Using the Linux® NFS Client with Network Appliance™ Filers

Getting the Best from Linux and Network Appliance Technologies

Chuck Lever, Network Appliance | March 12, 2004 | TR 3183

TECHNICAL REPORT

Network Appliance, a pioneer and industry leader in data storage technology, helps organizations understand and meet complex technical challenges with advanced storage solutions and global data management strategies.

Abstract

This report helps you get the best from your Linux NFS clients when used in an environment that includes Network Appliance filers. You will learn what level of performance to expect from your Linux systems. You will learn how to tune your Linux clients and diagnose performance and reliability problems. Finally, you will learn where to look for more information when faced with problems that are difficult to diagnose.

Filer tuning information specific to your application may be available in other Network Appliance technical reports.

This document is appropriate for customers, systems engineers, technical marketing engineers, and customer support personnel who install and configure Linux systems that use NFS to access Network Appliance filers and network caches.

Network Appliance Inc.

Table of Contents

Typographic Conventions

Linux and filer commands and file names appear in `Courier New`.

Summary information appears in red italicized type at the end of each section, and an executive summary appears at the end of the document.

Introduction

More and more Network Appliance customers recognize the value of Linux in their enterprises. Historically, the Linux NFS client has trailed the rest of Linux in providing the level of stability, performance, and scalability that is appropriate for enterprise workloads. In recent times, however, the NFS client has improved considerably and continues to improve in performance and ability to work under degraded network conditions.

Technical support for the Linux NFS client has also become more sophisticated, although it is not yet at an acceptable level for what can be a critical part of enterprise infrastructure. Because more and more of our customers depend on Linux NFS clients, we provide this document to help bridge the gap.

This document addresses several areas that concern those who are planning a new Linux deployment or are administering an existing environment that contains Linux NFS clients accessing Network Appliance filers. These areas include:

- What level of performance and stability to expect from Linux NFS clients
- How to tune Linux NFS clients to perform well with filers
- How to diagnose client and network problems that involve Linux NFS clients
- Which network interfaces and drivers work best
- How to configure other services required to provide advanced NFS features
- Where to find more tuning and diagnostic information on NOW™ (NetApp on the Web) and the Internet

Network Appliance does not recommend specific Linux kernel releases or distributions. First, there are too many distributions and releases to qualify them all. There are more than a half-dozen distributions and thousands of different kernel releases. Add to that the many different versions of user-level helper applications (such as the `mount` command). Because all of these are open source, you can modify or replace any part of your client.

Second, many hardware and application vendors specify a small set of releases or a single release and distribution that are certified and supported. It would be confusing for us to recommend one particular kernel or distribution when your hardware vendor recommends another, and your application vendor specifies yet a third.

Finally, some applications are more sensitive to NFS client behavior than others. Recommending a particular Linux NFS client depends on the applications you want to run and what your performance and reliability requirements are.

Therefore, instead of recommending one or two releases that work well, we provide some guidelines to help you decide among the many Linux distributions and releases, and we provide advice on how to make your Linux NFS clients work their best.

Which Linux NFS Client Is Right for Me?

Before we begin our focus on technical issues, we cover some basic technical support challenges specific to Linux. The Linux NFS client is part of the Linux kernel. Because Linux is open source, you might think that it is easy to provide Linux kernel patches to upgrade the NFS client. In fact, providing a patch that fixes a problem or provides a new feature can be complicated by several facts of life in the Linux and open source worlds.

There are many different parts to Linux, but the two we are concerned about are the distribution and the kernel. The distribution is the set of base operating system files that are included when customers install Red Hat or SuSE on their hardware. This includes commands, applications, and configuration files. A kernel comes with a distribution, but customers can replace it, usually without affecting the other files provided in the distribution.

Identifying Kernel Releases

Linus Torvalds leads a worldwide team that develops the Linux kernel. Linux distributors such as Red Hat and SuSE do not control most of the development team, although Red Hat and SuSE employ many of the best-known kernel contributors. The kernel is released independently from Linux distributions. This means that kernel patches, fixes, and new features may be difficult for a distributor to control.

Each distributor chooses a particular kernel release (for example, 2.4.18) and adds some modifications of its own before placing the kernel in a distribution. To reduce the amount of variation they encounter in their support contracts, distributors support only a small set of kernels, most of which are carefully designed for a specific distribution set.

Because the NFS client is part of the kernel, updates to the NFS client require that you replace the kernel. Technically, it is easy to replace a kernel after a distribution is installed, but Linux customers risk losing distributor support for their installation if they install a kernel that was not built by the distributor. For this reason, Network Appliance can recommend specific patches or kernel versions, but often support contracts constrain customers so they cannot install a patch until their chosen distributor provides a supported kernel that includes the recommended patch.

Torvalds believes in publishing new kernel releases as often as possible. This means that sometimes a kernel is released that contains a needed fix in one part of the code, but also has new bugs in other parts, making the new kernel unsuitable for use in production environments.

The kernel, therefore, is independent of the distribution, so it is numbered differently. The current kernel is released in two branches, known as the “stable” branch and the “development” branch. The stable branch is ostensibly the branch that is hardened, reliable, and has unchanging program interfaces, while the development branch can be (and often is) unstable and sometimes is even unbootable. Stable branches have even minor release numbers, such as 2.4, while development branches have odd minor release numbers, such as 2.5.

The previous stable branch is 2.2, the latest stable branch is 2.4, and the latest development branch is 2.5, which is, even now, becoming the 2.6 stable kernel. Each distribution is numbered very differently: the current Red Hat release is 9.0, for example, and uses kernel 2.4.20. In fact, Red Hat appends another number on the end of its kernel versions to indicate which revision of that kernel is in use. Red Hat shipped kernel 2.4.18-3 with its 7.3 distribution, but made several other errata kernels available over time to fix problems in its kernel: 2.4.18-10, 2.4.18-17, 2.4.18-18, and so on.

One more thing: Linux kernels are not published on a time-based schedule. You will never hear anyone discuss a Linux kernel release date, because kernels are released when the branch maintainer decides they are ready. New features and API changes are allowed in development kernels, but there is no schedule for when such a kernel will become a stable release. Development branches have historically taken two years to 30 months to become stable branches. It is for this reason that there is often significant pressure to add new features to stable releases.

With all that behind us, let us take a look at what features are available in the Linux NFS client in the latest stable releases.

The NFS Client in the 2.2 Kernel

Today, the previous stable branch is still being maintained, although it gets much less attention than the current stable branch. This kernel is appropriate for customers who have some requirement to stay with an old kernel or old distribution, such as a legacy application that for some reason does not run on 2.4 kernels. There are also rare cases where 2.2 kernels perform better than 2.4 kernels. If these conditions do not apply to you, you should use a member of the 2.4 kernel series. The 2.4 kernels have a wider variety of drivers and new features, and bugs in these kernels are actively addressed.

The NFS client in kernels later than 2.2.19 supports NFSv2 and NFSv3 and supports NFS over both UDP and TCP. Clients in earlier releases of the 2.2 branch did not support NFS over TCP. This NFS client is not as stable, nor does it perform as well as the NFS client in the 2.4 branch. However, it should work well enough for most small to moderate workloads. TCP support is less mature than UDP support in the 2.2 branch, so using UDP may provide higher reliability.

The NFS Client in the 2.4 Kernel

The NFS client in this kernel has many improvements over the 2.2 client, most of which address performance and stability problems. The NFS client in kernels later than 2.4.16 has significant changes to help improve performance and stability.

Some customers cannot upgrade their kernel to a version not supported by their chosen distributors or hardware vendors. Moreover, some customers do not feel comfortable using a kernel on production systems they built themselves. If you are comfortable building your own kernels for your production systems, use the latest kernel from kernel.org you can find. If you are comfortable using kernels prebuilt by your chosen distributor, you should choose the latest errata kernel supported for your distribution. Whether you build your own kernel or use prepackaged kernels, you should use the mount options described in the following section for best performance and reliability.

Support for NFS over UDP in late 2.4 kernels is quite mature, while support for NFS over TCP is somewhat less mature. On the other hand, NFS over UDP performance suffers more on lossy networks. You should experiment to see which suits your performance and reliability requirements. The TCP protocol is more reliable than UDP in the face of network problems, and the reliability and performance of NFS over TCP in Linux will only improve as time goes on. Remember to report any problems so the community can benefit from fixes as soon as possible.

Trond Myklebust, the Linux NFS client maintainer, makes beta patches available that address recently found problems. They are usually not appropriate for production environments. However, a report to the maintainer of a successful resolution of your problem might expedite the patch into tested and distributed kernels. To find out more about what patches are available, see this document’s appendix for Web site information.

Customers that use 2.4 kernels on hardware with more than 896MB should know that a special kernel compile option, known as CONFIG_HIGHMEM, is required for the system to access and use physical memory above 896MB. The Linux NFS client has a known problem in these configurations where an application or the whole client system can hang at random. This issue has been addressed in the 2.4.20 kernel, but still haunts kernels contained in distributions from Red Hat and SuSE that are based on earlier kernels.

Early releases of Red Hat 7.3 contained a kernel that demonstrated very poor NFS client performance when mounting with UDP. Recent errata kernels that fix some of the performance problems are available from Red Hat. Network Appliance recently published an article describing this problem in detail. See document ntapcs6648 on NOW (the URL is in the appendix).

