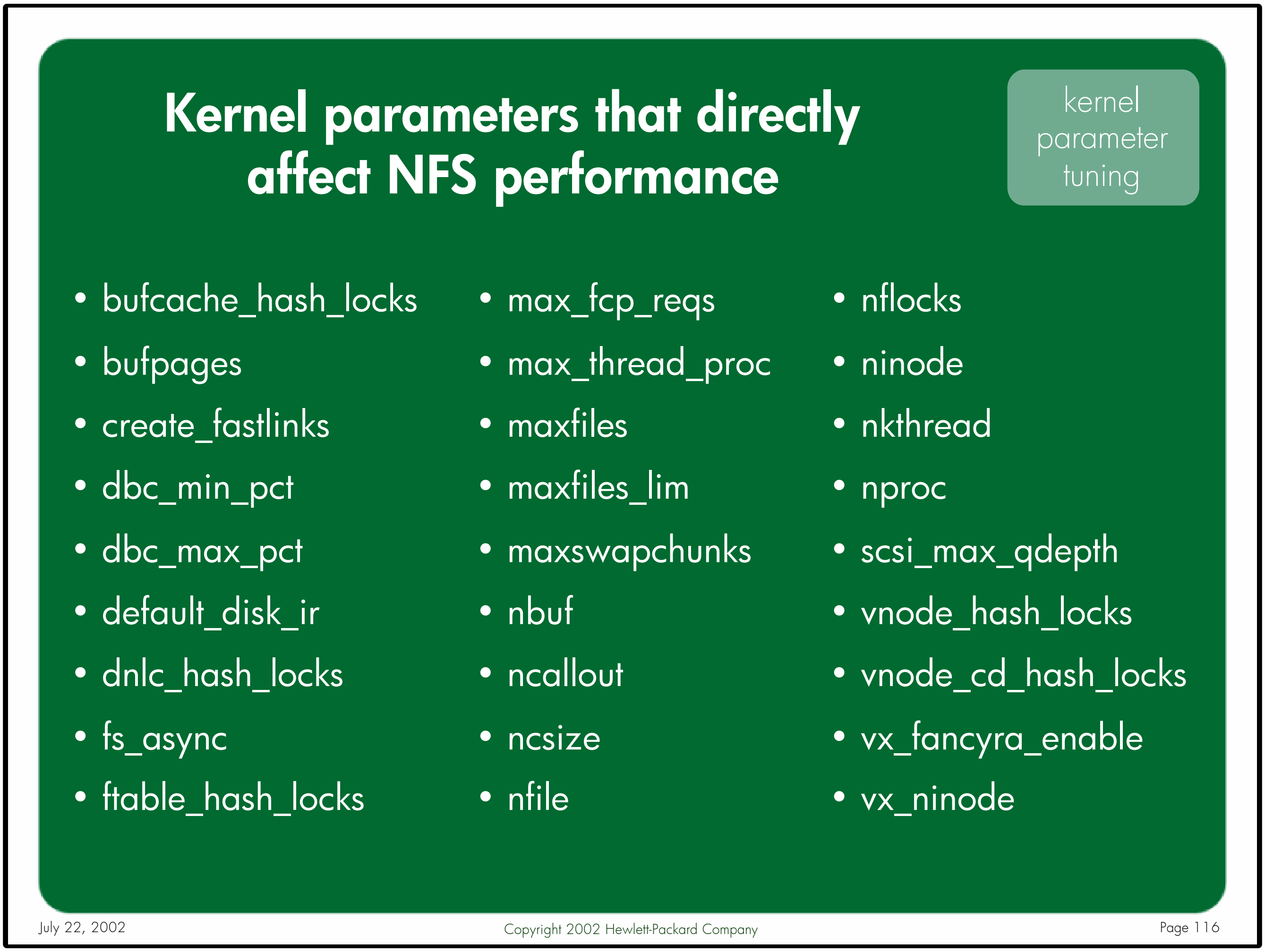
kernel parameter tuning

July 22, 2002

Page 115

Notes: Since NFS spends the majority of its time running in the kernel, it should come as no surprise that there are many kernel parameters that can positively or negatively impact NFS performance. Like most other facets of system performance, there is not one universal setting for kernel variables that will work for every client or server in every environment. The values described in this section are merely recommendations for starting values. You should perform extensive testing with your applications in your environment to determine the optimal settings for these variables on your systems.

In addition to describing the parameters, this section describes several tools available for inspecting the current values of these parameters. Also described are the tools that allow you to monitor the utilization rate of some of the key parameters.



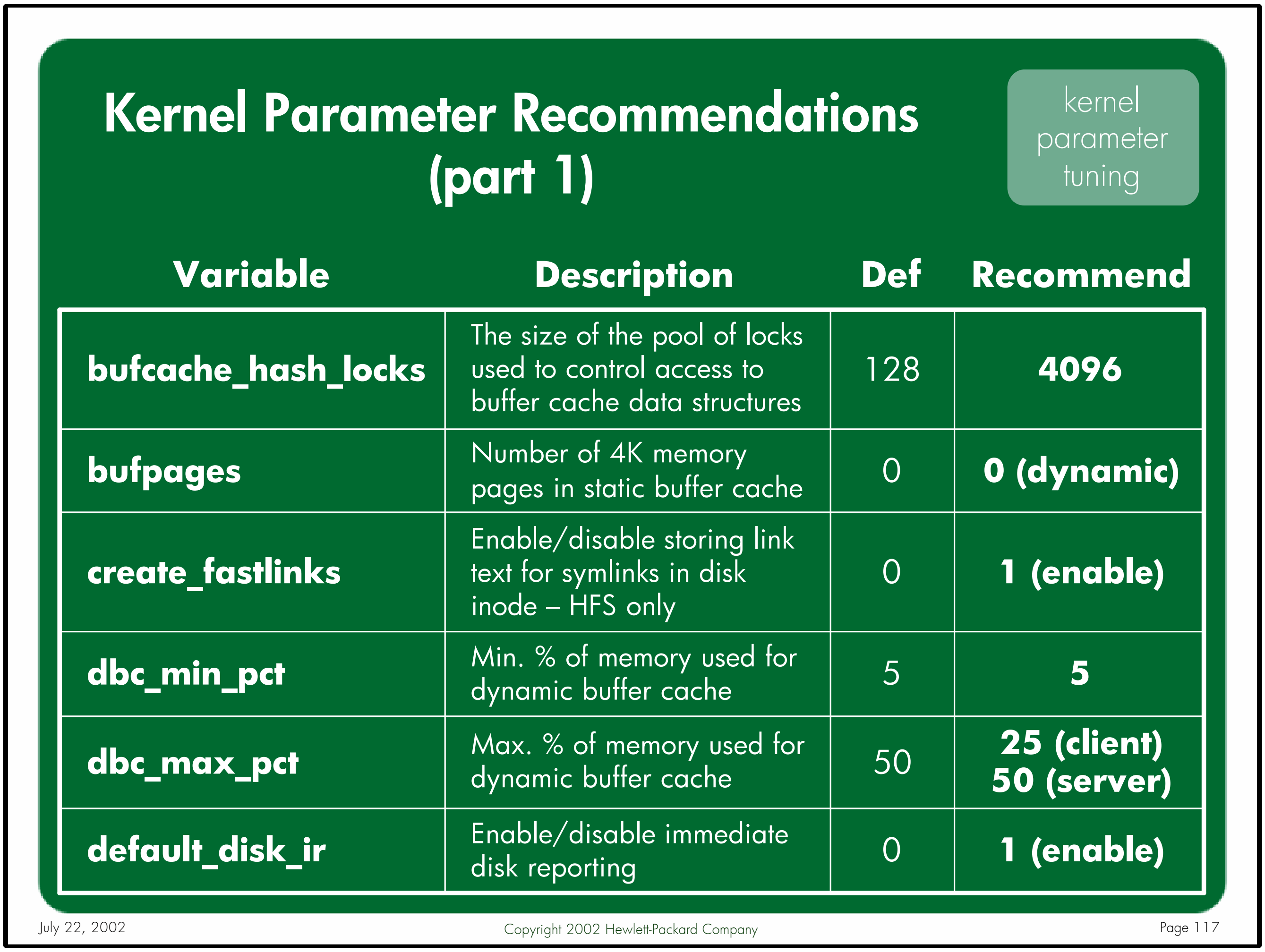
Kernel parameters that directly affect NFS performance

kernel parameter tuning

- bufcache_hash_locks
- bufpages
- create_fastlinks
- dbc_min_pct
- dbc_max_pct
- default_disk_ir
- dnlc_hash_locks
- fs_async
- ftable_hash_locks
- max_fcp_reqs
- max_thread_proc
- maxfiles
- maxfiles_lim
- maxswapchunks
- nbuf
- ncallout
- ncsize
- nfile
- nflocks
- ninode
- nkthread
- nproc
- scsi_max_qdepth
- vnode_hash_locks
- vnode_cd_hash_locks
- vx_fancyra_enable
- vx_ninode

Notes: The parameters shown here in no way constitute a complete list of the kernel parameters needed for every client or server system. The ones described in this section can directly impact NFS performance on HP-UX 11.0 and 11i systems.