The NFS Client in the 2.6 Kernel

Recently, an early test release of the Linux 2.6 kernel became available. This kernel is not ready for production use, but over the next year we expect it to become stable enough to be employed in standard Red Hat and SuSE distributions. Until it does, you should assume that it is useful for nonproduction workloads, where reliability and performance are not necessarily concerns. In other words, even though 2.6 is an even-number kernel release, it is not yet stable enough for production workloads.

A new feature in the 2.6 kernel is support for the latest version of the NFS protocol, version 4. Developers are still in the process of retrofitting the Linux NFS client and server implementations for the new protocol version. Certain features are available today in 2.6 kernels, but others, such as read and write delegation and replication and migration support, are still under development and are not yet available. Support for NFS version 4 in user-level utilities is also not yet available in common Linux distributions.

The 2.6 kernel also brings support for advanced authentication mechanisms such as Kerberos 5. Support for Kerberos will be available for NFS versions 2 and 3 as well as for NFS version 4. Kerberos authentication increases security by reducing the likelihood that user identities in NFS requests can be forged. It also provides optional facilities to ensure the integrity or privacy of communication between an NFS client and server.

In summary: You should use the latest kernel available from your distributor. Always check with your hardware and application vendors to be certain they support the kernel you choose to run. Contact us if you have any special requirements.

Foolproof Mount Options for Linux NFS Clients

If you have never set up mount options on an NFS client before, review the “nfs” man page on Linux to see how these terms are defined. You can type `man nfs` at a shell prompt to display the page. In addition, O’Reilly’s “Managing NFS & NIS” covers many of these topics (see the appendix for URL and ISBN information).

You can look in `/etc/fstab` on the client to see what options the client attempts to set when mounting a particular file system. Check your automounter configuration to see what defaults it uses when mounting. Running the `mount` command at a shell prompt tells you what options are actually in effect. Clients negotiate some options, for example, the `rsize` option, with servers. Look in the client’s `/proc/mounts` file to determine exactly what mount options are in effect for an existing mount.

The default NFS protocol version (2 or 3) used when mounting an NFS server can change depending on what protocols the server exports, which version of the Linux kernel is running on the client, and what version of the mount utilities package is in use. Version 2 of NFS is the default. To ensure that your client uses the NFSv3 protocol, you should specify “vers=3” when mounting a filer. Be sure that the NFSv3 protocol is enabled on your filer before trying to mount using “vers=3.”

The “hard” mount option is the default on Linux and is mandatory if you want data integrity. Using the “soft” option reduces the likelihood of client instability during server and network outages, but it exposes your applications to silent data corruption, even if you mount file systems read-only. If a soft timeout interrupts a read operation, the client’s cached copy of the file is probably corrupt. To purge a corrupted file requires that some application locks and unlocks the file, that the whole file system is unmounted and remounted, or that another client modifies the file’s size or mtime. If a soft timeout interrupts a write operation, there is no guarantee that the file on the server is correct, nor is there any guarantee that the client’s cached version of the file matches what is on the server.

A client can indicate that a soft timeout has occurred in various ways. Usually system calls return EIO when such a timeout occurs. You may also see messages in the kernel log suggesting that the client had trouble maintaining contact with the server and has given up. If you see a message that says the client is still trying, then the “hard” mount option is in effect.

As an alternative to soft mounting, consider using the “intr” option, which allows users and applications to interrupt the NFS client when it gets stuck waiting for server or network recovery. On Linux, interrupting applications or mount commands does not always work, so sometimes rebooting your client is necessary to recover a mount point that has become stuck because the server is not available.

When running applications such as databases that depend on end-to-end data integrity, you should use “hard,nointr.” Oracle has verified that using intr instead of nointr can expose your database to the risk of corruption when a database instance is signaled (for example, during a “shutdown abort” sequence).

The “soft” option is useful only in a small number of cases. If you expect significant server or network instability, try using the soft option with TCP to help reduce the impact of temporary problems. When using the soft option, especially with UDP, set a long retransmission timeout and a relatively large number of retries. This reduces the likelihood that very brief outages or a few dropped packets will cause an application failure or data corruption.

Choosing a Network Protocol

UDP is a simple protocol that has less CPU and network overhead than TCP. In addition, the Linux NFS over UDP implementation is more mature than the NFS over TCP implementation. As time goes on, the robustness and maturity of the Linux NFS over TCP implementation will improve. NFS over UDP has deficiencies that are exposed on congested networks, such as routed multispeed networks, DSL links, and slow WANs, and should not be used in those environments. Unless you need the very last bit of performance from your network, TCP is usually a safe bet, especially on more recent versions of Linux. Future versions of the NFS protocol may not support UDP at all.

NFS over TCP can handle multispeed networks (networks where the links connecting the server and the client use different speeds), higher levels of packet loss and congestion, and widely varying network and server latency, but can cause long delays during server recovery. Note that older releases of Data ONTAP™ do not enable NFS over TCP by default. From Data ONTAP release 6.2 onward, TCP connections are enabled automatically on new filer installations.

When using Linux kernel 2.4.22 or later, start with NFS over TCP. Although TCP has slightly greater network and CPU overhead on both the client and server, you will find that NFS performance on TCP remains stable across a variety of network conditions and workloads. Earlier kernels may work with TCP; however, you may need to resort to using UDP if you find there are stability or recovery problems that prevent TCP from working. The 2.4.22 kernel also has improved retransmit timeout estimation, which helps NFS over UDP work better over congested networks.

If you find UDP suits your needs better and you run kernels older than 2.4.22, be sure to enlarge your client's default socket buffer size by following the instructions listed in the appendix of this guide. Bugs in the IP fragmentation logic in these kernels can cause a client to flood the network with unusable packets, preventing other clients from accessing the filer. The Linux NFS client is especially sensitive to IP fragmentation problems that can result from congested networks or undersized switch port buffers. If you think IP fragmentation is an issue for your clients using NFS over UDP, the "netstat –s" command on the client and on the filer displays running counts of fragmentation errors. Be certain to apply this change to all Linux NFS clients on your network in order for the change to be completely effective.

For databases, we recommend NFS over TCP. There are rare cases where NFS over UDP on noisy or very busy networks can result in silent data corruption. In addition, Oracle9*i*™ RAC is certified on NFS over TCP running on Red Hat Advanced Server 2.1.

You can control RPC retransmission timeouts with the "timeo" option. Retransmission is the mechanism by which clients ensure a server receives and processes an RPC request. If the client does not receive a reply for an RPC within a certain interval for any reason, it retransmits the request until it receives a reply from the server. After each retransmission, the client doubles the retransmit timeout up to 60 seconds to keep network load to a minimum.

The Linux NFS client treats RPC timeouts differently, depending on which transport protocol you choose. By default, the client retransmits an unanswered UDP RPC request after 0.7 seconds. In general, it is not necessary to change the retransmission timeout for UDP, but in some cases, a shorter retransmission timeout for NFS over UDP may improve latencies due to packet losses. As of kernel 2.4.20, an estimation algorithm that adjusts the timeout for optimal performance governs the UDP retransmission timeout for some types of RPC requests.

Retransmission for NFS over TCP works somewhat differently. The TCP network protocol contains its own timeout and retransmission mechanism. The RPC client depends on this mechanism for recovering from the loss of RPC requests and thus uses a much longer timeout setting for NFS over TCP by default. Due to a bug in the `mount` command, the default retransmission timeout value on Linux for NFS over TCP is six seconds, unlike other NFS client implementations. To obtain standard behavior, you may wish to specify "timeo=600,retrans=2" explicitly when mounting via TCP. Unlike with NFS over UDP, using a short retransmission timeout with NFS over TCP does not have performance benefits and may increase the risk of data corruption.

The Linux NFS client quietly retransmits RPC requests several times before reporting in the kernel log that it has lost contact with an NFS server. You can control how many times the client retransmits the same request before reporting the loss using the "retrans" mount option. Remember that whenever the hard mount option is in effect, an NFS client never gives up retransmitting an RPC until it gets a reply. Be careful not to use the similar-sounding "retries" mount option, which controls the number of times the client retries a mount request before it fails.

In summary, "timeo=600,retrans=2" is appropriate for TCP mounts. When using NFS over UDP, "timeo=4,retrans=5" is a better choice. Using "timeo=600" with NFS over UDP will result in very poor performance in the event of network or server problems. Using a very short timeout with TCP could cause network congestion or, in rare cases, data corruption.

Capping the Size of Read and Write Operations

In Linux, the "rsize" and "wsize" mount options have additional semantics compared with the same options as implemented in other operating systems. Normally these options determine how large a network read or write operation can be before the client breaks it into smaller operations. Low rsize and wsize values are appropriate when adverse network conditions prevent NFS from working with higher values or when NFS must share a low-bandwidth link with interactive data streams.

By default, NFS client implementations choose the largest rsize and wsize values a server supports. However, if you do not explicitly set rsize or wsize when you mount an NFS file system on a Red Hat host, the default value for both is a modest 4,096 bytes. Red Hat chose this default because it allows the Linux NFS client to work without adjustment in most environments. Usually on clean high-performance networks, or with NFS over TCP, you can improve overall NFS performance by explicitly increasing these values.