Many applications require other kernel parameter settings for things like maximum process storage, text and data segment sizes, semaphores, shared memory, etc. Be sure to check with your software supplier to ensure that you configure the kernel variables appropriately for your application software.



Kernel Parameter Recommendations (part 1)

kernel parameter tuning

Variable	Description	Def	Recommend
bufcache_hash_locks	The size of the pool of locks used to control access to buffer cache data structures	128	**4096**
bufpages	Number of 4K memory pages in static buffer cache	0	**0 (dynamic)**
create_fastlinks	Enable/disable storing link text for symlinks in disk inode – HFS only	0	**1 (enable)**
dbc_min_pct	Min. % of memory used for dynamic buffer cache	5	**5**
dbc_max_pct	Max. % of memory used for dynamic buffer cache	50	**25 (client)** **50 (server)**
default_disk_ir	Enable/disable immediate disk reporting	0	**1 (enable)**

July 22, 2002

Page 117

Notes:

bufcache_hash_locks

Sizes the pool of locks used to control access to buffer cache data structures.

bufpages

The bufpages variable specifies how many 4096-byte memory pages are allocated for the static sized buffer cache. Both bufpages and nbuf must be set to 0 to enable dynamic buffer cache.

create_fastlinks

When create_fastlinks is enabled, it causes the system to create **HFS** symbolic links in a manner that reduces the number of disk accesses by one for each symbolic link in a pathname lookup.

dbc_min_pct

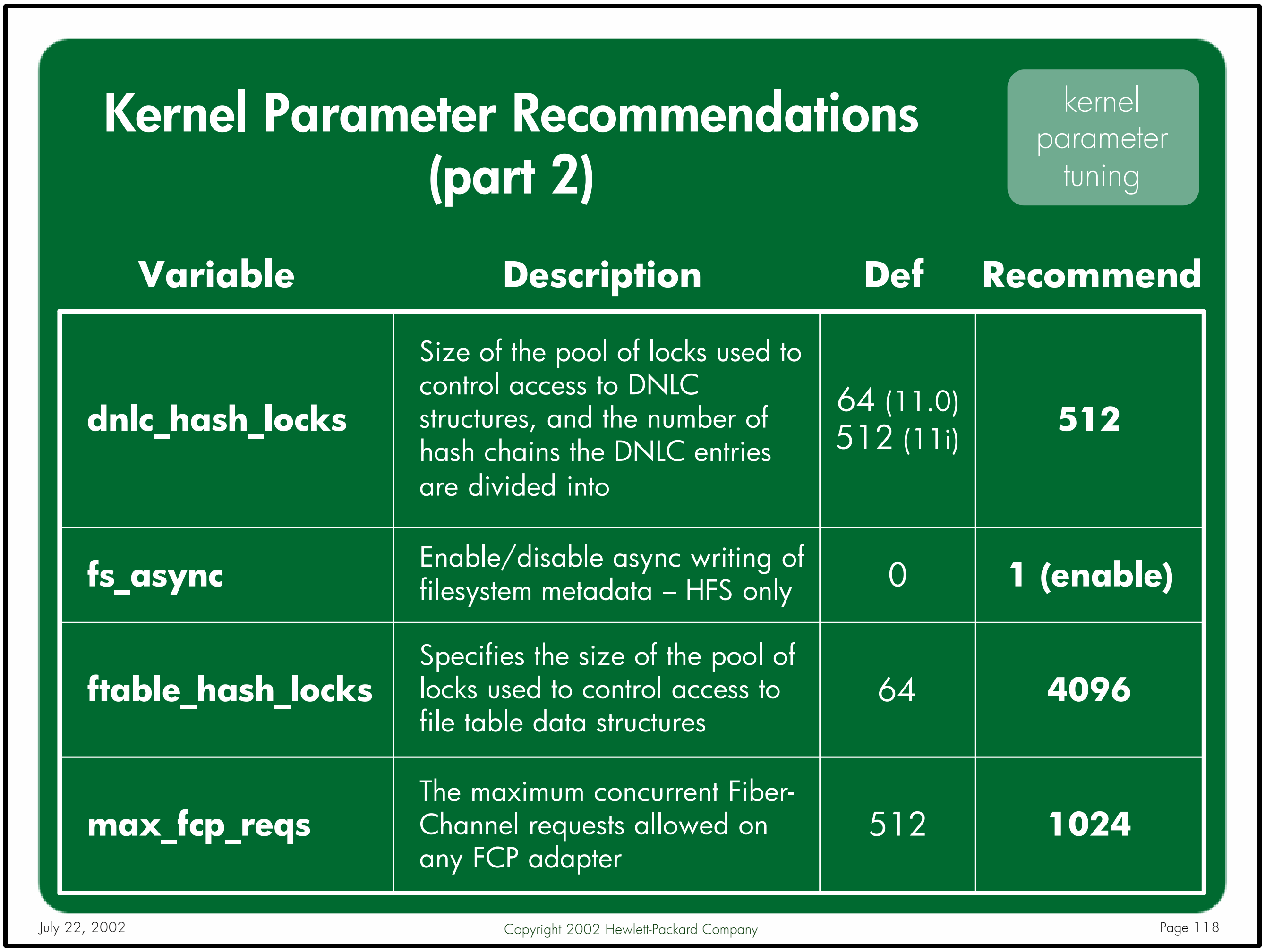
The value of dbc_min_pct specifies the minimum percentage of physical memory that is reserved for use by the dynamic buffer cache.

dbc_max_pct

The value of dbc_max_pct specifies the maximum percentage of physical memory that is reserved for use by the dynamic buffer cache.

default_disk_ir

When enabled, disk drives that have data caches return from a write() system call when the data is cached, rather than returning after the data is written on the media. Data loss can occur if the disk's power fails, however this is usually negated via RAID strategies or arrays with UPS.



Kernel Parameter Recommendations (part 2)

kernel parameter tuning

Variable	Description	Def	Recommend
dnlc_hash_locks	Size of the pool of locks used to control access to DNLC structures, and the number of hash chains the DNLC entries are divided into	64 (11.0) 512 (11i)	**512**
fs_async	Enable/disable async writing of filesystem metadata – HFS only	0	**1 (enable)**
ftable_hash_locks	Specifies the size of the pool of locks used to control access to file table data structures	64	**4096**
max_fcp_reqs	The maximum concurrent Fiber-Channel requests allowed on any FCP adapter	512	**1024**

July 22, 2002

Page 118

Notes:

dnlc_hash_locks

Specifies the size of the pool of locks used to control access to the DNLC (Directory Name Lookup Cache) data structures. This value also specifies the number of times the DNLC will be divided into hash chains for searching purposes.