Normally, the Linux client attempts to delay application write requests to hide their latency from applications, and it issues network read operations all at once before waiting for the server to reply. If the "rsize" option is set below the system's page size (4K on x86 hardware), the client issues individual read operations one at a time and waits for each operation to complete before issuing the next read operation. If the "wsize" option is set below the system's page size, the client issues synchronous writes without regard to the use of the "sync" or "async" mount options. As with reads, synchronous writes cause applications to wait until the NFS server completes each individual write operation before issuing the next operation or before letting an application continue with other processing. When performing synchronous writes, the client waits until the server has written its data to stable storage before allowing an application to continue.

Some hardware architectures allow a choice of different page sizes. Intel® Itanium™ systems, for instance, support pages up to 64KB. On a system with 64KB pages, the rsize and wsize limitations described above still apply; thus all NFS I/O is synchronous on these systems, significantly slowing read and write throughput. When running on hardware that supports different page sizes, choose a combination of page size and r/wsize that allows the NFS client to do asynchronous I/O if possible. Usually distributors choose a single large page size, such as 16KB, when they build kernels for hardware architectures that support multiple page sizes.

The network protocol (TCP or UDP) interacts in complicated ways with rsize and wsize. When you encounter poor performance because of network problems, switching to NFS over TCP is a better way to achieve good performance than using small read or write sizes over UDP. The client and server fragment large UDP datagrams, such as single read or write operations more than a kilobyte in size, into individual IP packets. RPC over UDP retransmits a whole RPC request if any part of it is lost on the network, whereas RPC over TCP efficiently recovers a few lost packets and reassembles the complete requests at the receiving end. Thus with NFS over TCP, 32K read and write size usually provides good performance by allowing a single RPC to transmit or receive a large amount of data. With NFS over UDP, 32K read and write size may provide good performance, but often using NFS over UDP results in terrible performance if the network is at all congested. For Linux, a good compromise value when using

NFS over UDP is 8K or less. If you find even that does not work well, and you cannot improve network conditions, we recommend switching to NFS over TCP if possible.

It is very important to note that the capabilities of the Linux NFS server are different from the capabilities of the Linux NFS client. As of the 2.4.19 kernel release, the Linux NFS server does not support NFS over TCP and does not support rsize and wsize larger than 8K. The Linux NFS client, however, supports NFS over both UDP and TCP and rsize and wsize up to 32K. Some online documentation is confusing when it refers to features that “Linux NFS” supports. Usually such documentation refers to the Linux NFS server, not the client.

Special Mount Options

Consider using the “bg” option if your client system needs to be available even if it cannot mount some servers. This option causes mount requests to put themselves in the background automatically if a mount cannot complete immediately. When a client starts up and a server is not available, the client waits for the server to become available by default. The default behavior results in waiting for almost a week before giving up.

The “fg” option is useful when you need to serialize your mount requests during system initialization. For example, you probably want the system to wait for `/usr` to become available before proceeding with multiuser boot. If you mount `/usr` or other critical file systems from an NFS server, you should consider using fg for these mounts.

For security, you can also use the “nosuid” mount option. This causes the client to disable the special bits on files and directories. The Linux man page for the `mount` command recommends also disabling or removing the `suidperl` command when using this option.

It is a common trick for system administrators to set the “noatime” mount option on local file systems to improve disk performance. Because NFS servers, not clients, control the values contained in a file’s timestamps (access time, metadata change time, and data modify time) by default, this trick is not effective for NFS mounts. However, filers allow you to reduce the overhead caused by aggressive atime flushing if you set a volume’s `no_atime_update` option. On a filer console, type “`help vol options`” for details.

Other mount options allow you to tailor the client’s attribute caching and retry behavior. It is not necessary to adjust these behaviors under most circumstances. However, sometimes you must adjust NFS client behavior to make NFS appear to your applications more like a local file system.

There are a few indirect ways to tune client-side caching. First, the most effective way to improve client-side caching is to add more RAM to your clients. Linux will make appropriate use of the new memory automatically. To determine how much RAM you need to add, determine how large your active file set is and increase RAM to fit. This greatly reduces cache turnover rate. You should see fewer read requests and faster client response time as a result.

Some working sets will never fit in a client’s RAM cache. Your clients may have 128MB or 4GB of RAM, for example, but you may still see significant client cache turnover. In this case, reducing “cache miss” latency is the best approach. You can do this by improving your network infrastructure and tuning your server to improve its performance. Because a client-side cache is not effective in these cases, you may find that keeping the client’s cache small is beneficial.

You may also want to improve attribute caching behavior. Due to the requirements of the NFS protocol, clients must check back with the server every so often to be sure cached attribute information is still valid. However, adding RAM on the client will not improve the rate at which the client tries to revalidate parts of the directory structure it has already cached. No matter how much of the directory structure is cached on the client, it must still validate what it knows when files are opened or when attribute cache information expires. You can lengthen the attribute cache timeout with the “actimeo” mount option to reduce the rate at which the client tries to revalidate its attribute cache. With the 2.4.19 kernel release, you can also use the “nocto” mount option to reduce the revalidation rate even further, at the expense of cache coherency among multiple clients.

The Linux NFS client delays application writes to combine them into larger, more efficiently processed requests. You can guarantee that a client immediately pushes every write system call an application makes to servers by using the “sync” mount option. This is useful when an application needs the guarantee that data is safe on disk before it continues. Frequently such applications already use the `O_SYNC` open flag or invoke the flush system call when needed. Thus, the sync mount option is often not necessary.

Normally, for each file in a file system that has been accessed recently, a client caches file attribute information, such as a file’s last modification time and size. To detect file changes quickly yet efficiently, the NFS protocol uses “close-to-open” cache semantics. When a client opens a file, it uses a GETATTR operation to check that the file still exists and any cached data it has is still up-to-date. A client checks back with the server only after a timeout indicates that the file’s attributes may be stale. During such a check, if the server’s version of the attributes has changed, the client purges its cache. A client can delay writes to a file indefinitely. When a client closes a file, however, it flushes all pending modifications to the file to the server. This allows a client to provide good performance in most cases, but means it might take some time before an application running on one client sees changes made by applications on other clients.

Delayed writes and the client’s attribute cache timeout can delay detection of changes on the server by many seconds while a file is open. The “noac” mount option prevents the client from caching file attributes. This means that every file operation on the client that requires file attribute information results in a GETATTR operation to retrieve a file’s attribute information from the server. Note that noac also causes a client to process all writes to that file system synchronously, just as the sync mount option does. Disabling attribute caching is only one part of noac; it also guarantees that data modifications are visible on the server so that other clients using noac can detect them immediately. Thus noac is shorthand for “actimeo=0,sync.”

When the noac option is in effect, clients still cache file data as long as they detect that a file has not changed on the server. This allows a client to keep very close track of files on a server so it can discover changes made by other clients quickly. Normally you will not use this option, but it is important when an application that depends on single system behavior is deployed across several clients.

Using the noac mount option typically causes a 30% to 40% performance degradation. Noac generates a very large number of GETATTR operations and sends write operations synchronously. Both of these add significant protocol overhead. The noac mount option trades off single-client performance for client cache coherency. Only applications that need tight cache coherency among multiple clients require that file systems be mounted with the noac mount option.

Some applications require direct, uncached access to data on a server. Using the noac mount option is sometimes not good enough, because even with this option, the Linux NFS client still caches reads. To ensure your application sees the server's version of a file's data and not potentially stale data cached by the client, your application can lock and unlock the file. This pushes all pending write operations back to the server and purges any remaining cached data, so the next read operation will go back to the server rather than reading from a local cache.

Alternatively, the Linux NFS client in the RHEL 3.0 and SuSE SLES 8 SP3 kernels supports direct I/O to NFS files when an application opens a file with the O_DIRECT flag. Direct I/O is a feature designed to benefit database applications that manage their own data cache. When this feature is enabled, an application's read and write system calls are translated directly into NFS read and write operations. The Linux kernel never caches the results of any read or write when a file is opened with this flag, so applications always get exactly what's on the server. Because of I/O alignment restrictions in the Linux O_DIRECT implementation, applications must be modified to support direct I/O properly. See the appendix for more information on this feature.

For some servers or applications, it is necessary to prevent the Linux NFS client from sending Network Lock Manager requests. You can use the "nolock" mount option to prevent the Linux NFS client from notifying the server's lock manager when an application locks a file. Note, however, that the client still uses more restrictive write back semantics when a file lock is in effect. The client always flushes all pending writes whenever an application locks or unlocks a file.

For detailed information on configuring and tuning filers in an Oracle® environment, see the Network Appliance tech reports at *www.netapp.com*.

Unmounting NFS File Systems

This section discusses the unique subtleties of unmounting NFS file systems on Linux.

Like other *NIX operating systems, the `umount` command detaches one or more file systems from a client's file system hierarchy (like several other common features of *NIX, the name of this command is missing a letter). Normally, you use `umount` with no options, specifying only the path to the root of the file system you want to unmount.

Sometimes you might want more than a standard unmount operation—for example, when programs appear to be waiting indefinitely for access to files in a file system or if you want to clear the client's data cache.

If you want to unmount all currently mounted NFS file systems, use:

```
umount -a -t nfs
```

Sometimes unmounting a file system becomes stuck. For this, there are two options:

```
umount -f /path/to/filesystem/root
```

This forces an unmount operation to occur if the mounted NFS server is not reachable, as long as there are no RPC requests on the client waiting for a reply. After kernel 2.4.11,

```
umount -l /path/to/filesystem/root
```

usually causes the kernel to detach a file system from the client's file system hierarchy immediately, but allows it to clean up RPC requests and open files in the background.

As mentioned above, unmounting an NFS file system does not interrupt RPC requests that are awaiting a server reply. If an `umount` command fails because processes are waiting for network operations to finish, you must interrupt each waiting process using ^C or an appropriate `kill` command. Stopping stuck processes usually happens automatically during system shutdown so that NFS file systems can be safely unmounted. The NFS client allows these methods to interrupt pending RPC requests only if the intr option is set for that file system. To identify processes waiting for NFS operations to complete, use the `lsof` command.

The `umount` command accepts file system–specific options via the “ o” flag. The NFS client does not have any special options.

Mount Option Examples

We provide the following examples as a basis for beginning your experimentation. Start with one of these examples that closely matches your scenario, then thoroughly test the performance and reliability of your application while adjusting the mount options.

If you do not specify any mount options, the Linux mount command (or the automounter) automatically chooses these defaults:

```
mount –o rw,fg,vers=2,udp,rsize=4096,wsize=4096,hard,intr,
  timeo=7,retrans=5
```

These default settings are designed to make NFS work right out of the box in most environments. Almost every NFS server supports NFS version 2 over UDP. Rsize and wsize are relatively small because some network environments fragment large UDP packets, which can hurt performance if there is a chance that fragments can be lost. The RPC retransmit timeout is set to 0.7 seconds by default to accommodate slow servers and networks.

On clean single-speed networks, these settings are unnecessarily conservative. Over some firewalls, UDP packets larger than 1,536 are not supported, so these settings do not work. On congested networks, UDP may have difficulty recovering from large numbers of dropped packets. NFS version 2 write performance is usually slower than NFS version 3. As you can see, there are many opportunities to do better than the default mount options.

Here is an example of mount options that are reasonable defaults:

```
mount –o rw,bg,vers=3,tcp,timeo=600,rsize=8192,wsize=8192,hard,intr
```

Using the bg option means our client will be able to finish booting without waiting for filers that may be unavailable because of network outages. The hard option minimizes the likelihood of data loss during network and server instability, while intr allows users to interrupt applications that may be waiting for a response from an unavailable server. The tcp option works well on many typical LANs with 8K read and write size. Using “timeo=600” is a good default for TCP mounts, but for UDP mounts, “timeo=4” might be more appropriate.

Here is an example of a poor combination of mount options:

```
mount –o rw,soft,udp,rsize=1024,wsize=1024
```

Using soft mounts with UDP is a recipe for silent data corruption. On Linux with these mount options it is made worse. On Linux clients “wsize=1024” mandates synchronous writes; writes go to the server one at a time rather than in groups, and the client requires the server to push data onto disk before responding to the client's write operation request. If a server gathers writes to improve disk bandwidth, it delays its response to each write request waiting for more write requests, which can trigger the client to retry write requests. A server that delays responding to writes as long as several hundred milliseconds will probably cause the client to drop requests unnecessarily after several retries (note that filers usually do not delay writes because they can cache these requests in nonvolatile RAM to fulfill NFSv2's requirement for stable writes). To address these issues, always use the hard option and use read and write sizes larger than your client's page size.

When mounting a group of home directories over a WAN, you might try:

```
mount –o rw,bg,vers=3,nosuid,tcp,timeo=600,retrans=2,rsize=2048,
  wsize=2048,soft,intr
```

This example uses NFS over TCP because NFS clients often reside on slower, less capable networks than servers. In this case, the TCP protocol can provide fast recovery from packet losses caused by network speed transitions and noisy phone lines. Using the nosuid mount option means users cannot create or use suid programs that reside in their home directories, providing a certain degree of safety from Trojan horses. Limiting the maximum size of read and write operations gives interactive sessions on slow network links an advantage by keeping very large packets off the wire. On fast networks, large rsize and wsize values, such as 32,768, are more appropriate. The soft option helps to recover quickly from server or network outages with a minimal risk of possible data loss, and the "timeo=600" option allows the TCP protocol a long time to attempt recovery before the RPC client interferes.

When mounting a filer from an anonymous FTP or HTTP server, you could use:

```
mount –o ro,fg,vers=3,tcp,timeo=600,retrans=2,rsize=32768,wsize=32768,
  hard,nointr,nolock,nocto,actimeo=600
```

Here we use the fg option to ensure that NFS files are available before the FTP or HTTP server is started. The ro option anticipates that the FTP or HTTP server will never write data into files. File locking is also unlikely to be necessary, so we use the nolock option to eliminate the overhead of NLM. The nocto option helps reduce the number of GETATTR and LOOKUP operations at the expense of tight cache coherency with other clients. The FTP server will see changes to files on the server after its attribute cache times out (usually after about one minute). Lengthening the attribute cache timeout also reduces the attribute cache revalidation rate.

When mounting a filer for use with a single-instance Oracle Database system over a clean Gigabit Ethernet, you might try:

```
mount –o rw,fg,vers=3,tcp,timeo=600,retrans=2,hard,nointr
```

Again, the fg option ensures that NFS file systems are available before the database instance starts up. We use TCP here because even though the physical network is fast and clean, TCP adds extra data integrity guarantees. The hard option ensures data integrity in the event of network problems or a cluster failover event. The nointr option prevents signals from interrupting NFS client operations. Such interruptions may occur during a shutdown abort, for instance, and are known to cause data corruption. File locking should be enabled when running databases in production as a degree of protection against corruption caused by improper backup procedures (for example, another instance of the same database running at a disaster recovery site against the same files as your normal production instance).

When setting up an Oracle9*i* RAC node on Linux, try these mount options first:

```
mount -o rw,fg,vers=3,tcp,timeo=600,retrans=2,hard,nointr,noac
```

These are the same as the single-instance mount options, with two differences. For Oracle RAC, you also need to add the “noac” option. Oracle RAC requires the noac option so that clients in a database cluster maintain a coherent image of the data files on the storage servers.

See the appendix for more information on how to set up databases on Linux, including details on how to adjust the Linux read-ahead algorithm.

In summary: Use NFS version 3 if possible. Use NFS over TCP wherever possible. If NFS over UDP is slow or hangs, this is a sign of network problems, so try TCP instead. Avoid using the soft mount option. Try the special mount options if you need an extra boost in performance.

Performance

This section covers aspects of Linux client performance, with a special focus on networking.

Setting Performance Expectations

The Linux NFS client runs in many different environments, from light desktop usage to database with a dedicated private SAN. In general, the Linux client can perform as well as most other clients, and better than some, in these environments. However, you must plan your mount options and observe network behavior carefully to ensure the Linux client can perform at its best.

On low-speed networks (10Mb/sec or 100Mb/sec), the Linux NFS client can read and write as fast as the network allows. This means Linux NFS clients running anything faster than a 400 MHz processor can saturate a 100Mb link. Slower network interfaces (for example, 16-bit PCMCIA cards on laptops) noticeably reduce client NFS bandwidth. Be sure your clients have enough CPU to drive the network while handling your application workload.

If your clients use high-performance networking (gigabit or faster), you should plan to provide enough CPU and memory bandwidth on your clients to handle the interrupt and data rate. The NFS client software and the gigabit driver cut into resources available to applications, so make sure there are enough to go around. Most gigabit cards that support 64-bit PCI or better should provide good performance. We have found these cards work well with Linux:

- SysKonnect—The SysKonnect SK-98XX series cards work very well with Linux and support single- and dual-fiber and copper interfaces for better performance and availability. A mature driver for this card exists in the 2.4 kernel source distribution.
- Broadcom—Many cards and switches use this chipset, including the ubiquitous 3Com solutions. This provides a high probability of compatibility between your switches and your clients. The driver software for this chipset appeared in the 2.4.19 Linux kernel and is included in Red Hat distributions with earlier 2.4 kernels. Be sure the chipset firmware is up to date.
- AceNIC Tigon II—Several cards, such as the NetGear GA620T, use this chipset, but none of them are made any longer. A mature and actively maintained driver for this chipset exists in the kernel source distribution.
- Intel EEPro/1000—This appears to be the fastest gigabit card available for Intel-based systems, but the card’s driver software is included only in recent kernel source distributions (2.4.20 and later) and may be somewhat unstable. You can find the card’s driver software for

earlier kernels at Intel's Web site. There are reports that the jumbo frame MTU for Intel's cards is only 8,998 bytes, not the standard 9,000 bytes.

All of these cards support Gigabit Ethernet's jumbo frames option. If you use Linux NFS clients and filers together on an unrouted network, consider using jumbo frames to improve the performance of your application. Be sure to consult your switch's command reference to make sure it is capable of handling jumbo frames in your environment. There are some known problems in Linux drivers and the networking layer when using the maximum frame size (9,000 bytes). If you experience unexpected performance slowdowns when using jumbo frames, try reducing the MTU to, say, 8,960 bytes.

SysKonnect now publishes an independent Gigabit Ethernet NIC shootout on its Web site. See the appendix for information on how to obtain this document.

For most purposes, Gigabit Ethernet over copper works about as well as Gigabit Ethernet over fiber for short-distance links. Category 5E or Category 6 cables are necessary for reliable performance on copper links. Fiber adds long-haul capabilities and even better reliability, but at a significant cost. Some find that copper terminations are more rugged and reliable than fiber terminations.

The Linux NFS client and network layer are very sensitive to network performance and reliability. After you have set your mount options as we recommend, you should get reasonable performance. If you do not and your workload is not already CPU-bound, you should begin looking at network conditions between your clients and servers.