The *dnlc_hash_locks* kernel variable did not exist when HP-UX 11.0 was released. It was introduced in patch PHKL_12965, which has since been replaced by patch PHKL_18543. *dnlc_hash_locks* has a default value of 64 on HP-UX 11.0. The default value was increased to 512 in HP-UX 11i.

fs_async

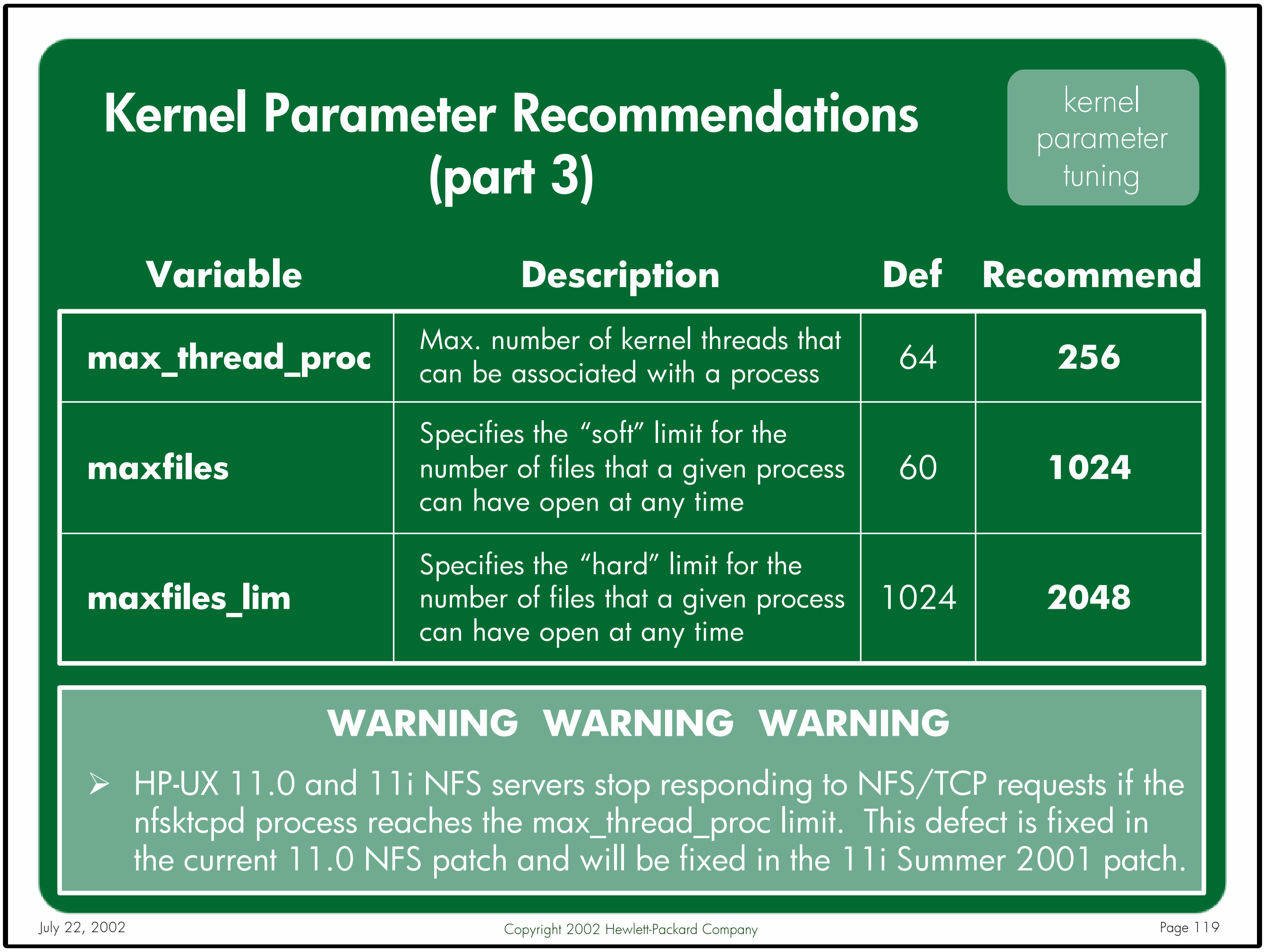
Specifies whether or not asynchronous writing of **HFS** filesystem metadata is allowed.

ftable_hash_locks

Specifies the size of the pool of locks used to control access to the file table data structures.

max_fcp_reqs

This variable specifies the maximum number of Fiber-Channel requests that may be queued on any FCP adapter in the system at any time.



Kernel Parameter Recommendations (part 3)

kernel parameter tuning

Variable	Description	Def	Recommend
max_thread_proc	Max. number of kernel threads that can be associated with a process	64	**256**
maxfiles	Specifies the "soft" limit for the number of files that a given process can have open at any time	60	**1024**
maxfiles_lim	Specifies the "hard" limit for the number of files that a given process can have open at any time	1024	**2048**

WARNING WARNING WARNING

- HP-UX 11.0 and 11i NFS servers stop responding to NFS/TCP requests if the nfsktcpd process reaches the max_thread_proc limit. This defect is fixed in the current 11.0 NFS patch and will be fixed in the 11i Summer 2001 patch.

July 22, 2002

Page 119

Notes:

max_thread_proc

Limits the number of threads a single process is allowed to create. This protects the system from excessive use of system resources if a run-away process creates more threads than it should.

A defect exists in the HP-UX 11.0 and 11i NFS/TCP server code which causes the server to stop responding to inbound requests once the nfsktcpd process reaches the maximum number of kernel threads defined by max_thread_proc, effectively causing all NFS/TCP access to this server to hang. This defect is fixed in the current 11.0 NFS patch. The same fix will be made available for 11i in the Summer 2001 NFS patch.

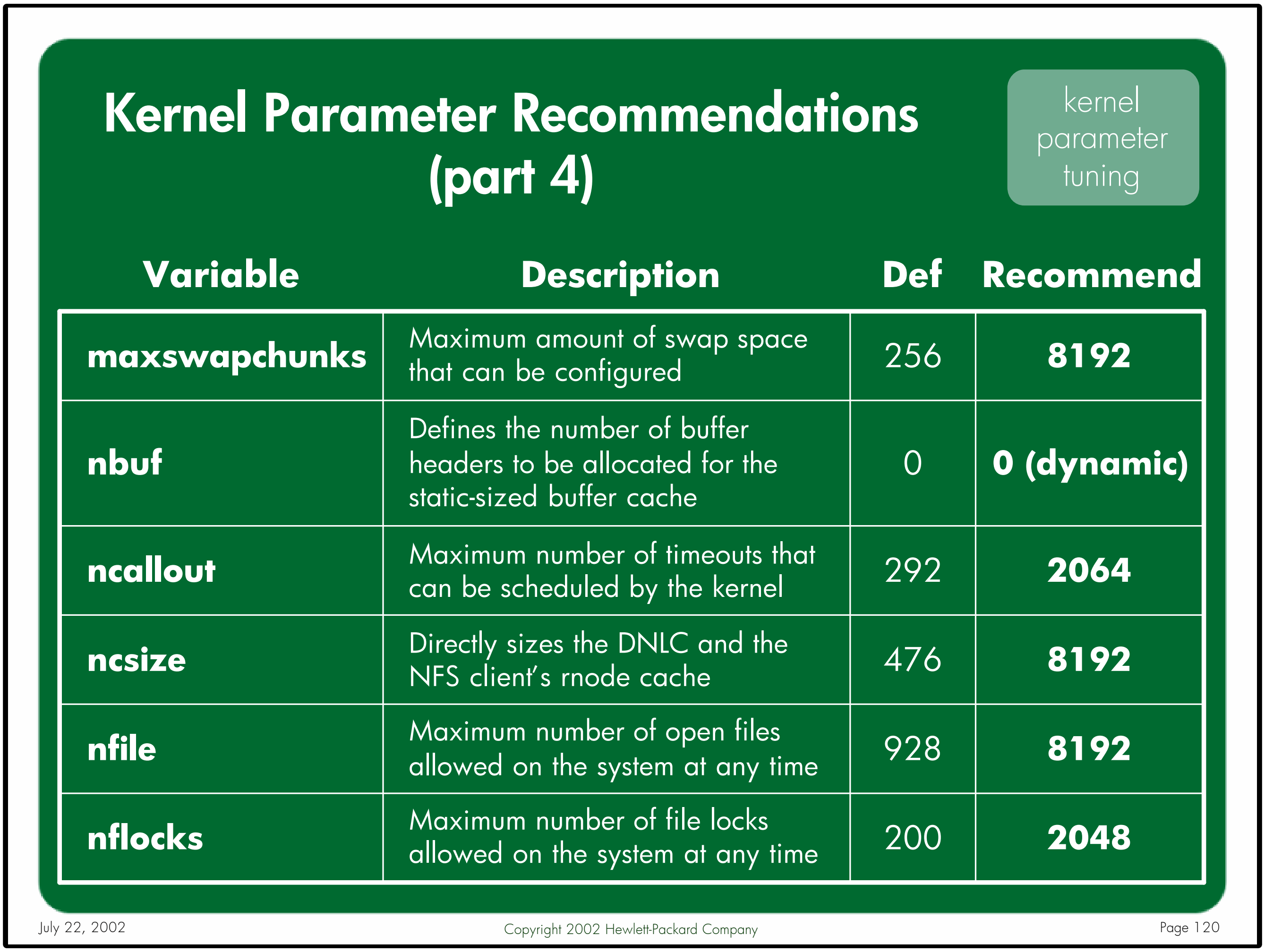
Since TCP is the default network protocol used for NFS filesystems in 11i, it is important to make sure that 11i servers configure max_thread_proc large enough to avoid this hang. Once the Summer 11i NFS patch becomes available, max_thread_proc may be reduced to a lower value.