For example, on a clean Gigabit Ethernet network, a single Linux client can write to an F880 filer as fast as the filer can put data on disk. If there is other network traffic or packet loss, write performance from a Linux client on NFS over UDP can drop rapidly, though on NFS over TCP, performance should remain reasonable. Read performance depends on the size and speed of the client's memory.

The most popular choices for 100BaseT network interfaces are the Intel EEPro/100 series and the 3Com 3C905 family. These are often built into server mainboards and can support network booting via PXE, described in more detail later in this document. These cards perform well in general, but some versions of their driver software have known bugs. Check the Red Hat errata database for more information. Recent versions should work fairly well.

The original driver for Intel EEPro/100 series cards, which has resided in Linux kernels for some time, now has competition from a driver written by developers at Intel. This alternative driver was recently added to 2.4.20.

When choosing a 100BaseT card for your Linux systems, look for a card that has a mature driver that is already integrated into the Linux kernel. Features such as checksum offloading are beneficial to performance. You can use Ethernet bonding, or trunking, to improve the reliability or performance of your client.

Most network interface cards use autonegotiation to obtain the fastest settings allowed by the card and the switch port to which it attaches. Sometimes, PHY chipset incompatibilities may result in constant renegotiation or negotiating half duplex or a slow speed. When diagnosing a network problem, be sure your Ethernet settings are as you expect before looking for other problems. To solve an autonegotiation problem, you may be inclined to hard code the settings you want, but avoid this because it only masks a deeper problem. Work with your switch and card vendors to resolve these problems.

In summary: Whether you use TCP or UDP, be sure you have a clean network. Ensure your network cards always negotiate the fastest settings. Increasing the size of the client's socket buffer (see appendix) is always a wise thing to do.

Linux NFS Client Architecture

There are five layers of software between your application and the network. First, the generic VFS layer in Linux adapts system calls made by your application to generic interface calls supported by all file systems. It also manages the main file system caches, including the data cache and the attribute caches.

The second layer is the NFS client itself, which adapts the generic file system interface calls into NFS RPC requests to a server. The client is responsible for converting the local environmental conditions of the Linux kernel to the NFS protocol and back. It generates RPC requests that it hands to the RPC client.

The RPC client is the third layer. It takes whole NFS RPC requests from the client and converts them into socket calls. It marshals and unmarshals data, manages byte ordering, waits for server replies, and runs extra tasks, such as the flusher daemon, when necessary.

The next lower layer is the Linux network layer, which handles TCP, UDP, and IP processing. The bottom layer on the client contains the network interface device driver and kernel interrupt scheduling logic.

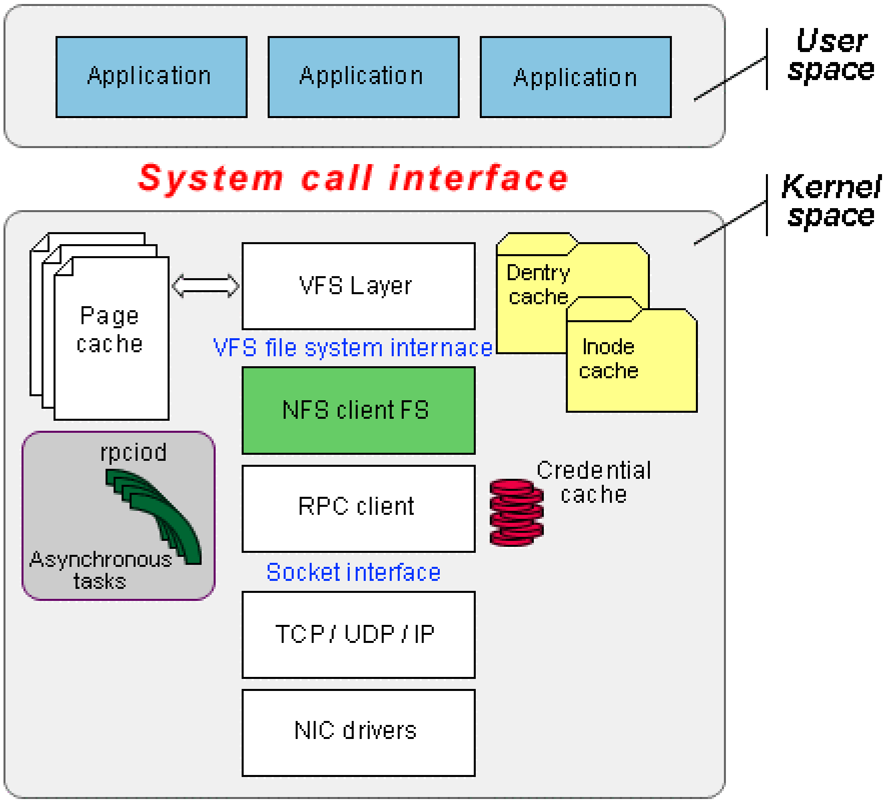

Figure 1) Linux NFS client schematic diagram.
There are five individual layers of software between an application and the network. These manage data and attribute caches, asynchronous tasks, and network protocol translation.

This architecture is unlike that of some other NFS client implementations, which do not separate their NFS file system implementation from the RPC client. Other implementations use a transport-independent RPC implementation that is less efficient than using sockets directly, as Linux does.

The stability and performance of what you know as the Linux NFS client depend on the ability of all of these layers to function well. Very often, the NFS client layer functions well and properly, but some other part of the kernel, such as the Linux TCP/IP implementation or page cache, or something outside your Linux client such as your network topology, causes a problem that appears as though your NFS client is misbehaving. When diagnosing client problems, you should always keep in mind that there are many possible ways for a complex system such as this to misbehave.

Diagnosing Performance Problems with the Linux NFS Client

Now that you understand how the client is structured, let us review specific methods and tools for diagnosing problems on the client. The client works best when the network does not drop any packets. The NFS and RPC clients also compete with applications for available CPU resources. These are the two main categories of client performance problems you may encounter.

Checking for network packet loss is the first area to look for problems. With NFS over UDP, a high retransmission count can indicate packet loss due to network or server problems. With NFS over TCP, the network layer on the client handles network packet loss, but server problems still show up as retransmissions. On some 2.4 kernels, TCP retransmissions are also a sign of large application writes on the client that have filled the RPC transport socket's output buffer.

To see retransmissions, you can use `nfsstat -c` at a shell prompt. At the top of the output, you will see the total number of RPCs the client has sent and the number of times the client had to retransmit an RPC. The retransmit rate is determined by dividing the number of retransmissions by the total number of RPCs. If the rate exceeds a few tenths of a percent, network losses may be a problem for your performance.

NFS over TCP does not show up network problems as clearly as UDP and performs better in the face of packet loss. *If your TCP mounts run faster than your UDP mounts, that's a sure sign that the network between your clients and your filer is dropping packets or is otherwise bandwidth-limited.* Normally UDP is as fast as or slightly faster than TCP. The client keeps network statistics that you can view with `netstat -s` at a shell prompt. Look for high error counts in the IP, UDP, and TCP sections of this command's output. The same command also works on a filer's console. Here look for nonzero counts in the "fragments dropped after timeout" and "fragments dropped (dup or out of space)" fields in the IP section.

There are a few basic sources of packet loss.

1. If the end-to-end connection between your clients and servers contains links of different speeds (for instance, the server is connected via Gigabit Ethernet, but the clients are all connected to the network with 100Base-TX), packet loss occurs at the point where the two speeds meet. If a gigabit-connected server sends a constant gigabit stream to a 100Mb client, only one packet in 10 can get to the client. UDP does not have any flow control built in to slow the server's transmission rate, but TCP does; thus, it provides reasonable performance through a link speed change.

2. Another source of packet loss is small packet buffers on switches. If either the client or server bursts a large number of packets, the switch may buffer them before sending them on. If the switch buffer overflows, the packets are lost. It is also possible that a switch can overrun a client's NIC in a similar fashion. This becomes a greater possibility for large UDP datagrams because switches and NICs tend to burst an entire IP packet (all of its fragments) at once.
3. The client's IP layer will drop UDP datagrams if the client's send or receive socket buffer runs out of space for any reason. The client's RPC client allocates a socket for each mount. By default, these sockets use 64KB input and output buffers, which is too small on systems that use large rsize or wsize or generate a large number of NFS operations in a short period. To increase the size of these buffers, follow the instructions in the appendix to this document. There is no harm in doing this on all your Linux clients unless they have less than 16MB of RAM.

If you have resolved these issues and still have poor performance, you can attempt end-to-end performance testing between one of your clients and a similar system on the server's LAN using a tool such as `ttcp` or `iPerf`. This exposes problems that occur in the network outside of the NFS protocol. When running tests such as `iPerf`, select UDP tests, as these directly expose network problems. If your network is full duplex, run `iPerf` tests in both directions concurrently to ensure your network is capable of handling a full load of traffic in both directions simultaneously.

If you find some client operations work normally while others cause the client to stop responding, try reducing your rsize and wsize to 1,024 to see if there are fragmentation problems preventing large operations from succeeding.

One more piece of network advice: Become familiar with network snooping tools such as `tcpdump`, `tcpslice`, and `ethereal`. On the filer, you can run `pktt`, which generates trace files in tcpdump format that you can analyze later on a client. These tools provide the last word in what is really happening on your network between your clients and filers. You may need to run both `tcpdump` on a client and `pktt` on your filer at the same time and compare the traces to determine where the problem lies.