maxfiles

Specifies the "soft" limit for the number of files a process may open at any one time without having to call `setrlimit(2)` to increase the soft limit.

maxfiles_lim

Specifies the "hard" limit for the number of files a process may open at any one time.



Kernel Parameter Recommendations (part 4)

kernel parameter tuning

Variable	Description	Def	Recommend
maxswapchunks	Maximum amount of swap space that can be configured	256	**8192**
nbuf	Defines the number of buffer headers to be allocated for the static-sized buffer cache	0	**0 (dynamic)**
ncallout	Maximum number of timeouts that can be scheduled by the kernel	292	**2064**
ncsize	Directly sizes the DNLC and the NFS client's rnode cache	476	**8192**
nfile	Maximum number of open files allowed on the system at any time	928	**8192**
nflocks	Maximum number of file locks allowed on the system at any time	200	**2048**

July 22, 2002 Copyright 2002 Hewlett-Packard Company Page 120

Notes: **maxswapchunks**

Used to compute the maximum amount of configurable swap space on the system.

nbuf

Specifies the number of buffer headers to be allocated for the static file system buffer cache. Each buffer is allocated 4096 bytes of memory unless overridden by a conflicting value for bufpages. Both nbuf and bufpages need to be set to 0 to enable dynamic buffer cache.

ncallout

Specifies the maximum number of timeouts that can be scheduled by the kernel at any given time. Timeouts are used by: alarm(), setitimer(), select(), drivers, uucp, and process scheduling.

ncsize

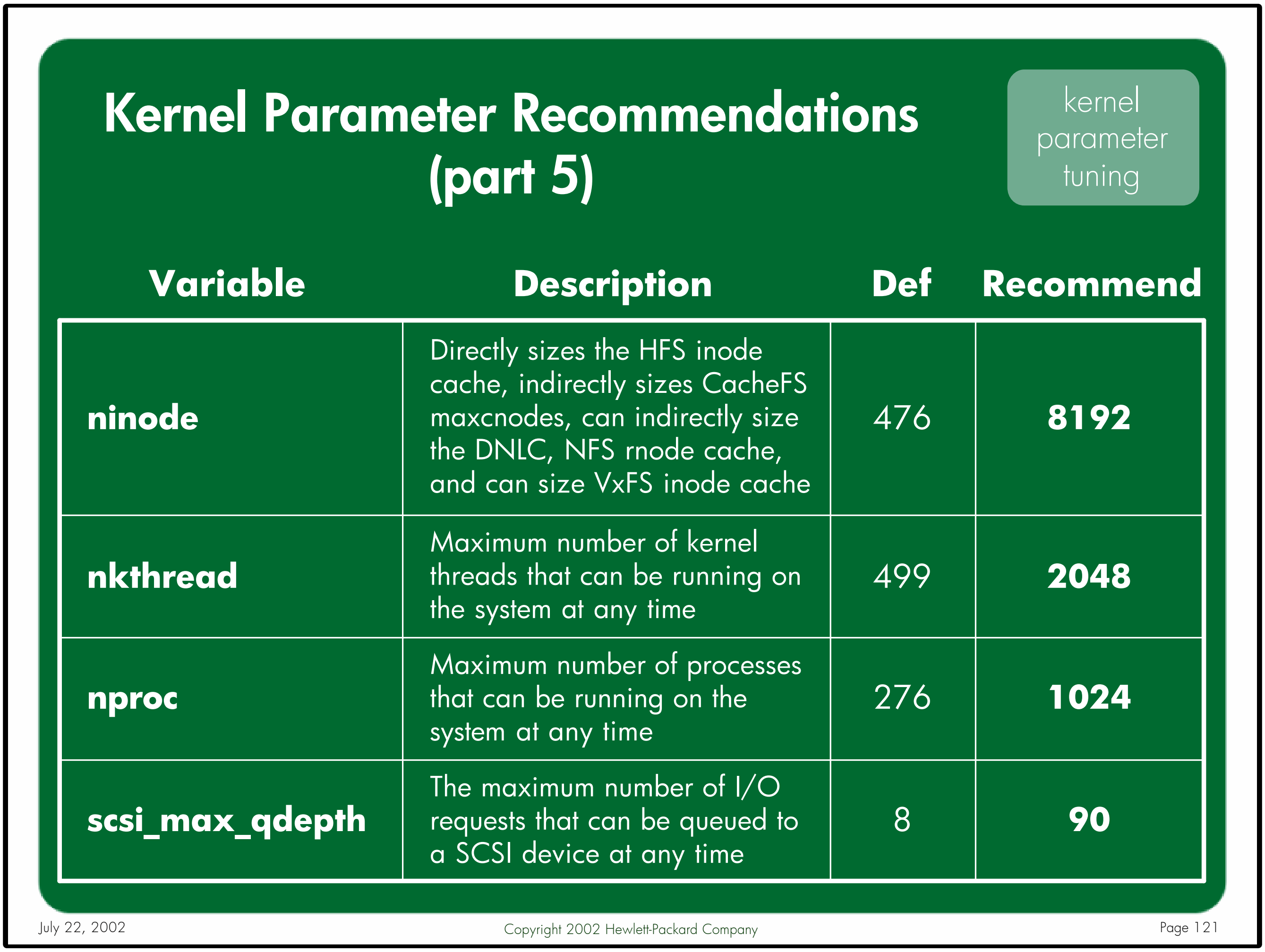
Directly sizes the DNLC, used to store directory pathname information related to recently accessed directories and files in the file system. Also sizes the NFS client's rnode table.

nfile

Defines the maximum number files that can be open at any one time, system-wide.

nflocks

Specifies the maximum number of file/record locks that are available system-wide.



Kernel Parameter Recommendations (part 5)

kernel parameter tuning

Variable	Description	Def	Recommend
ninode	Directly sizes the HFS inode cache, indirectly sizes CacheFS maxcnodes, can indirectly size the DNLC, NFS rnode cache, and can size VxFS inode cache	476	**8192**
nkthread	Maximum number of kernel threads that can be running on the system at any time	499	**2048**
nproc	Maximum number of processes that can be running on the system at any time	276	**1024**
scsi_max_qdepth	The maximum number of I/O requests that can be queued to a SCSI device at any time	8	**90**

July 22, 2002

Page 121

Notes: **ninode**

Directly sizes the HFS inode table, indirectly sizes the DNLC, indirectly sizes the NFS client's rnode table, is used by CacheFS (HP-UX 11i systems only) to size the maxcnodes kernel variable, and can be used to indirectly size the VxFS-specific inode table – only if the ninode value is larger than the VxFS inode table size calculated by the kernel based on the amount of physical memory in the system. Since the kernel typically builds very large VxFS inode caches, the chances of ninode directly sizing the VxFS inode cache are extremely remote.

nkthread

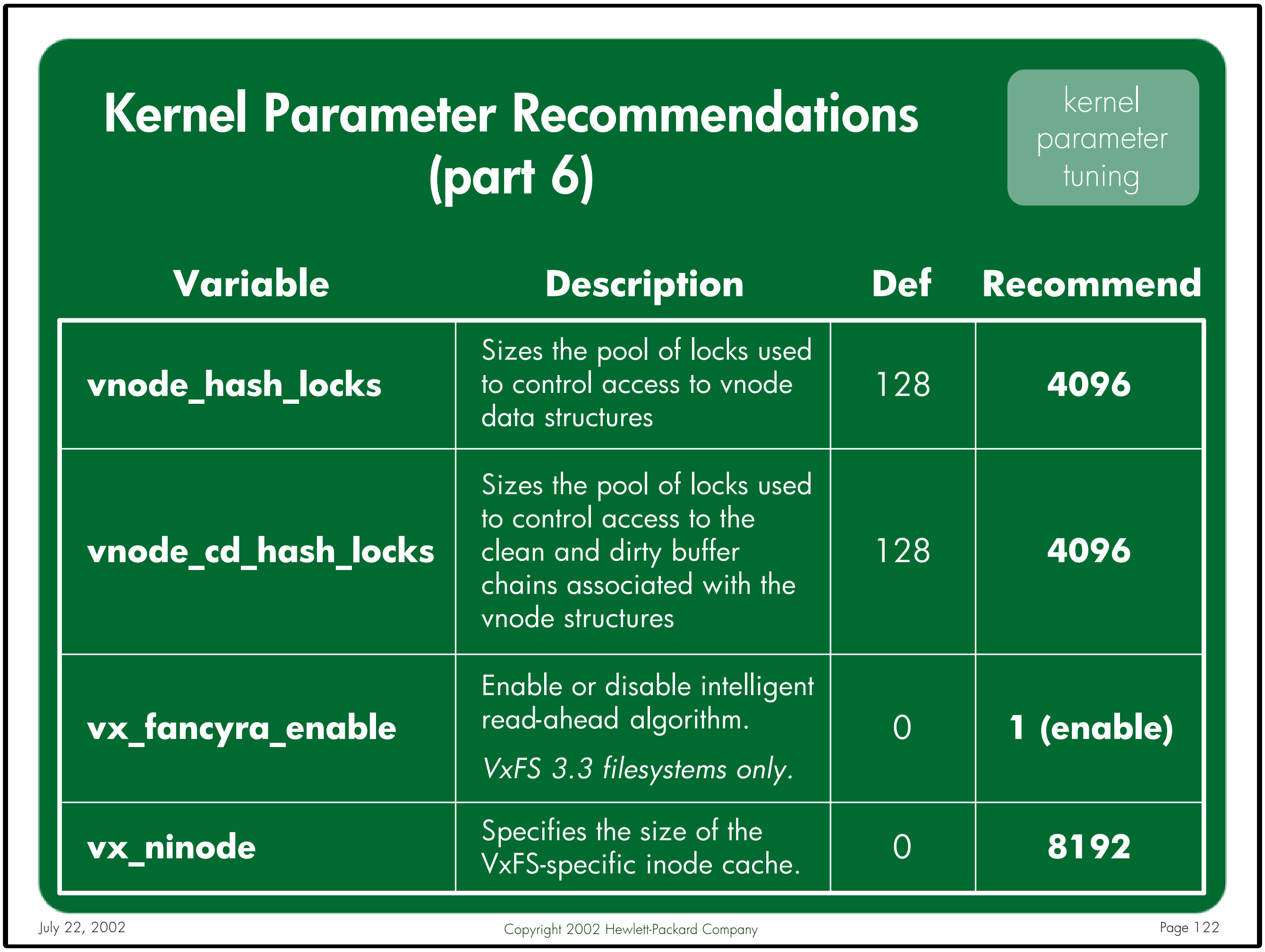
Limits the total number of threads that can be running on the system at any given time from all processes.

nproc

Directly specifies the maximum total number of processes that can exist simultaneously in the system at any given time.

scsi_max_qdepth

The maximum number of I/O requests that may be queued to a SCSI device at any time.



Kernel Parameter Recommendations (part 6)

kernel parameter tuning

Variable	Description	Def	Recommend
vnode_hash_locks	Sizes the pool of locks used to control access to vnode data structures	128	**4096**
vnode_cd_hash_locks	Sizes the pool of locks used to control access to the clean and dirty buffer chains associated with the vnode structures	128	**4096**
vx_fancyra_enable	Enable or disable intelligent read-ahead algorithm. *VxFS 3.3 filesystems only.*	0	**1 (enable)**
vx_ninode	Specifies the size of the VxFS-specific inode cache.	0	**8192**

July 22, 2002 Copyright 2002 Hewlett-Packard Company Page 122

Notes:

vnode_hash_locks

Specifies the size of the pool of locks used to control access to the vnode data structures.

vnode_cd_hash_locks

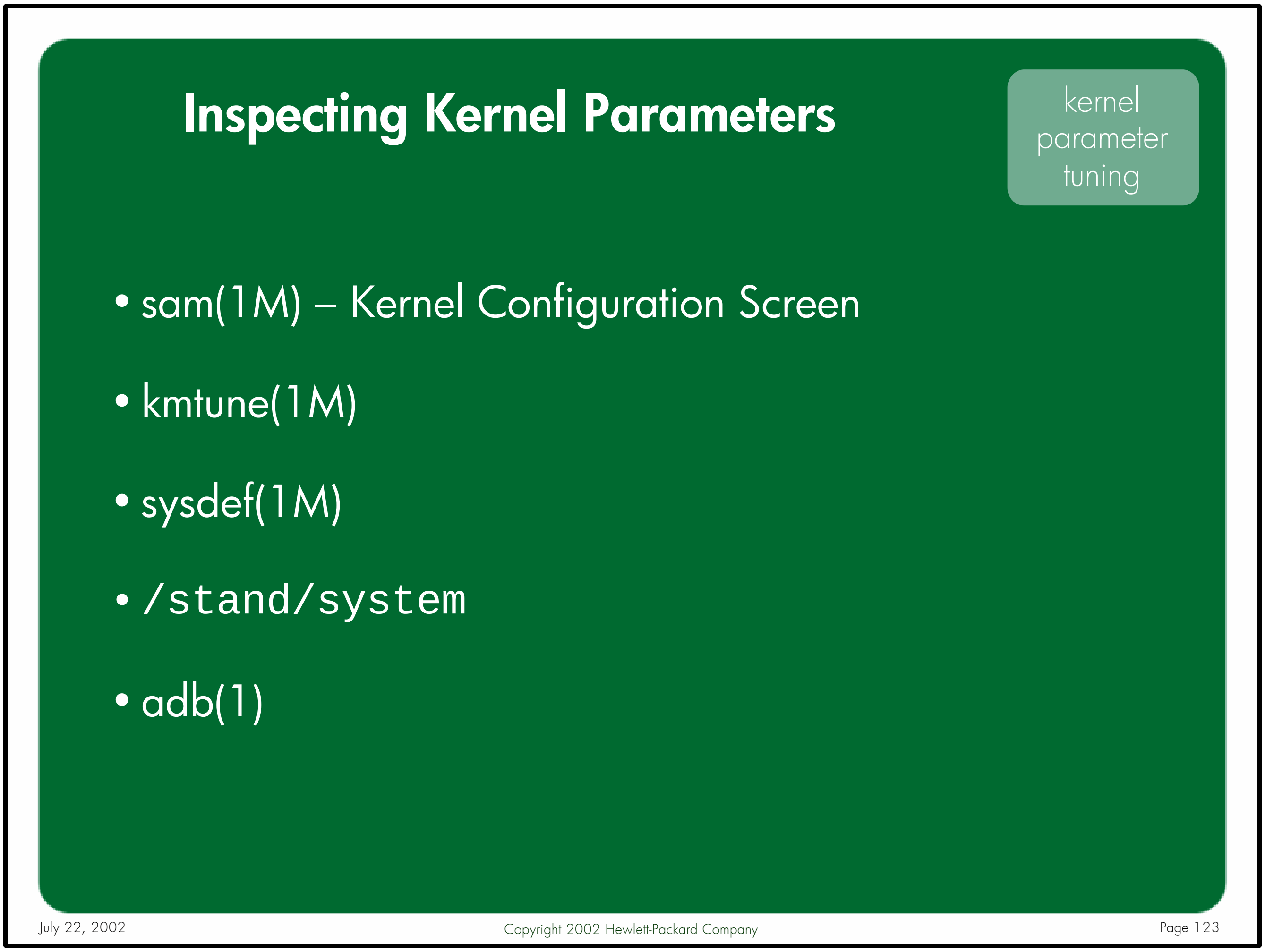
Specifies the size of the pool of locks used to control access to the clean and dirty buffer chains associated with the vnode structures.