You must explicitly specify several options to collect clean network traces with `tcpdump`. Be sure the snaplen option (-s) is set large enough to capture all the interesting bytes in each packet, but small enough that tcpdump is not overwhelmed with incoming traffic. If `tcpdump` is overwhelmed, it drops incoming packets, making the network trace incomplete. The default value is 96 bytes, which is too short to capture all the RPC and NFS headers in each packet. Usually a value of 256 bytes is a good compromise for UDP, but you can set it to zero if you need to see all the data in each packet. Snooping TCP packets requires a zero snaplen because TCP can place several RPC requests in a single network packet. If snaplen is short, the trace will miss RPCs that are contained near the end of long packets.

In addition, always use filtering to capture just the traffic between the client and the server. Again, this reduces the likelihood that `tcpdump` or your local file system will be overwhelmed by incoming traffic and makes later analysis easier. You can collect traffic to or from your client using the hostname filter. Several other `tcpdump` options allow you to collect traffic destined for one or more hosts at a time; read the manual to find out more.

An automounter can cause a lot of network chatter, so it is best to disable the automounter on your client and set up your mounts by hand before taking a network trace.

To find out if your application is competing for CPU resources with the NFS client, RPC client, or network layer on your client system, you can use the `top` program. If you see the `rpciod` process at the top of the listing, you know that NFS operations are dominating the CPU on your system. In addition, if you see system CPU percentage increase significantly when your application accesses NFS data, this also can indicate a CPU shortage. In many cases, adding more CPUs or faster CPUs helps. Switching to UDP may also be a choice for reducing CPU load if an uncongested high-speed network connects your clients and filer. As the Linux NFS client and networking layer improve over time, they will become more CPU-efficient and include features such as TCP offload that reduce the amount of CPU required to handle large amounts of data.

There are certain cases where processes accessing NFS file systems may hang. This is most often due to a network partition or server outage. Today's client implementation is robust enough to recover in most cases. Occasionally a client fails because of high load or some other problem. Unfortunately, little can be done in these cases other than rebooting the client and reporting the problem.

Error Messages in the Kernel Log

There are two messages that you may encounter frequently in the kernel log (this is located in `/var/log/messages` on Linux systems). The first is `server not responding`. This message occurs after the client retransmits several times without any response from a server. If you know the server is up, this can indicate the server is sluggish or there are network problems. If you know the server is down, this indicates the client is waiting for outstanding operations to complete on that server, and it is likely there are programs waiting for the server to respond.

The second, perhaps more frustrating, message is `can't get request slot`. This message indicates that the RPC client is queuing messages and cannot send them. This is usually due to network problems such as a bad cable, incorrectly set duplex or flow control options, or an overloaded switch. It may appear as if your client is stuck at this point, but you should always wait at least 15 minutes for network and RPC client timeouts to recover before trying harsher remedies such as rebooting your client or filer.

Oops

If a program encounters an unrecoverable situation in user space, it stops running and dumps core. If the same process encounters an unrecoverable situation within the Linux kernel, the kernel attempts to isolate the process, stop it, and report the event in the kernel log. This is called an "oops." Many times a failing process holds system resources that are not released during an oops. The kernel can run for a few more moments, but generally other processes deadlock or themselves oops when they attempt to allocate resources that were held by the original failing process.

If you are lucky, the kernel log contains useful information after the system recovers. Red Hat kernels automatically translate the kernel log output into symbolic information that means something to kernel developers. If the oops record contains only hexadecimal addresses, try using the ksymoops tool in the kernel source tree to decode the addresses.

Sometimes using a serial console setup can help capture output that a failing kernel cannot write into its log. Use a crossover cable (also sometimes called a file transfer or null modem cable) to connect your client to another system's serial console via its COM1 or COM2 port. On the receiving system, use a tool such as the `minicom` program to access the serial port. Serial console support is built into kernels distributed by Red Hat, but be sure to enable the serial console option in kernels you build yourself. Finally, you should set appropriate boot command line options (instructions provided in the

Documentation directory of the Linux kernel source tree) to finish enabling the serial console. Optionally, you can also update `/etc/inittab` to start a `mingetty` process on your serial console if you'd like to log in there.

Getting Help

Most Linux NFS client performance problems are due to lack of CPU or memory on the client, incorrect mount options, or packet losses on the network between the client and servers. If you have set up your client correctly and your network is clean, but you still suffer from performance or reliability problems, you should contact experts to help you proceed further.

Currently, there is no professionally maintained knowledge base that tracks Linux NFS client issues. However, expert help is available on the Web at nfs.sourceforge.net, where you can find a Linux NFS Frequently Asked Questions list, as well as several how-to documents. There is also a mailing list specifically for helping administrators get the best from Linux NFS clients and servers. Network Appliance customers can also search the NOW database for Linux-related issues. Network Appliance also maintains some of the more salient Linux issues within its BURT database. See the appendix in this report for more information.

If you find there are missing features or performance or reliability problems, we encourage you to participate in the community development process. Unlike proprietary operating systems, new features appear in Linux only when users implement them. Problems are fixed when users are diligent about reporting them and following up to see that they are really fixed. If you have ever complained about the Linux NFS client, here is your opportunity to do something about it.

When you have found a problem with the Linux NFS client, you can report it to your Linux distributor. Red Hat, for instance, supports an online bug database based on bugzilla. You can access Red Hat's bugzilla instance at *http://bugzilla.redhat.com/*.

When filing a BURT that relates to Linux client misbehavior with a filer, be sure you report:

- The Linux distribution *and* the Linux kernel release (e.g., Red Hat 7.2 with kernel 2.4.7-10).
- The client's kernel configuration (`/usr/src/linux/.config` is the usual location) if you built the kernel yourself.
- Any error messages that appear in the kernel log, such as oops output or reports of network or server problems.
- All mount options in effect (use `cat /proc/mounts` to display them, and do not assume they are the same as the options you specified on your `mount` commands).
- Details about the network topology between the client and the filer, such as how busy the network is, how many switches and routers, what link speeds, and so on. You can report network statistics on the client with `nfsstat –c` and `netstat –s`.
- Client hardware details, such as SMP or UP, which NIC, and how much memory. You can use the `lspci –v` command and `cat /proc/cpuinfo` on most distributions to collect most of this.
- Include a network trace and/or a dump of debugging messages (see the appendix).

Most importantly, you should carefully describe the symptoms on the client. A “client hang” is generally not specific enough. This could mean the whole client system has deadlocked or that an application on the client has stopped running. Always be as specific as you can.

In summary: If you cannot find what you need in this paper or from other resources, contact your Linux distributor or Network Appliance to ask for help.

Other Sundries

This section covers auxiliary services you may need to support advanced NFS features.

Telling Time

The clock on your Linux clients must remain synchronized with your filers to avoid problems such as authentication failures or incomplete software builds. Usually you set up a network time service such as NTP and configure your filers and clients to update their time using this service. After you have properly configured a network time service, you can find more information on enabling NTP on your filers in the *Data ONTAP System Administrator’s Guide*.

Linux distributions usually come with a prebuilt network time protocol daemon. If your distribution does not have an NTP daemon, you can build and install one yourself by downloading the latest ntpd package from the Internet (see the appendix).

There is little documentation available for the preinstalled NTP daemon on Linux. To enable NTP on your clients, be sure the ntpd startup script runs when your client boots (look in `/etc/rc.d` or `/etc/init.d`; the exact location varies, depending on your distribution; for Red Hat systems, you can use `chkconfig –level 35 ntpd on`). You must add the network time server’s IP address to `/etc/ntp/step-tickers` and `/etc/ntp.conf`.

If you find that the time protocol daemon is having some difficulty maintaining synchronization with your time servers, you may need to create a new drift file. Make sure your client’s `/etc/ntp` directory and its contents are permitted to the `ntp` user and group to allow the daemon to update the drift file, and disable authentication and restriction commands in `/etc/ntp.conf` until you are sure everything is working correctly.

As root, shut down the time daemon and delete the drift file (usually `/etc/ntp/drift`). Now restart the time daemon again. After about 90 minutes, it will write a new drift file into `/etc/ntp/drift`. Your client system should keep better time after that.

Note that until recent releases of Data ONTAP, filer time would often drift considerably, even when ntp was enabled. If you have used small values of timed.sched to minimize the time skew on your filers, we recommend you increase timed.sched after you have upgraded to Data ONTAP 6.4 or later. To do this, follow this procedure:

1. Wait until a period of light filer load.
2. Set these options on the filer’s console:

```
options timed.sched 1m
options timed.auto_adjust on
options timed.min_skew 0
options timed.window 5s
```

3. Watch the auto_adjust option for about an hour. The filer will automatically disable it when it has determined the correct clock skew for the filer hardware.
4. Set this option on the filer's console:

```
options timed.sched 1h
```

Security

Today, the Linux NFS client supports only two types of authentication: NULL and UNIX®. In future releases, Linux will support Kerberos 5, just as Solaris™ does today, via RPCSEC GSS. Later versions of the NFS protocol (e.g., NFSv4) support a wide variety of authentication and security models.

To maintain the overall security of your Linux clients, be sure to check for and install the latest security updates from your distributor. You can find specific information on Linux NFS client security in Chapter 6 of the NFS how-to located at nfs.sourceforge.net (see appendix).