vx_fancyra_enable

Enables or disables a new intelligent read-ahead algorithm. This algorithm allows read-aheads to occur in cases such as “backward read,” “stride read,” files read by multiple processes simultaneously, and “collective reads.” These types of reads are typically performed by EDA applications. The *vx_fancyra_enable* kernel parameter did not exist when HP-UX 11.0 was released. It was introduced in patch PHKL_22414. This parameter was included in HP-UX 11i.

vx_ninode

Specifies the size of the VxFS-specific inode cache. This can be used to override the size of the VxFS cache calculated by the kernel at boot time based on physical memory. Can be very important on large memory systems. For example, on a system with 128GB of memory, the kernel will create a VxFS inode table containing 1,024,000 entries. *vx_ninode* did not exist when HP-UX 11.0 and 11i released. It was introduced in patch PHKL_18543 (11.0) and PHKL_24783 (11i).



Inspecting Kernel Parameters

kernel parameter tuning

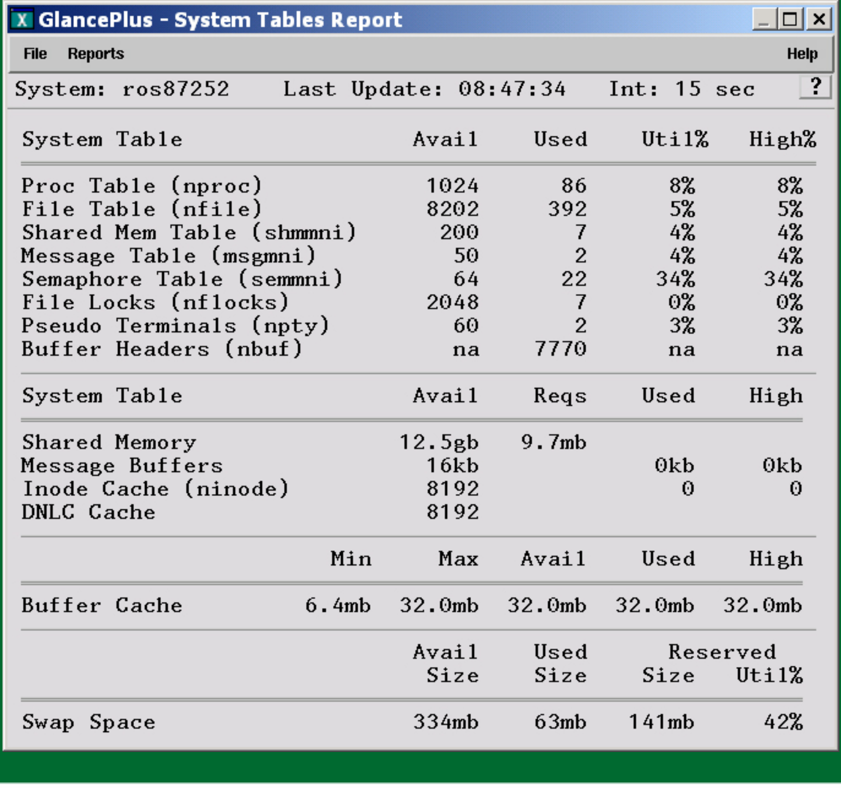
- sam(1M) – Kernel Configuration Screen
- kmtune(1M)
- sysdef(1M)
- `/stand/system`
- adb(1)

Notes: There are several tools available for determining the current size of the various kernel parameters on the system. Some of these tools can also describe how the sizes of these parameters are calculated in the kernel. Additionally, sam(1M), kmtune(1M), and adb(1) can be used to change the values of the kernel parameters.

The `/stand/system` file is the configuration file used to build the kernel. Among other things, it contains a list of the modified kernel parameters used on the system. Since it is an ASCII text file, it can be viewed with any text editor. Changes made to this file only take effect when a new kernel is built referencing the changed file via the mk_kernel(1M) command.

Measuring Kernel Parameter Usage

kernel parameter tuning



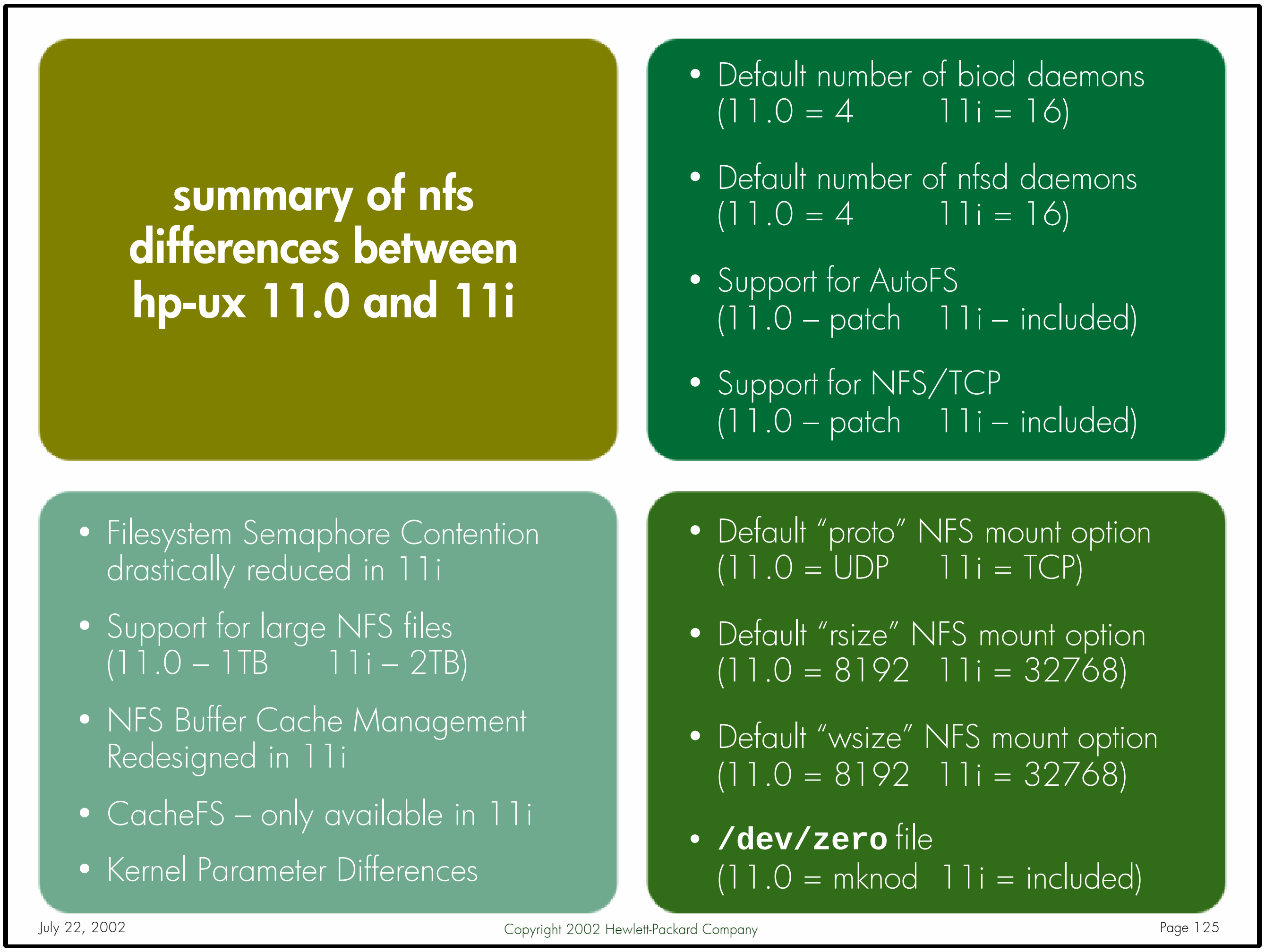
```
System: ros87252    Last Update: 08:47:34    Int: 15 sec

 System Table                     Avail    Used    Util%    High%

 Proc Table (nproc)                1024      86       8%       8%
 File Table (nfile)                8202     392       5%       5%
 Shared Mem Table (shmmni)          200       7       4%       4%
 Message Table (msgmni)              50       2       4%       4%
 Semaphore Table (semmni)            64      22      34%      34%
 File Locks (nflocks)              2048       7       0%       0%
 Pseudo Terminals (npty)             60       2       3%       3%
 Buffer Headers (nbuf)               na    7770       na       na

 System Table                     Avail    Reqs    Used     High

 Shared Memory                   12.5gb   9.7mb
 Message Buffers                   16kb             0kb      0kb
 Inode Cache (ninode)              8192               0        0
 DNLC Cache                        8192

                          Min       Max   Avail    Used     High

 Buffer Cache          6.4mb    32.0mb  32.0mb  32.0mb   32.0mb

                                  Avail    Used     Reserved
                                   Size    Size    Size   Util%

 Swap Space                       334mb    63mb   141mb     42%
```

July 22, 2002

Copyright 2002 Hewlett-Packard Company

Page 124

Notes: The GlancePlus "System Tables Report" screen displays several critical kernel parameters, along with their current utilization rate. In the above screenshot we can see the number of proc table entries in use, the amount of the file table consumed, the current rate of buffer cache usage, etc. Unfortunately we see nothing in this screen about things like kernel thread consumption or callout table usage.



summary of nfs differences between hp-ux 11.0 and 11i

- Default number of biod daemons
  (11.0 = 4   11i = 16)
- Default number of nfsd daemons
  (11.0 = 4   11i = 16)
- Support for AutoFS
  (11.0 – patch   11i – included)
- Support for NFS/TCP
  (11.0 – patch   11i – included)

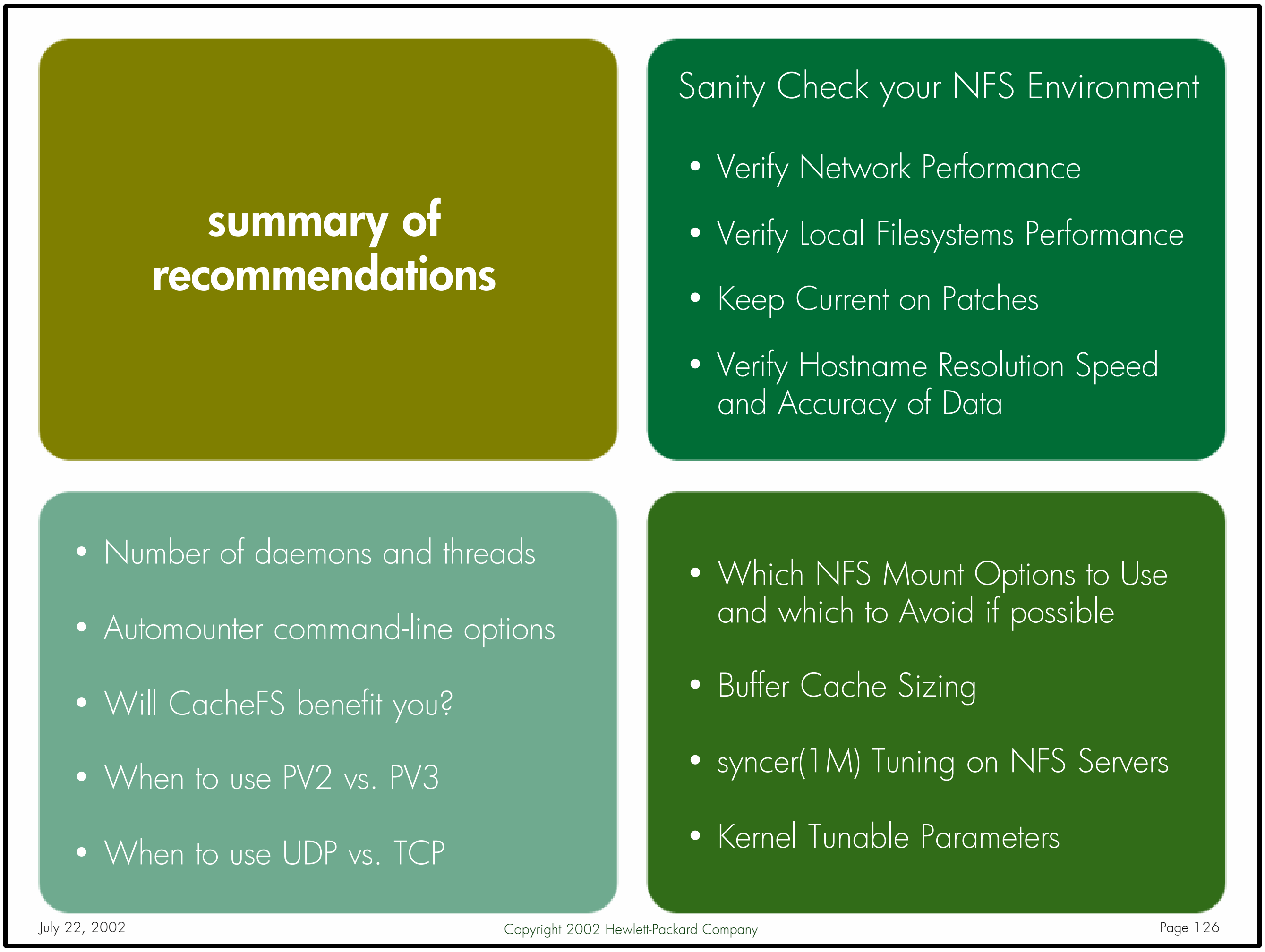
- Filesystem Semaphore Contention drastically reduced in 11i
- Support for large NFS files
  (11.0 – 1TB   11i – 2TB)
- NFS Buffer Cache Management Redesigned in 11i
- CacheFS – only available in 11i
- Kernel Parameter Differences

- Default “proto” NFS mount option
  (11.0 = UDP   11i = TCP)
- Default “rsize” NFS mount option
  (11.0 = 8192   11i = 32768)
- Default “wsize” NFS mount option
  (11.0 = 8192   11i = 32768)
- **`/dev/zero`** file
  (11.0 = mknod   11i = included)

July 22, 2002

Notes: The default behavior of NFS in HP-UX 11i has changed significantly, providing a much better “out of the box” experience than HP-UX 11.0.