When a host wants to send UDP packets that are larger than the network's maximum transfer unit, or MTU, it must fragment them. Linux divides large UDP packets into MTU-sized IP fragments and sends them to the receiving host in reverse order; that is, the fragment that contains the bytes with the highest offset are sent first. Because Linux sends IP fragments in reverse order, its NFS client may not interoperate with some firewalls. Certain modern firewalls examine the first fragment in a packet to determine whether to pass the rest of the fragments. If the fragments arrive in reverse order, the firewall discards the whole packet. A possible workaround is to use only NFS over TCP when crossing such firewalls. If you find some client operations work normally while others cause the client to stop responding, try reducing your rsize and wsize to 1,024 to see if there are fragmentation problems preventing large RPC requests from succeeding.

Firewall configuration can also block auxiliary ports that the NFS protocol requires to operate. For example, traffic may be able to pass on the main NFS port numbers, but if a firewall blocks the mount protocol or the lock manager or port manager ports, NFS cannot work. This applies to standalone router/firewall systems as well as local firewall applications such as tcpwrapper, ipchains, or iptables that might run on the client system itself. Be sure to check if there are any rules in `/etc/hosts.deny` that might prevent communications between your client and server.

Network Lock Manager

The NFS version 2 and 3 protocols use separate side-band protocols to manage file locking. On Linux 2.4 kernels, the `lockd` daemon manages file locks using the NLM (Network Lock Manager) protocol, and the `rpc.statd` program manages lock recovery using the NSM (Network Status Monitor) protocol to report server and client reboots. The `lockd` daemon runs in the kernel and is started automatically when the kernel starts up at boot time. The `rpc.statd` program is a user-level process that is started during system initialization from an init script. If `rpc.statd` is not able to contact servers when the client starts up, stale locks will remain on the servers that can interfere with the normal operation of applications.

The `rpcinfo` command on Linux can help determine whether these services have started and are available. If `rpc.statd` is not running, use the `chkconfig` program to check that its init script (which is usually `/etc/init.d/nfslock`) is enabled to run during system bootup. If the client host's network stack is not fully initialized when `rpc.statd` runs during system startup, `rpc.statd` may not send a reboot notification to all servers. Some reasons network stack initialization can be delayed are slow NIC devices, slow DHCP service, or CPU-intensive programs running during system startup. Network problems external to the client host may also cause these symptoms.

Because status monitoring requires bidirectional communication between server and client, some firewall configurations can prevent lock recovery from working. Firewalls may also significantly restrict communication between a client's lock manager and a server. Network traces captured on the client and server at the same time usually reveal a networking or firewall misconfiguration. Read the section on using Linux NFS with firewalls carefully if you suspect a firewall is preventing lock management from working.

Your client's nodename determines how a filer recognizes file lock owners. You can easily find out what your client's nodename is using the `uname -n` or hostname command. (A system's nodename is set on Red Hat systems during boot using the `HOSTNAME` value set in `/etc/sysconfig/network.`) The `rpc.statd` daemon determines which name to use by calling `gethostbyname(3)`, or you can specify it explicitly when starting `rpc.statd` using the "-n" option.

If the client's nodename is fully qualified (that is, it contains the hostname and the domain name spelled out), then `rpc.statd` must also use a fully qualified name. Likewise, if the nodename is unqualified, then `rpc.statd` must use an unqualified name. If the two values do not match, lock recovery will not work. Be sure the result of `gethostbyname(3)` matches the output of `uname -n` by adjusting your client's nodename in `/etc/hosts`, DNS, or your NIS databases.

Similarly, you should account for client hostname clashes in different subdomains by ensuring that you always use a fully qualified domain name when setting up a client's nodename during installation. With multihomed hosts and aliased hostnames, you can use `rpc.statd`'s "-n" option to set unique hostnames for each interface. The easiest approach is to use each client's fully qualified domain name as its nodename.

When working in high-availability database environments, test all worst-case scenarios (such as server crash, client crash, application crash, network partition, and so on) to ensure lock recovery is functioning correctly before you deploy your database in a production environment. Ideally, you should examine network traces and the kernel log before, during, and after the locking/disaster/locking recovery events.

The file system containing `/var/lib/nfs` must be persistent across client reboots. This directory is where the `rpc.statd` program stores information about servers that are holding locks for the local NFS client. A `tmpfs` file system, for instance, is not sufficient; the server will fail to be notified that it must release any POSIX locks it might think your client is holding if it fails to shut down cleanly. That can cause a deadlock the next time you try to access a file that was locked before the client restarted.

Locking files in NFS can affect the performance of your application. The NFS client assumes that if an application locks and unlocks a file, it wishes to share that file's data among cooperating applications running on multiple clients. When an application locks a file, the NFS client purges any data it has already cached for the file, forcing any read operation after the lock to go back to the server. When an application unlocks a file, the NFS client flushes any writes that may have occurred while the file was locked. In this way, the client greatly increases the probability that locking applications can see all previous changes to the file.

However, this increased data cache coherency comes at the cost of decreased performance. In some cases, all of the processes that share a file reside on the same client; thus aggressive cache purging and flushing unnecessarily hamper the performance of the application. Solaris allows administrators to disable the extra cache purging and flushing that occur when applications lock and unlock files with the “llock” mount option. Current versions of the Linux NFS client do not have such an option.

Using the Linux Automounter

Because Linux minor device numbers have only eight bits, a single client cannot mount more than 250 or so NFS file systems. The major number for NFS mounts is the same as for other file systems that do not associate a local disk with a mount point. These are known as *anonymous* file systems. Because the NFS client shares the minor number range with other anonymous file systems, the maximum number of mounted NFS file systems can be even less than 250.

The preferred mechanism to work around this problem is to use an automounter. This also helps performance problems that occur when mounting very large root-level directories. There are two Linux automounters available: AMD and automounter. The autofs file system is required by both and comes built into Red Hat releases after 7.1. If you build your own kernels and you require automounter support, be sure autofs support is enabled in your kernel configuration. More information is available on Linux automounters on the Web; see the appendix for the specific URL.

A known problem with the automounter in Linux is that it polls NFS servers on every port before actually completing each mount to be sure the mount request won’t hang. This can result in significant delays before an automounted file system becomes available. If your applications hang briefly when they transition into an automounted file system, make sure your network is clean and that the automounter is not using TCP to probe the filer’s portmapper. Ensure your infrastructure services, such as DNS, are responding quickly and with consistent results. Also consider upgrading your filer to the latest release of Data ONTAP.

An automounter can cause a lot of network chatter, so it is best to disable the automounter on your client and set up your mounts by hand before taking a network trace. Automounters depend on the availability of several network infrastructure services. If any of these services is not reliable or performs poorly, it can adversely affect the performance and availability of your NFS clients. When diagnosing an NFS client problem, triple-check your automounter configuration first. It is often wise to disable the automounter before drilling into client problem diagnosis.

Using an automounter is not recommended for production servers that may need immediate access to files after long periods of inactivity. Oracle, for example, may need immediate access to its archive files every so often, but an automounter may unmount the archive file system due to inactivity.

Fun with Net Booting

Intel systems can support network booting using DHCP, BOOTP, and TFTP. The Intel standard for supporting network booting is called a preexecution environment, or PXE. Usually this requires network interface hardware that contains a special PROM module that controls the network boot process. Network booting is especially helpful for managing clusters, blade servers, or a large number of workstations that are similarly configured.

Generally, Linux is loaded via a secondary loader such as grub, lilo, or syslinux. The secondary loader of choice for network booting is *pxelinux*, which comes in the *syslinux* distribution. See the appendix for information on how to obtain and install the *syslinux* distribution.

Data ONTAP releases 6.3.2 and 6.4 support *pxelinux*, allowing Linux to boot over a network. Earlier versions of Data ONTAP support booting filers, but do not support pxelinux because certain TFTP options were missing in the filer's TFTP server. To enable TFTP access to your filer, see the *Data ONTAP System Administrator's Guide.*

You must ensure that your client hardware supports network booting. Both the client's mainboard and network interface card must have support for network booting built in. Usually you can tell whether network booting is supported by reviewing the BIOS settings on your client or by consulting the mainboard manual for information on how to set up network booting on your client hardware. The specific settings vary from manufacturer to manufacturer.

You must configure a DHCP server on the same LAN as your clients. A DHCP server provides unique network configuration information for each host on a commonly administered network. For network booting, the DHCP server also instructs each client where to find that client's boot image. You can find instructions for configuring your DHCP server to support network booting included with the *pxelinux* distribution. The specific instructions for setting up a DHCP server vary from vendor to vendor.

If you intend to share a common root file system among multiple Linux clients, you must create the root file system on a filer using NFSv2. This is because of problems with how filers interpret major and minor device numbers and because of differences between how NFSv2 and NFSv3 Linux clients transmit these numbers. Linux clients, unless told otherwise, attempt to mount their root file systems with NFSv2. If the file system was created using NFSv3, the major and minor numbers will appear incorrect when mounted with NFSv2. Kernels use these numbers to determine the correct device driver for device special files. If the numbers are wrong, the Linux kernel will not be able to find its console or root file system, and thus it cannot boot.

When setting up multiple clients with NFS root file systems, common practice is to maintain a separate root file system for each client and mount each with "ro,nolock." Sharing a root file system among clients is not recommended.

Note that `/var/lib/nfs` must be persistent across reboots and unique for each client. A `tmpfs` file system per client, for instance, is not sufficient; the server will fail to be notified that it must release any POSIX locks it might think your client is holding if it fails to shut down cleanly. That can cause a deadlock the next time you try to access a file locked before the client restarted. If each client mounts a private `/var/lib/nfs` directory via NFS, it must be mounted using the nolock mount option, and before `rpc.statd` and `lockd` have started on the client.

For more information on Linux cluster computing, search the Internet for Beowulf or OpenMosix, or see the Linux high-availability site listed in the appendix of this document.

In summary: Make sure to check with your Linux distributor for any errata on a regular basis to maintain a secure system. Be sure your filers and clients agree on what time it is. If your client has trouble seeing the filer, look for packet filtering on your client or switches. Make each client's nodename is its fully qualified domain name.

Executive Summary

When setting up a Linux NFS client, you should try to get the latest kernel supported by your Linux distributor or hardware vendor. If you feel comfortable building kernels, you should build and install the latest stable kernel release available at *www.kernel.org*.

When using the latest Linux kernels, mount with NFS over TCP. You should use NFS version 3 where possible, and use the largest rsize and wsize that still provide good performance. Start with the hard and intr mount options.

Be sure the network between your clients and filer drops as few packets as possible.

Make sure to follow the instructions in the appendix for enlarging the transport socket buffers on all your UDP mounts.

If you have special needs, review this document carefully. Always look for the latest errata and bug fixes from your Linux distributor and watch for new Network Appliance technical reports.

Appendix

Related Material

Network Appliance NOW Web site: Type in *Linux* in the NOW PowerSearch text box.

SysKonnect's Gigabit Ethernet performance study
www.syskonnect.com/syskonnect/performance/gig-over-copper.htm

Red Hat 7.3 performance alert details
http://now.netapp.com/Knowledgebase/solutionarea.asp?id=ntapcs6648

O'Reilly's "Managing NFS & NIS, Second Edition" (ISBN 1-56592-510-6)
www.oreilly.com/catalog/nfs2

Linux tcpdump manual page
`man tcpdump`

Tcpdump home page
www.tcpdump.org

Linux ethereal manual page
`man ethereal`

Ethereal home page
www.ethereal.com

Data ONTAP packet tracer
`type "pktt list" at your filer's console`

Linux NFS manual page
`man nfs`

Linux NFS FAQ and how-to
http://nfs.sourceforge.net

Linux NFS mailing list
nfs@list.sourceforge.net

Linux network boot loader information
http://syslinux.zytor.com

Linux manual page with automounter information
`man autofs`

Linux automounter information

www.spack.org/index.cgi/AutomountHelp?action=show&redirect=LinuxAutomounter

Linux high-availability site

www.linux-ha.org

Network Time Protocol daemon home pages

www.cis.udel.edu/~ntp
www.ntp.org

Linux NFS client maintainer's Web site (with experimental patches)

www.fys.uio.no/~trondmy/src

Enlarging a Client's Transport Socket Buffers

Enlarging the transport socket buffers your client uses for NFS traffic helps reduce resource contention on the client, reduces performance variance, and improves maximum data and operation throughput. In Linux kernels after 2.4.20, the following procedure is not necessary, as the client will automatically choose an optimal socket buffer size.

```
1.  Become root on your client
2.  cd into /proc/sys/net/core
3.  echo 262143 > rmem_max
4.  echo 262143 > wmem_max
5.  echo 262143 > rmem_default
6.  echo 262143 > wmem_default
7.  Remount your NFS file systems on the client
```

This is especially useful for NFS over UDP and when using Gigabit Ethernet. You should consider adding this to a system startup script that runs before the system mounts NFS file systems. The size we recommend is the largest safe socket buffer size we've tested. On clients smaller than 16MB, you should leave the default socket buffer size setting to conserve memory.

Red Hat distributions after 7.2 contain a file called /etc/sysctl.conf where you can add changes such as this so they will be executed after every system reboot. Add these lines to your /etc/sysctl.conf file on your Red Hat system:

```
net.core.rmem_max = 262143
net.core.wmem_max = 262143
net.core.rmem_default = 262143
net.core.wmem_default = 262143
```

Other TCP Enhancements

The following settings can help reduce the amount of work your clients and filers do when running NFS over TCP:

```
1.  echo 0 > /proc/sys/net/ipv4/tcp_sack
2.  echo 0 > /proc/sys/net/ipv4/tcp_timestamps
```

These disable optional features of TCP to save a little processing and network bandwidth. If you build your own kernels, be sure that CONFIG_SYNCOOKIES is disabled. SYN cookies slow down TCP connections by adding extra processing on both ends of the socket. Some Linux distributors provide kernels with SYN cookies enabled.

Linux 2.2 and 2.4 kernels support large TCP windows (RFC 1323) by default. No modification is needed to enable large TCP windows.

Controlling File Read-Ahead in Linux

Read-ahead occurs when Linux predicts that an application may soon require file data it has not already requested. Such prediction is not always accurate, so tuning read-ahead behavior can have some benefit. Certain workloads benefit from more aggressive read-ahead, while other workloads perform better with little or no read-ahead. By default, Linux will attempt to read ahead by at least three pages, and up to 31 pages, when it detects sequential read requests from an application. Some file systems use their own private read-ahead values, but the NFS client uses the system defaults.

To control the amount of read-ahead performed by Linux, you can tune the system default read-ahead parameters using the sysctl command. To see what your system's current default read-ahead parameters are, you can try:

```
sysctl vm.min-readahead vm.max-readahead
```

The min-readahead parameter sets the least amount of read-ahead the client will attempt, and the max-readahead parameter sets the most read-ahead the client may attempt, in pages. Linux determines dynamically how many pages to read ahead based on the sequentiality of your application's read requests. Note that these settings affect all reads on all NFS file systems on your client system.

You can increase read-ahead if you know your workload is mostly sequential data or you are trying to improve WAN performance.

`Become rootsysctl –w vm.max-readahead=127sysctl –w vm.min-readahead=15` will set your system's read-ahead minimum and maximum to relatively high values, allowing Linux's read-ahead algorithm to read ahead as many as 127 pages. This value takes effect immediately. Usually the best setting for min-readahead is the number of pages in rsize, minus one. For example, if your client uses typically `rsize=32768` when mounting NFS servers, you should set min-readahead to 7. You can add this to the /etc/sysctl.conf file on your client if it supports this; see the section above for details.

On client systems that support a database workload, try setting the minimum and maximum read-ahead values to one or zero. This optimizes Linux's read-ahead algorithm for a random-access workload and prevents the read-ahead algorithm from polluting the client's data cache with unneeded data. As always, test your workload with these new settings before making changes to your production systems.

How to Enable Trace Messages

Sometimes it is useful to enable trace messages in the NFS or RPC client to see what it does when handling (or mishandling) an application workload. Normally you should use this only when asked by an expert for more information about a problem. You can do this by issuing the following commands:

```
1.  Become root on your client
2.  sysctl –w sunrpc.nfs_debug=3
3.  sysctl –w sunrpc.rpc_debug=3
```

Trace messages appear in your system log, usually `/var/log/messages`. To disable these trace messages, echo a zero into the same files. This can generate an enormous amount of system log traffic, so it can slow down the client and cause timing-sensitive problems to disappear or change in behavior. You should use this when you have a simple, narrow test case that reproduces the symptom you are trying to resolve. To disable debugging, simply echo a zero into the same files.

To help the syslogger keep up with the log traffic, you can disable synchronous logging by editing `/etc/syslog.conf` and appending a hyphen in front of `/var/log/messages`. Restart the syslog daemon to pick up the updated configuration.

How to Enable Uncached I/O on RHEL AS 2.1

Red Hat Enterprise Linux Advanced Server 2.1 Update 3 introduces a new feature that is designed to assist database workloads by disabling data caching in the operating system. This new feature is called “uncached NFS I/O” and is similar to the NFS O_DIRECT feature found in Enterprise Linux 3.0 and SuSE’s SLES 8 Service Pack 3. When this feature is enabled, an application’s read and write system calls are translated directly into NFS read and write operations. The Linux kernel never caches the results of any read or write, so applications always get exactly what’s on the server.

Uncached I/O affects an entire mount point at once, unlike NFS O_DIRECT, which affects only a single file at a time. System administrators can combine mount points that are uncached (say, for shared data files) and mount points that cache data normally (say, for program executables or home directories) on the same client. Also unlike NFS O_DIRECT, uncached I/O is compatible with normal I/O. There are no alignment restrictions, so any application can use uncached I/O without modification.

When uncached I/O is in effect, it changes the semantics of the “noac” mount option. Normally the “noac” mount option means that *attribute caching* is disabled. When the uncached I/O feature is in effect, “noac” is changed to mean that *data caching* is disabled. When uncached I/O is not in effect, “noac” mount points behave as before. Uncached I/O is turned off by default.

To enable uncached I/O, follow this procedure:

```
  Become root
  Start your favorite editor on /etc/modules.conf
  Add this line anywhere: “options nfs nfs_uncached_io=1”
```

Uncached I/O will take effect after you reboot your client. Only mount points that use the “noac” mount option will be effected by this change.

For more information on how to use this feature with Oracle9*i* RAC, see NetApp TR 3189.

NetApp®

Network Appliance, Inc.
495 East Java Drive
Sunnyvale, CA 94089
www.netapp.com