While 11.0 NFS clients and servers can be configured to mimic some of the behavioral improvements made in 11i (such as rsize, wsize, and TCP support), HP has no plans to back-port many of the significant 11i performance improvements (including reduced filesystem semaphore contention, redesigned buffer cache management routines, and CacheFS) to HP-UX 11.0.



summary of recommendations

Sanity Check your NFS Environment

- Verify Network Performance
- Verify Local Filesystems Performance
- Keep Current on Patches
- Verify Hostname Resolution Speed and Accuracy of Data

- Number of daemons and threads
- Automounter command-line options
- Will CacheFS benefit you?
- When to use PV2 vs. PV3
- When to use UDP vs. TCP

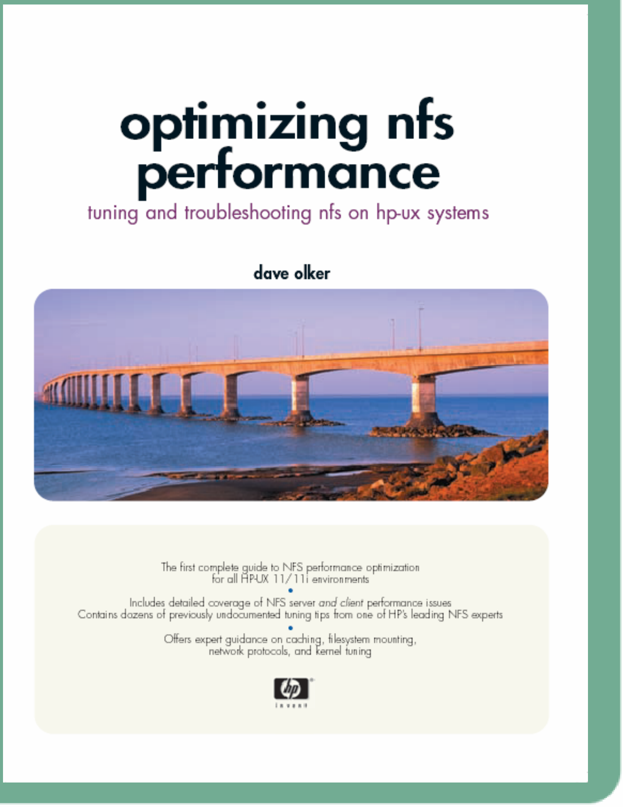
- Which NFS Mount Options to Use and which to Avoid if possible
- Buffer Cache Sizing
- syncer(1M) Tuning on NFS Servers
- Kernel Tunable Parameters

July 22, 2002

Notes: Remember that any suggestions made in this presentation are merely those – **suggestions to begin testing with**. There is not one “golden” set of tuning parameters and configuration settings that will work optimally in every customer’s NFS environment. Only through testing and experimentation can you identify the settings that provide the best NFS performance for your specific mix of applications.

For More Information

- Published by Prentice Hall PTR (Professional Technical Reference) Series
- ISBN 0130428167
- Available In Bookstores September 2002



optimizing nfs performance

tuning and troubleshooting nfs on hp-ux systems

dave olker

The first complete guide to NFS performance optimization
for all HP-UX 11/11i environments

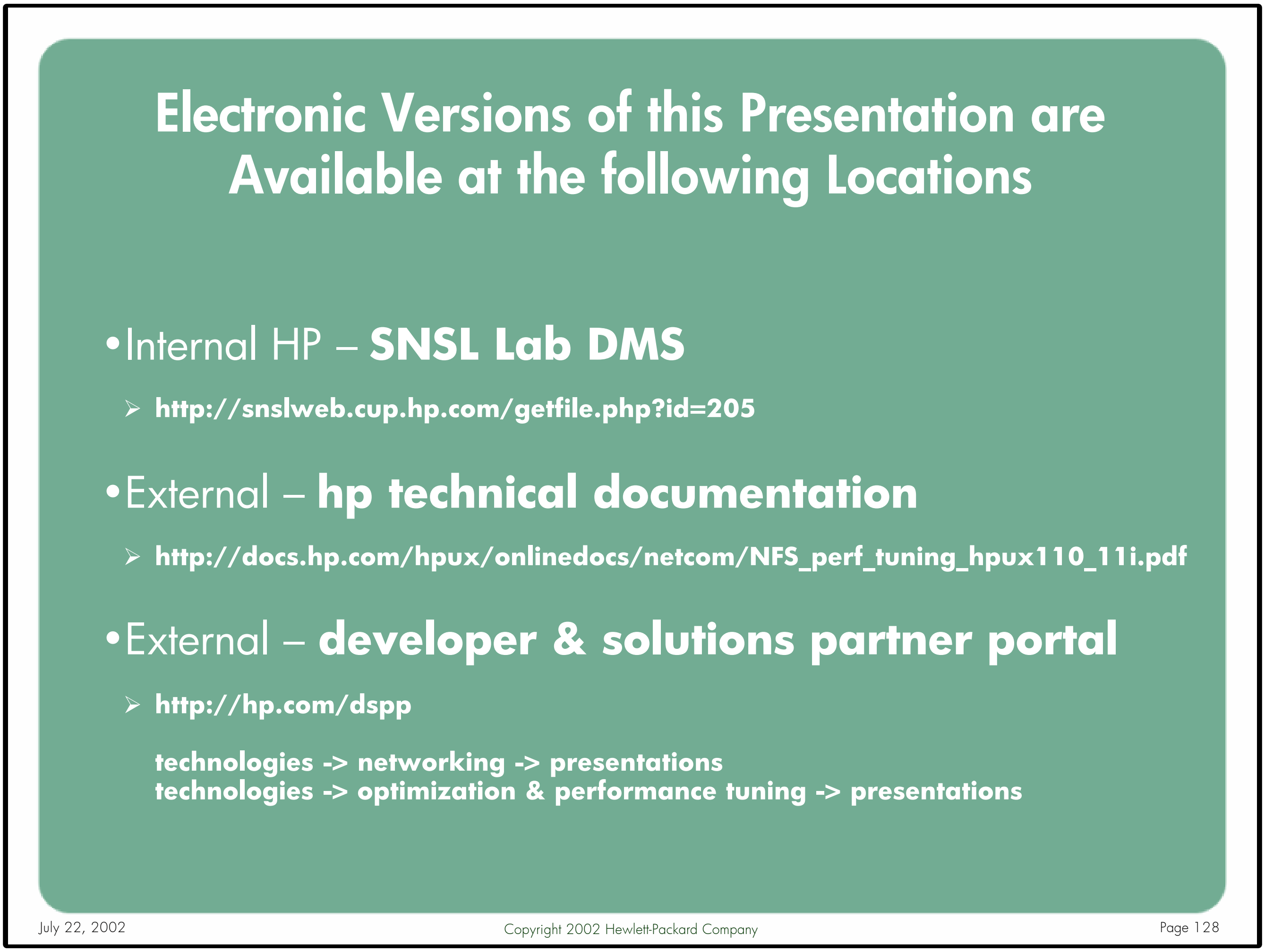
•

Includes detailed coverage of NFS server *and client* performance issues
Contains dozens of previously undocumented tuning tips from one of HP's leading NFS experts

•

Offers expert guidance on caching, filesystem mounting,
network protocols, and kernel tuning

Notes: The Optimizing NFS Performance book contains everything in this presentation and a whole lot more.



Electronic Versions of this Presentation are Available at the following Locations

- Internal HP – **SNSL Lab DMS**
  - **http://snslweb.cup.hp.com/getfile.php?id=205**
- External – **hp technical documentation**
  - **http://docs.hp.com/hpux/onlinedocs/netcom/NFS_perf_tuning_hpux110_11i.pdf**
- External – **developer & solutions partner portal**
  - **http://hp.com/dspp**

    technologies -> networking -> presentations
    technologies -> optimization & performance tuning -> presentations

July 22, 2002

Page 128

Notes: Electronic copies of this presentation are available in PDF format from both internal-HP and external web sites. HP employees can download copies from the SNSL (Systems Networking Solutions Lab) Documentation Management System:

http://snslweb.cup.hp.com/getfile.php?id=205

HP customers can download the latest version of this presentation from HP's Developer & Solutions Partner Portal:

http://hp.com/dspp

Once on the dspp site, click on "technologies," then either "networking" or "optimization & performance tuning," and then "presentations" to find a copy of this presentation.

Alternately, the presentation may be downloaded from HP's Technical Documentation Repository:

http://docs.hp.com/hpux/onlinedocs/netcom/NFS_perf_tuning_hpux110_11i.pdf

Notes:

HP WORLD 2003

Solutions and Technology Conference & Expo

Interex, Encompass and HP bring you a powerful new HP World.

interex

shared knowledge • shared power

encompass

AN HP USER GROUP

hp

invent

Notes: