Technical Report

Using Red Hat Client with NetApp Storage over NFS

Bikash Roy Choudhury, NetApp
October 2013 | TR-3183

Abstract

This report helps you to get the best from your Linux® NFS clients when used in an environment that includes NetApp® storage. Most of the discussion revolves around Red Hat Enterprise (RHEL) clients. This document does not include best practices and recommendations for block protocols like FCP and iSCSI. The paper highlight some of the best practices and tuning parameters that would be required for running user home directories and regular UNIX® file systems. The paper describes various tuning options on RHEL clients and diagnoses performance and reliability problems. Every workload has a different access pattern and characterization; we recommend appropriate tuning on RHEL clients based on the workload characterization, which should not be treated as “one size fits all" for all workloads. Separate technical reports have been published for different applications that provide recommendations for specific workloads. In this paper you will learn where to look for more information when faced with problems that are difficult to diagnose.

Storage tuning information specific to your application may be available in other NetApp technical reports and solution guides.

TABLE OF CONTENTS

1 Introduction

Traditionally NetApp has not recommended a specific Linux client because there are more than a hundred different Linux distributions and thousands of different Linux kernel releases. When you add to that the fact that there are many different versions of user-level helper applications (such as the `mount` command), you can understand why NetApp is reluctant to choose a single, or even two or three, specific clients.

Additionally, 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 a hardware vendor recommends another and an application vendor certifies yet a third.

However, more and more NetApp customers recognize the value of Red Hat Enterprise Linux (RHEL) in their environments, including the value of the RHEL NFS client. The NFS client in recent RHEL versions has been improved considerably with respect to stability, performance, scalability, and the ability to work under degraded network conditions when running enterprise workloads.

In addition, because Red Hat is the most commonly used commercial Linux distribution, including the NetApp customer base, in this report we touch on some of the key improvements that Red Hat included in its newer kernels to support applications that use NFSv3, NFSv4.x, and pNFS. This document highlights the changes and new functionalities available on the newer Red Hat kernels with respect to the Network File System (NFS) that can integrate with NetApp storage for optimum performance. NetApp also certifies the major Red Hat releases for specific NFS versions and updates that are in the interoperability matrix: http://support.netapp.com/matrix.

Although this document covers best practices for Red Hat Enterprise Linux and NetApp NFS, tuning information for specific applications is not within the scope of the document.

This document focuses mainly on Red Hat Enterprise Linux (RHEL) versions of the kernel and on identifying some of the key areas that may be considered while planning a new Red Hat client deployment or while administering an existing environment that contains NFS clients accessing NetApp storage. These areas include:

- The level of performance and stability to expect from RHEL NFS clients
- How to tune RHEL NFS clients to perform well with appliances
- How to diagnose client and network problems that involve RHEL NFS clients
- How to configure other services required to provide advanced NFS features
- Where to find more tuning and diagnostic information on support.netapp.com (formerly known as NOW™, NetApp on the Web) and the Internet

Finally, some applications are more sensitive to NFS client behavior than others. Recommending a particular RHEL kernel version depends on the applications that you want to run and your performance and reliability requirements. Therefore, instead of recommending one or two releases that work well, this paper provides some guidelines to help you decide on a release based on the different features available among the newer RHEL releases. We also provide some high-overview guidelines to make your Linux NFS clients work their best.

2 Which RHEL 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 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 a Linux distribution, but for Linux NFS client functionality we focus on the RHEL distribution kernel. The kernel contains the set of base operating system files that are included when customers install Red Hat Enterprise Linux on their hardware. A RHEL distribution comes with an initial kernel version, but, like all software in the distribution, the kernel can be updated to gain additional functionality, address bugs, or address security issues.

2.1 Identifying Kernel Releases

The version number of a Linux distribution and the release number of a Linux kernel use different naming schemes. While planning a distribution, each distributor chooses a particular kernel release (for example, 2.6.32) and adds some modifications of its own before placing the kernel into a distribution. Current Red Hat distributions identify their kernel modifications by adding a number to the base kernel release; for example, RHEL 6.3 ships a 2.6.32-279 kernel. To reduce the amount of variation encountered in support contracts, distributors support a small set of kernels.

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, NetApp does not recommend unsupported specific patches or kernel versions.

There is a single upstream kernel branch in which all new development is integrated and that is periodically checkpointed into kernel versions. Each checkpointed version (2.6.39, 3.0.0–3.8.0) evolves into a stable branch containing back-ported fixes from upstream kernels deemed necessary for stability. The stable branch is intended to remain "stable" at all times. Some new features (submodules) can be marked as experimental (or in Tech Preview) when they are not ready for regular enterprise consumption.

As of this writing, the latest development branch version is 3.8.0. Linux kernels are not published on a time-based schedule. Kernel revisions 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 2 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 instead of working them into development releases. To expedite the addition of new features, kernel developers recently changed the way stable and development branches are treated.

2.2 Today's Linux distributions

As mentioned above, distributions are numbered differently than kernels. Each distributor chooses its own numbering scheme. When describing your Linux environment to anyone, be sure to list both the distribution release number and the kernel release number. Distributors usually append another number on the end of their kernel versions to indicate which revision of that kernel is in use. For instance, Red Hat initially shipped kernel 2.6.32-272 with its RHEL 6.3 distribution, and then released the 2.6.32-279 update kernel several months later.

An important trend in Linux distributions is the existence of commercial, or "enterprise," Linux distributions. Enterprise distributions are quality-assured releases that come with support. NetApp recommends that its Linux customers always use the latest actively maintained distributions available, along with the latest errata patches.

Customers running older and/or unsupported distributions do not get the benefits of rapid security fixes and bug fixes on their Linux systems. Most Linux distributors will not address bugs in older distributions at all.

To find out which kernel your clients run, you can use this command:

```
# uname -r
2.6.32-358.el6.x86_64
#
```

Kernels built from community source code usually have only three or four dot-separated numbers, like 2.6.32.1. Distributors generally add a hyphen and more version numbers (in this case, -358), which indicate that additional patches over the community source base have been applied. The keyword on the end, such as "hugemem" or "smp" or "x86_64," shows additional hardware capabilities for which this kernel was built.

2.3 The NFS Client in the 2.6 Kernel

During 2004, distributions based on the 2.6 kernel appeared and became stable enough to be deployed in production environments.

A new feature in the 2.6 kernel is support for the latest version of the NFS protocol, version 4.0 / 4.1 (hereafter called NFS version 4). Developers are still in the process of retrofitting the Linux NFS client and server implementations for the new protocol version. Certain features like read and write delegations are available today in the 2.6.32 (RHEL6) kernel, but others, such as replication and migration support, are still under development and are not yet available. Support for NFS version 4 is available now in Fedora Linux and RHEL 6, and is regularly tested with the NFS version 4 implementation in the Data ONTAP® operating system as support for new features such as delegation. Refer to the interoperability matrix table published by NetApp (http://support.netapp.com/matrix) to find out the recommended RHEL version that supports NFSv4 and integrates with Data ONTAP.

Although full file system replication and migration support still is not available in the mainline kernel, there is support for NFSv4 "referrals." Referrals use the replication/migration protocol used in order to tell a client that a given subdirectory exists on another server (rather like an automounter). Clustered Data ONTAP 8.1 and later support referrals in which the referral feature is only effective during a mount operation. When a volume is mounted from a node where the volume is remote, the mount is automatically redirected to the node where the volume is local. When you move a volume nondisruptively in the cluster namespace on NetApp storage, the RHEL clients can still access the now-remote volume in the new location on the NetApp storage from the mount point. This means that moving a volume in the background in the cluster namespace does not cause disruptions to the applications running on the RHEL client. To regain direct access to the now-remote volume, an "unmount" and a "mount" have to be performed on the client for it to identify the new location of the volume and access the file(s) in it using a direct data path.

The 2.6 kernel also brings support for advanced security frameworks such as Kerberos 5. Support for Kerberos works with NFS version 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 enable the integrity or privacy of communication between an NFS client and a server. As with NFS version 4, developers are still refining support for NFS with Kerberos in Linux, so there are still some missing features. In RHEL5 (2.6.18 kernel), it is safe to use for Kerberos authentication, integrity, and privacy. The newer versions of RHEL5 and RHEL6 support DES, DES3, and AES128-/256-bit encryptions. Because the Linux implementation of NFS with Kerberos is based on Internet standards and because it is tested regularly with the leading proprietary Kerberos implementations, Linux NFS with Kerberos interoperates seamlessly with NetApp storage.

The NFS client in the 2.6 kernel has demonstrated superior performance and stability compared to older Linux NFS clients; however, as usual customers should be cautious about moving their production workloads onto releases of the Linux kernel. The newer releases of RHEL5 are based on Fedora Core and the kernel version is 2.6.18-308.8.1.el5. The kernel for RHEL6.4 GA is 2.6.32-358.

2.4 Highlights of RHEL6.x kernel

- Because the bandwidth requirements are increasing, it is very difficult for a single CPU to handle all the data received over the wire. Starting with RHEL6.x, networking is multithreaded. That means multicores on the node help to receive data from the wire a lot faster. Receive Flow Steering (RFS) and Receive Packet Steering (RPS) work hand in hand to forward the packets received from the network layer to the appropriate CPU where the corresponding application thread is processed. In other words, instead of picking a random CPU to receive the data from the wire, the receive buffers are processed by the same CPU that sent the data request. This also helps to alleviate the bottleneck of having to receive all the traffic for that network interface queue on one particular CPU. The 10Gbe drivers are also improved to provide much better performance.
- The Linux kernel RPC implementation uses an RPC slot construct to manage interaction with the transport layer. The RPC is the layer just below the application layer in the ISO-OSI model. Each RPC slot represents an NFS request ready to be sent on the wire, and the number of RPC slots represents the number of possible in-flight NFS requests. If an NFS request can't get an RPC slot because all slots are in use, then the request waits for an available slot in the RPC layer. This in

turn means that the transport layer (TCP) receives no pressure from the RPC layer. The TCP layer is responsible for handling the advertised window size on the network to the storage. The number of RPC slots needed to fill the TCP window is the TCP window size divided by the wsize for WRITEs or the rsize for READS. For some network configurations and workloads, the TCP window size can be larger than the number of RPC slots times the wsize or rsize. In this case the RPC slot layer throttles the TCP layer as NFS requests wait for an available RPC slot, so the TCP window may not be filled and performance can suffer under certain network configurations and workloads.

- In the static RPC slot implementation used by Linux kernels 3.1 and earlier, the number of slots is set by the administrator with a default of 16 slots and a maximum of 128 slots using the `sunrpc.tcp_slot_table_entries` sysctl option. If the TCP window is being throttled by the number of RPC slots as described above, the administrator needs to recognize this situation and increase the number of RPC slots. On some long-haul WAN configurations the maximum of 128 RPC slots is not enough to fill the TCP window. For example, in the figure below, if an application sends 2MB WRITE requests with the rsize and wsize mount option values set to 65536 (65k), then each 2MB write request is broken up into 32 (64k) RPC requests in which each 64k takes one slot in the RPC layer. If these requests are being sent over a long-haul WAN configuration or over a 10G network, then increasing the number of RPC slots from 16 to 128 may not be enough data to fill the TCP window.

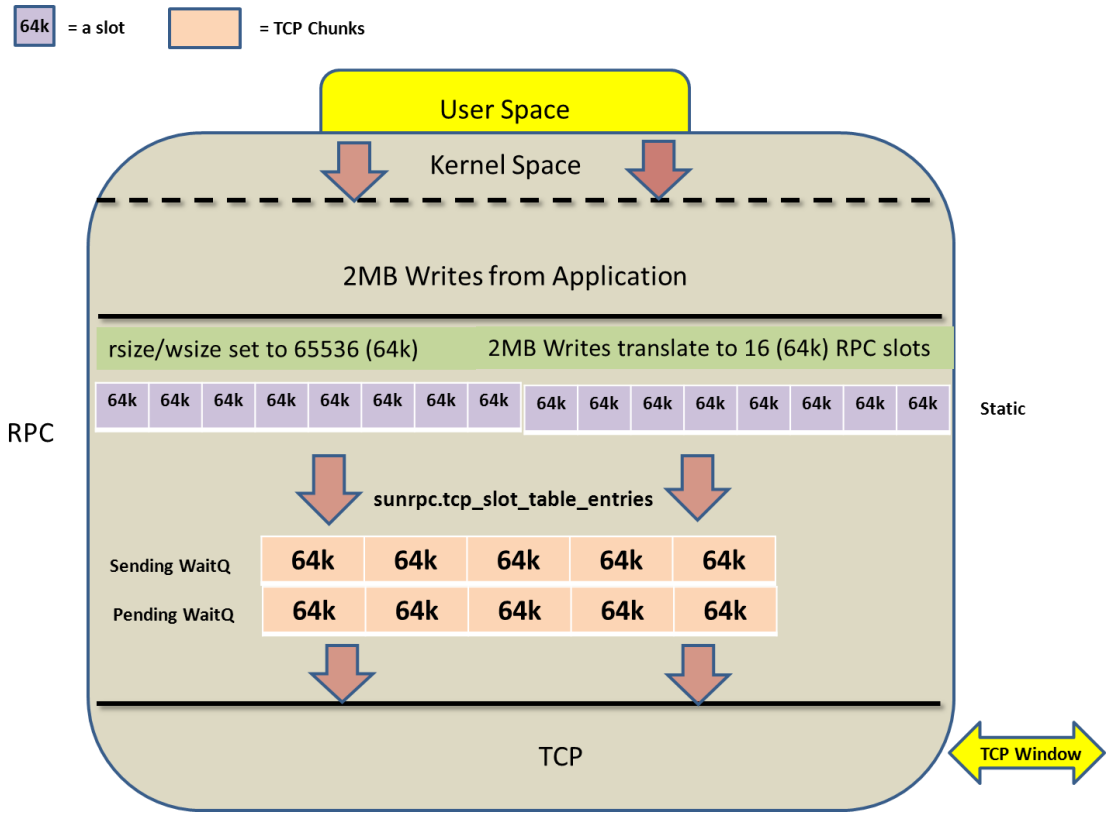

FIGURE 1) STATIC RPC SLOT LAYOUT WITH NFSV3/NFSV4.

- The dynamic slot implementation present in the newer releases of Red Hat, RHEL 6.3 and later, and in upstream kernels 3.2 and later removes this performance barrier for non-sessions-based NFS (v3 and v4.0). Each RPC client starts with `sunrpc.tcp_slot_table_entries` number of RPC slots, which by default is set to 2. RPC slots are then dynamically allocated as needed up to `sunrpc.tcp_max_slot_table_entries` and then freed down to sunrpc.tcp_slot_table_entries if not in use. The default maximum number of RPC slots is increased to 65536 slots. With this

implementation, the RPC layer never waits for RPC slots unless the maximum number of RPC slots has been hit or the allocation fails due to memory pressure. The TCP layer is not throttled. The administrator does not need to set any RPC slot values. This change is expected to improve the performance for clients mounted over NFSv3 and NFSv4.

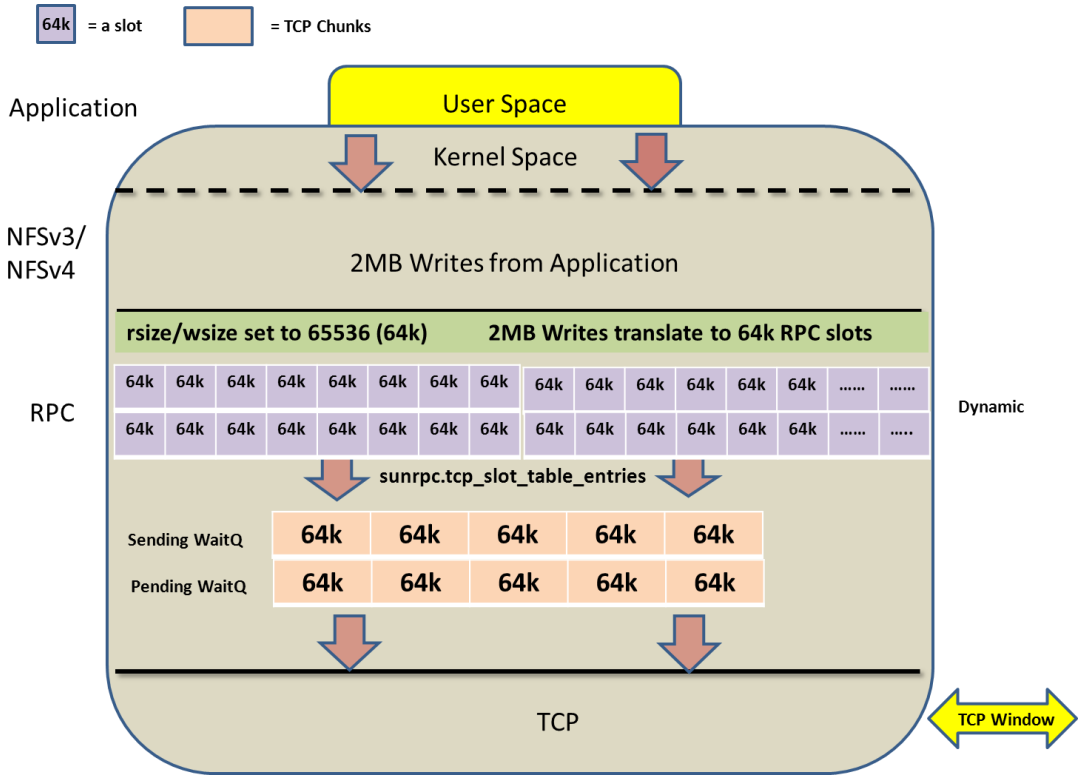

FIGURE 2) DYNAMIC RPC SLOT LAYOUT WITH NFSV3/NFSV4.

- The NFSv4.1 protocol introduces Sessions (RFC 5661 reference), which exist for every NFSv4.1 connection. The session is part of the NFS layer. Sessions also uses a slot construct that is similar to the RPC slot. Each session slot represents an NFSv4.1 request ready to be sent on the wire, and the number of session slots represents the number of possible in-flight NFS requests. If an NFS request can¹t get a session slot because all slots are in use, then the request waits for an available session slot in the NFS layer. Once a request has a session slot, it then must obtain an RPC slot and all of the issues with RPC slots explained above apply.

  Unlike the RPC slots construct, which is internal to the Linux RPC implementation, session slot behavior is dictated by the NFSv4.1 protocol. The number of session slots on an NFS v4.1 connection is negotiated with the server or the NetApp storage when the session is established. Today NetApp Data ONTAP 8.1 has 180 session slots and in future releases this will be a higher value, around 1,000, to dynamically grow the number of slots. The NetApp storage controls the number of slots it is going to provide to multiple clients requesting sessions. Although the NFSv4.1 protocol provides dynamic session slot implementations for all versions of RHEL 6, the number of slots that are negotiated between the NetApp storage and the Linux client during the time of establishing the session does not change as long as the session is active and open.

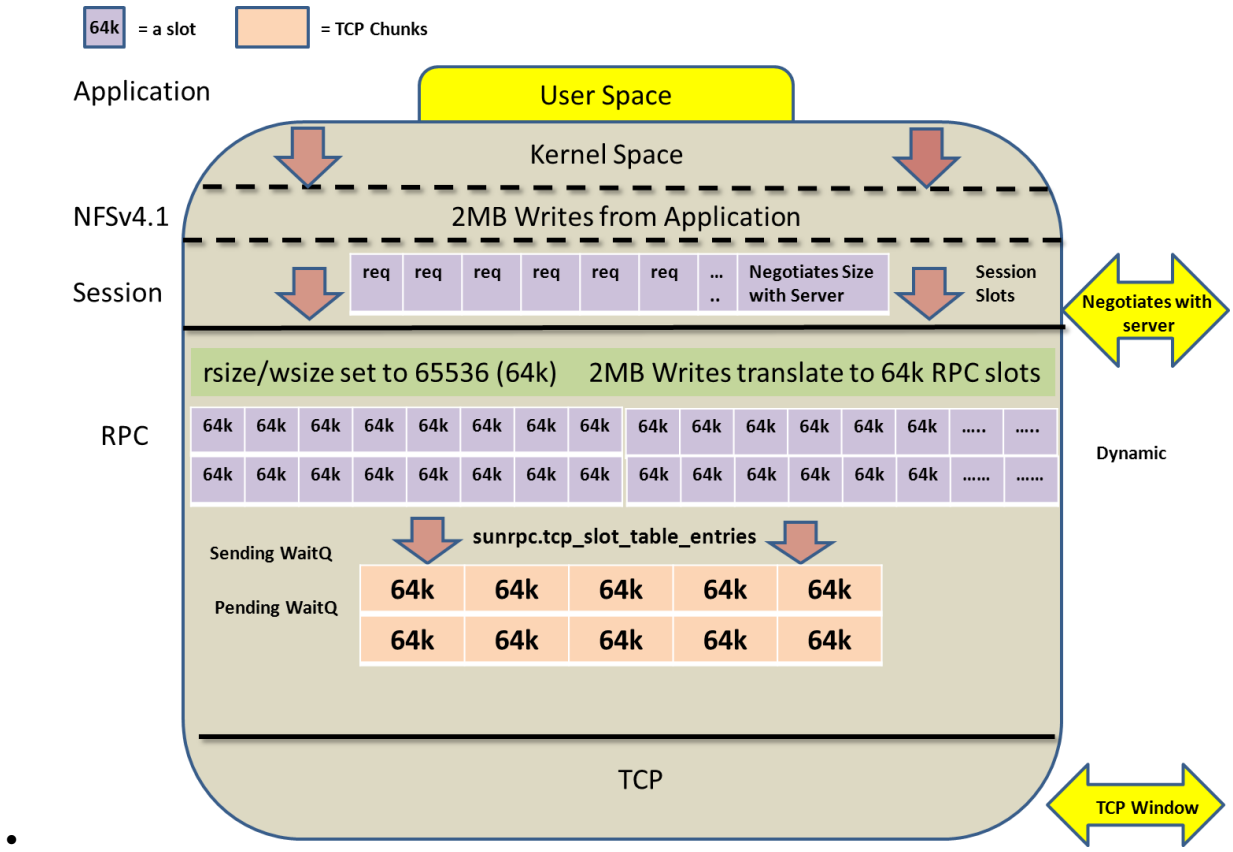

FIGURE 3) DYNAMIC RPC SLOTS LAYOUT WITH NFSV4.1.

- DirectIO or the O_DIRECT is an OPEN() system call argument in the Linux kernel and is normally used by applications that do not want to use page cache or memory buffers on the hosts running Red Hat Enterprise Linux (RHEL). A good example is Oracle® Real Application Cluster (RAC) nodes that would like to write the new or modified blocks from the Shared Global Area (SGA) of the database to NetApp storage directly, bypassing the page cache or the memory buffers of the host. This reduces additional buffering on the RAC nodes and improves performance by providing a zero copy mechanism from the application buffer to the disks in the storage subsystem. The application determines if it would use DIRECTIO to communicate synchronously or asynchronously using the POSIX Asynchronous IO (AIO) interface inside the kernel to improve performance.

  Prior to RHEL6.4, the regular READ/WRITE path shared the same code while the DIRECTIO used a different code path. Due to the disparate code path between the normal READ/WRITE and DIRECTIO, the fixes or patches applied to the normal path did not apply to the DIRECTIO path and vice versa. Similarly the DIRECTIO code path was not available for pNFS. It only benefited applications that used NFSv3 and NFSv4 protocols.

  The READ vector (readv) and WRITE vector (writev) are more efficient ways of reading into and writing from multiple memory buffers compared to the normal read() and write() operations. One of the most important characteristic of the readv/writev is to coalesce the vector buffers that are not contiguous in an array and convert them into one buffer and perform a single read or write operation. This eliminates the overhead involved with the trivial way of performing multiple writes into different buffers; copy them into one block of memory using "memcpy," following with a single write operation. Prior to RHEL6.4 the process of coalescing vector buffers was unavailable for applications that used DIRECTIO with NFSv3 or NFSv4. This posed issues for hypervisors like Red Hat KVM that use DIRECTIO and coalescing of vector buffers to improve performance. The normal read and write operations that use the regular page cache always had the ability to coalesce the vector buffers into large reads and writes.

In RHEL6.4 the READ/WRITE code path was rewritten and was merged with DIRECTIO. Now the normal READ/WRITE operations that use page cache and the DIRECTIO path share the same code. Therefore, fixes applied to either one will have a global impact. The change in the READ/WRITE code helped to apply DIRECTIO feature for applications that use pNFS. By reusing the page cache code, the support for DIRECTIO was achieved for pNFS without having to change the pNFS layout drivers. Now RHEL6.4 has the capability to use DIRECTIO on all of its data paths as required by the applications using pNFS. The readv/writev buffers are also effectively coalesced in large reads and writes, thereby improving the performance of the applications that use DIRECTIO.

- The NFS cache consistency behavior is also optimized in RHEL6.4. In NFSv3 "mtime" is always checked for validating the cache contents. NFSv4.x would use "ctime," "mtime," and "change attributes" to check the validity of the cache. This seems to be redundant and extra effort has to be taken to keep all these variables up to date on the share storage infrastructure. With the new fix in RHEL6.4, the "change attribute" on a file is the only one that is kept up to date on the share storage infrastructure. However, for operations like REMOVE, CREATE, LINK, and OPEN, the change attribute is updated and thereby invalidates the cache. The other variables like "mtime," ctime," and "atime" get updated eventually.

- The Linux kernel has an inherent behavior of tagging GETATTRs to a write operation to enable consistency for a file system accessed over NFSv4 and NFSv4.1. This is known as a postattribute operation. When NetApp clustered Data ONTAP receives an NFSv4.x write request, it generates two SPIN-NP operations: one for the WRITE and the other for the GETATTR. This proves to be a performance deterrent for applications that run over NFSv4.x. This also impacts the ability to do "zero-copy" operations in clustered Data ONTAP for NFSv4.x WRITE requests. This issue is now fixed in clustered Data ONTAP and patches are rolled into RHEL6.4. Now a single NFSv4.x WRITE to clustered Data ONTAP will perform a single SPIN-NP operation and a prefetch of a GETATTR. Instead of generating two SPIN-NP operations we have just one operation to satisfy on clustered Data ONTAP for a WRITE request.

  The fix also allows clustered Data ONTAP to perform "zero-copy" operations for file systems accessed over NFSv4.x in which it steals the memory buffers from the network and adds them to the buffer cache. The postattribute patches in RHEL6.4 allow applications sending WRITE operations to clustered Data ONTAP over NFSv4.x using the DIRECTIO path to drop all the GETATTRs. The GETATTRs that are tagged to a WRITE operation are also dropped when delegations are enabled.

  These fixes are mainly targeted when you are using either DIRECTIO or delegations. If the setup does not have any of these conditions, then the client would do a WRITE and check for the change attributes. A regular WRITE operation that does not use DIRECTIO or delegations will tag the WRITEs with a GETATTR. Technically there would be two operations. The reason why the fix only applies to DIRECTIO and delegation scenarios is that DIRECTIO bypasses the page cache and the cache consistency is not required; in the case of delegations the cache consistency already exists.

  However, these fixes in clustered Data ONTAP and the patches in RHEL6.4 have nothing to do with NFSv3 because the postattributes are part of the NFSv3 WRITE operation. NFSv3 performs "zero-copy" operations in clustered Data ONTAP and does not send any postattributes like GETATTR for a write operation. pNFS, which is component of NFSv4.1, also does not send any postattributes for regular operations or over DIRECTIO for WRITE operations.

- Where NFSv3 places the numeric values for UID and GID in a GETATTR or SETATTR call, the NFSv4.x protocol replaces the numeric syntax with a string format such as name@xyx.com in which the name portion is the user or group name and the xyz.com portion is a DNS name. Although this allows servers to more easily export any local file system representation of cty, it imposes a requirement for translation between the on-the-wire string representation of a user or group into the local file system representation. On Linux the idmapd daemon performs this task. However, not all customers who plan to move to an NFSv4.x setup wish to set up a translation

service that can include LDAP configuration. Therefore, to fall back to the old method of identification that used UID and GID, the "numeric_id" feature was implemented. When using this implementation, an external translation by *idmapd* is not required because UID and GID information is exchanged in the NFSv4.x protocol by the kernel, automatically translating a UID or GID into the string syntax. Therefore UID 100 becomes the string 100@xyx.com. The numeric_id feature is not permitted for AUTH_GSS security flavors and is limited to the AUTH_SYS security flavor. This means that that this feature is only useful when you export a file system over AUTH_SYS and not in environments in which the users are configured in Kerberos as per RFC 3530bis. In a Kerberos environment a DNS and an LDAP are still required to identify and resolve the Kerberos Principal name. The numeric_id feature has to be supported by the Linux client and also by the server. The RHEL6.4 kernel and clustered Data ONTAP 8.1 and later support the numeric_id feature for file systems mounted over NFSv4.x.

- Under a high I/O workload, certain Linux clients fail to perform a state recovery on multiple LIF migrate operations when the file system is mounted over NFSv4.1 and file delegations are enabled on clustered Data ONTAP. The Linux client hangs in this scenario when the file system is mounted over NFSv4.1 because it appears to the client that the server is rebooting multiple times when multiple overlapping LIF migrate operations are performed. When an attempt to unmount the NFSv4.1/pNFS file system is made on the client mounting from a clustered Data ONTAP node that is a pNFS metadata server as well as the data server, the operation does not allow the unmount process to complete cleanly. The RHEL6.4 kernel has the fixes for this issue.

- Clustered Data ONTAP 8.2 introduces the Quality of Service (QoS) feature. When QoS is enabled on clustered Data ONTAP for a file system mounted over NFSv4.1, the RHEL6.4 NFS client is optimized to handle the IOPs limit set by the QoS feature on the volume more efficiently.

- In RHEL6.x, the `intr` mount option is deprecated. The kernel is hard coded to `nointr` by default. That mean you need to interrupt the process by using the `kill -9` command. The applications that use `nointr` to interrupt the Linux NFS client when it gets stuck waiting for server or network recovery will no longer be recommended to have that as an explicit mount option.

Note: RHEL6.4 GA may not have all the feature fixes. Use the "yum update" after RHEL6.4 is installed to download the bz stream that has all the fixes.

Best Practice

Use the latest distribution and kernel available from your distributor when installing a new deployment and attempt to keep your existing Linux clients running the latest updates from your distributor. Always check with your hardware and application vendors to be certain that they support the distribution and kernel you choose to run. Contact NetApp if you have special requirements for integration with NetApp storage.

3 Foolproof Mount Options for Linux NFS Clients

If you 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.

Also look in /etc/fstab on the client to see which options the client attempts to set when mounting a particular file system. Check your automounter configuration to see which defaults it uses when mounting. Running the `mount` command at a shell prompt tells you which 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 which mount options are in effect for an existing mount.

The NFS protocol version (3 or 4.x) used when mounting an NFS server can change depending on which protocols the server exports, which version of the Linux kernel is running on the client, and which version of the mount utilities package you use. NFSv3 is the minimum NFS version recommended to mount any file system. So that your client uses the NFSv3 protocol, specify "vers=3" when mounting a system. Be sure that the NFSv3 protocol is enabled on NetApp storage before trying to mount using "vers=3" on the Linux client. RHEL6.x mounts NFSv4 by default.

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 as read-only. If a “soft” timeout interrupts a read operation, the client’s cached copy of the file is probably corrupt. Purging a corrupt 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.

Best Practice

NetApp recommends using hard mounts.

When running applications such as databases that depend on end-to-end data integrity, use “hard” as the mount option. Database vendors like Oracle have 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). However, with `intr` being deprecated in RHEL6.x, `nointr` is now the default value and does not require any explicit way to mention in the mount options.

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. This reduces the likelihood that very brief outages or a few dropped packets will cause an application failure or data corruption.

In 2.4 and older kernels, there was a hard limit of 255 mounts per file system type because the minor number was only 8 bits. The 2.6.18 kernels (on which RHEL5.x is based) and later have a 20-bit minor number, so this removes any practical limit.

There is a limit on the number of reserved ports available. Communication with the NetApp storage NFS service and mount service occurs over privileged ports. By default there are 358 available reserved ports for use (min: 665, max: 1,023). The lower limit can be updated through sysctl (`sunrpc.min_resvport`), but do it with care because it will reduce the number of reserved ports for other kernel services.

The reserved ports limit the number of "active" mounts that are being accessed at the same time. There is a limitation during a mount storm or when all of the mounts are continuously active. If all of the mounts are listed in /etc/fstab, then upon reboot the client will attempt to mount all of them at the same time. This means that you'll be limited to ~179 (358/2) mounts. This is because each mount will use one port to communicate with mountd and another port to communicate with NFS on the storage. The mountd port connection is closed after a couple of minutes and the NFS connection remains active as long as there's activity on the mount point. If there is no activity then the connection is closed and may be used by another mount point.

If the environment has a large number of mounts it is better to use an automounter to avoid mount storms. An automounter will allow the client to have more mounts available but not have all of them triggered at the same time.

If the mountd port number is fixed on the server, then adding 'mountport=<portnr>' to the mount options should also help reduce the total number of requests. The mountd daemon for NFSv3 normally listens on port 4046 on NetApp storage. On the RHEL client the file system can be mounted with “-o mountport” specifying the exact port number 4046 - -o mountport=4046.” In order to avoid any overhead from portmap requests, port=2049 can be used as a mount option. This option can also be included in the /etc/sysconfig/nfs file.

Finally, you may also specify 'mountproto=udp' in order to avoid any extra TCP connections for mountd requests.

3.1 Choosing a Network Transport Protocol

NetApp recommends using TCP as the transport mechanism for all applications and workloads.

NFS over TCP can handle multispeed networks (networks in which the links connecting the server and the client use different speeds), higher levels of packet loss and congestion, fair bandwidth sharing, and widely varying network and server latency, but this can cause long delays during server recovery. 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. With the new NFSv4 and NFSv4.1 protocols, UDP is forbidden; currently TCP is the only transport protocol available.

You can control RPC retransmission timeouts with the `timeo` option. Retransmission is the mechanism by which clients enable a server to receive and process 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.

Retransmission for NFS over TCP works somewhat differently. The TCP network protocol contains its own timeout and retransmission mechanism that enables packets to arrive at the receiving end reliably and in order. 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.

In summary, we strongly recommend using TCP as the transport of choice for NFS on modern Linux distributions. To avoid IP fragmentation issues on both the client and NetApp storage, consider explicitly specifying "tcp" on all your NFS mounts. In addition, we strongly recommend the explicit use of the `timeo=600` mount option on Linux mount options to shorten the retransmit timeout.

For NFSv4, NetApp recommends using `timeo=6000` against clustered Data ONTAP 8.2 in order to work around replay cache issues.

Best Practice

NetApp recommends using `timeo=600` for NFSv3 and `timeo=6000` for NFSv4.

3.2 Capping the Size of Read and Write Operations

NFS clients break application read and write requests into smaller chunks when communicating with NFS servers. The maximum size, in bytes, that a client uses for NFS read requests is called the rsize, and the maximum size a client uses for NFS write requests is called the wsize. Together, these two are often referred to as the transfer size, because there are few cases in which the two need to have different values. In RHEL6.x, the maximum transfer size is 1MB, but NetApp recommends using 64k rsize and wsize values in the mount options for optimum performance.

The network transport protocol (TCP) interacts in complicated ways with the transfer size. When you encounter poor performance because of network problems, using NFS over TCP is a better way to achieve good performance. Ideally the MTU size is always set to 1,500 by default. With jumbo frames the MTU size can be set to 9,000. However, caution should be exercised while setting jumbo frames. The 9,000-MTU value has to be set end to end from the client network interfaces, switch ports, and on the NetApp storage to provide better performance. The number of frames that move on the wire depends on the rsize/wsize divided by the MTU; for example, 65536/1500 = 43 frames plus additional headers. Thus with NFS over TCP, 64KB read and write size usually provides good performance by allowing a single RPC to transmit or receive a large amount of data.

By default, in the newer kernels of RHEL 5.x and 6.x, the default rsize and wsize are set to 65536 (64k). The early RHEL5.x kernel had a default rsize and wsize value set to 32768 (32k). Usually on clean high-performance networks, or with NFS over TCP, you can improve NFS performance by explicitly increasing these values. However, this has changed in 2.6-based kernels. The default rsize and wsize are set to 65536 unless otherwise specified. Currently the default transport is TCP, which is highly recommended in all kinds of workloads.

In Linux, the rsize and wsize mount options have additional semantics compared with the same options implemented in other operating systems. Normally, the Linux client caches application write requests, issuing NFS WRITE operations when it has at least wsize bytes to write. The NFS client

often returns control to a writing application before it issues any NFS WRITE operations. It also issues NFS READ operations in parallel before waiting for the server to reply to any of them. If the client is requesting to read a 128k block of data, then it issues eight concurrent 16k read requests to the NetApp storage. If rsize is set below the system's page size (4KB on x86 hardware), the NFS client issues individual read operations one at a time and waits for each operation to complete before issuing the next read operation. If wsize is set below the system's page size, the NFS 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. This limitation has been removed in 2.6 kernels so that all read and write traffic is asynchronous whenever possible, independent of the transfer size settings. 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.

In the 2.6-based kernel, the memory manager deals with memory in 4KB pages on x86 systems. The actual page size is architecture dependent. For most uses, pages of this size are the most efficient way for the memory manager to deal with memory. Some applications, however, make use of extremely large amounts of memory. Large databases are a common example of this. For every page mapped by each process, page-table entries must also be created to map the virtual address to the physical address. If you have a process that maps 1GB of memory with 4KB pages, it would take 262,144 page-table entries to keep track of those pages. If each page-table entry consumes 8 bytes, then that would be 2MB of overhead for every 1GB of memory mapped. This is quite a bit of overhead by itself, but the problem becomes even worse if you have multiple processes sharing that memory. In such a situation, every process mapping that same 1GB of memory would consume its own 2MB's worth of page-table entries. With enough processes, the memory wasted on overhead might exceed the amount of memory the application requested for use.

One way to help alleviate this situation is to use a larger page size. Most modern processors support at least a small and a large page size, and some support even more than that. On x86, the size of a large page is 4MB, or 2MB on systems with physical address extension (PAE) turned on. Assuming that a large page size of 4MB is used in the example mentioned earlier, that same 1GB of memory could be mapped with only 256 page-table entries instead of 262,144. This translates to only 2,048 bytes of overhead instead of 2MB.

The use of large pages can also improve performance by reducing the number of translation lookaside buffer (TLB) misses. The TLB is a sort of cache for page tables that allows virtual to physical address translation to be performed more quickly for pages that are listed in the table. Of course, the TLB can hold only a limited number of translations. Large pages can accommodate more memory in fewer actual pages, so as more large pages are used, more memory can be referenced through the TLB than with smaller page sizes.

3.3 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, which you can adjust with the `retry` mount option, 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 a multiuser boot. If you mount /usr or other critical file systems from an NFS server, you should consider using `fg` for these mounts. The `retry` mount option has no effect on foreground 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. Note that the storage also has a `nosuid` export option that does roughly the same thing for all clients accessing an export. Interestingly, the storage's `nosuid` export option also disables the creation of special devices. If you notice programs that use special sockets and devices (such as "screen") behaving strangely, check for the `nosuid` export option on your storage.

To enable Kerberos authentication on your NFS mounts, you can specify the `sec=krb` mount option. In addition to Kerberos authentication, you can also choose to enable authentication with request integrity checking (`sec=krb5i`), or authentication with privacy (`sec=krb5p`). Note that most Linux distributions do not yet support krb5p as of the current writing.

3.4 Tuning NFS Client Cache Behavior

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, or to improve performance for metadata-intensive workloads.

There are a few indirect ways to tune client-side caching. The most effective way 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 the cache turnover rate. You should see fewer read requests and faster client response time as a result.

Some working sets may or may not fit in a client's RAM cache. The clients may have 16GB to 128GB or more RAM, 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 it beneficial to keep the client's cache small.

Normally, for each file in a file system that has been accessed recently, the 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 that 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 it means that it might take some time before an application running on one client sees changes made by applications on other clients. In a pure "read only" that does not have a single writer writing to a file(s), the `nocto` mount option can be used to reduce the amount of additional GETATTRs generated from the close-to-open consistency. The `cto` mount option is enabled by default and should be ON in any read and write workload environment.

In a high-file-count scenario that has a deep directory structure, the Directory Name Lookup Cache (DLNC) on the RHEL client that is autotuned can be further optimized if the application tries to read the files immediately after they are created. Setting "lookupcache=pos" (by default it is set to ALL) does not cache negative LOOKUPs, meaning that you can look up "foo"; if it doesn't exist, the next time you look it up it will go to the NetApp storage to check again. If you look up "bar" and it exists, the next time you look it up it will get it from the cache and not go to the NetApp storage. However, this will reduce efficiency, because every LOOKUP for an entry that doesn't exist will generate a new LOOKUP request. This will have a major impact on builds, for example, or on launching programs that always look in the current directory for the dynamic loadables. You really only want to use "lookupcache=pos" when you know that you have files that are created rapidly and when you need to detect that on another client. This is used in environments that use creation of files between distributed applications to notice when one of the clients can proceed after the first client has created the file. Additionally Flash Cache™ intelligent caching on NetApp storage can cache the metadata when the client cache does not have it in its DLNC.

Clients check back with the server every so often to verify that cached attribute information is still valid. However, adding RAM on the client will not affect 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. The `nocto` mount option reduces the revalidation rate even further, at the expense of cache coherency among multiple clients. The `nocto` mount option is appropriate for read-only mount points where files change infrequently, such as a lib, include, or bin directory; static HTML files; or image libraries. In combination with judicious settings of `actimeo` you can significantly reduce the number of on-the-wire operations generated by your NFS clients. Test this setting with your application to verify that it will tolerate the delay before the NFS client notices file changes made by other clients and fetches the new versions from the server.

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.

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 mount option 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 causes performance degradation on typical workloads, but some common workloads, such as sequential write workloads, can have a high impact. Database workloads that consist of random reads and writes are generally less affected by `noac`.

The `noac` mount option generates a very large number of GETATTR operations and sends write operations synchronously. Both of these add significant protocol overhead. This 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 verify that 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, an NFS client in RHEL 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 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. Further improvements available in the DIRECTIO code path are discussed in section 2.4.

To enable direct I/O, the application must open the file(s) with the O_DIRECT flag. The good part about this is that in a volume, some files can be opened with O_SYNC and some can be opened with O_DIRECT as per needs. In general this will degrade performance, but it is useful in special situations, such as when applications do their own caching. File I/O is done directly to/from user

space buffers. The I/O is synchronous; that is, at the completion of a read or write, data is guaranteed to have been transferred.

If a file is opened with no special open flags, the client writes normally. It can delay (cache) writes until the application does an `fflush`, `fsync`, or `close`, at which point any preceding writes are flushed to the server and committed to disk. System memory pressure may also cause the client to push writes to the server.

If a file is opened with O_SYNC, the client writes normally on the wire, but it always guarantees that all dirty data is flushed to the server's disks before the write(2) system call returns to the application.

If a Linux NFS file system is mounted with the `sync` mount option, then the client writes everything in page-sized chunks, one write at a time and in ascending byte offset order, and guarantees that all dirty data is flushed to the server before the write(2) system call returns to the application.

All of this is entirely independent of the NFS version 3 unstable write mechanism. This is a little oversimplified, but, to write data, the client has two choices:

- Write the data with UNSTABLE writes.
- Write the data with DATA_SYNC or FILE_SYNC writes.

The server can reply to each write request in one of two ways:

- The write was committed to permanent storage, in which case the server replies, "this was a DATA_SYNC or FILE_SYNC write."
- The write was saved in volatile storage, and a subsequent COMMIT request is required to guarantee that the data is permanently saved, in which case the server replies, "this was an UNSTABLE write."

The client looks at the reply to each write. If any of the writes come back marked UNSTABLE, then the client must do a subsequent COMMIT before it reclaims dirty pages for other use. A COMMIT can be a range of bytes or the whole file.

The key here is that the O_SYNC / `sync` mount option semantic determines whether the client holds up the application until the writes are on the server's disk. The client can choose to use either UNSTABLE or FILE_SYNC writes when writing data to the server to meet the requirements of O_SYNC or the `sync` mount option.

`fsync()` waits for all dirty data in a file to be written to storage before allowing the application to regain control. This guarantee is provided by the NFS client. `sync()`, which flushes all dirty data for all file systems to permanent storage, but it does not wait for the flushes to complete.

Data ONTAP's NFS server promotes all write requests to FILE_SYNC, whether the client requests UNSTABLE, DATA_SYNC or FILE_SYNC writes. Thus the client never has to send a COMMIT request when it is writing to NetApp storage because all the writes are logged in the NVRAM and the clients get an acknowledgement for the writes right away. Hence the writes with NetApp storage are asynchronous in nature and are much faster.

For some servers or applications, it may be a requirement 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: The client still flushes its data cache and 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.

3.5 Mounting with NFS Version 4

NetApp recommends RHEL5.7 and later kernels to mount file systems over NFSv4. We highly recommend using NFSv4 in your production environment with Data ONTAP 7.3 and later. Clustered Data ONTAP 8.1 and later also support NFSv4.

The Linux NFS client now recognizes two different file system types. The nfs file system type uses the `vers=` mount option to determine NFS version 3 when communicating with a server. The nfs4 file system type supports only NFS version 4 and does not recognize the `vers=` mount option. If you have scripts that specify particular file system types to act on NFS file systems, you need to modify them to work with both nfs and nfs4 file systems.

An example of mounting NFSv4 using /etc/fstab:

```
172.17.44.102:/vol/vol1         /b1      nfs4
rw,bg,hard,rsize=65536,wsize=65536,proto=tcp,suid,timeo=6000
```

Compared to an NFSv3 mount entry in the /etc/fstab:

```
172.17.44.102:/vol/vol2         /b2       nfs
vers=3,rw,bg,hard,rsize=65536,wsize=65536,proto=tcp,suid,timeo=600
```

The NFS version 4 wire protocol represents users and groups as strings instead of UIDs and GIDs. Before attempting to mount with NFS version 4, verify that the client's UID mapper is configured and running. Otherwise, the client will map all UIDs and GIDs to the user "nobody." The mapper's configuration file is /etc/idmap.conf. Typically the only change needed is to specify the real domain name of the client so that it can map local UIDs and GIDs on the client to network names. When this is done, start the UID mapper daemon with this command:

```
/etc/init.d/rpcidmapd start
```

As mentioned in section 2.4, RHEL6.4 and Data ONTAP 7.3.3 and clustered Data ONTAP 8.1 and later support "numeric_id" for sec=sys mounts that do not require configuring /etc/idmap.conf file.

To mount a server that supports NFS version 4, use the following command:

```
mount –t nfs4 –o filer:/vol/vol0  /mnt/nfs
```

For additional NFSv4 implementation information, refer to TR-3580 for Data ONTAP operating in 7-Mode and TR-4067 for clustered Data ONTAP implementation information. Some mount options you may be accustomed to using with NFS version 3 are no longer supported with the nfs4 file system type. As mentioned, `vers=` is not supported. The `udp` and `tcp` mount options are no longer supported; instead, use `proto=` if you would like to choose a transport that is not TCP. Data ONTAP follows the NFS version 4 Request for Comments (RFC) 3530. NFS version 4 does not support UDP, so `proto=tcp` is the only protocol that can be used with NFS version 4 when your clients communicate with NetApp storage. Other mount options that do not work with nfs4 file systems include `noacl`, `nocto`, and `nolock`.

3.6 Mounting with NFS Version 4.1

NFSv4.1 is the next release of NFS after NFSv4. NFSv4.1 is a minor release version and has all the functionalities of NFSv4. Unlike NFSv4, it only uses port 2049 and it does not have any other ancillary protocols like portmap, mount, Network Lock Manage (NLM), and Network Status Monitor (NSM). The locking mechanism is lease-based just as in the case of NFSv4. NFSv4.1 has some bug fixes and introduces three new features:

- Sessions
- Directory Delegations
- Parallel NFS (pNFS)

In a typical NFSv3/NFSv4 scenario, every session had a single TCP connection. But with the sessions model, depending on the application, each session can generate multiple TCP connections. These TCP connections can be trucked to generate high bandwidth. But at this time no Linux client supports session trucking. Sessions are used to provide replay cache correctness. The sessions slots are negotiated by the client and the NetApp storage and represent limits to how many NFS requests can be on the wire. The server is required to have the resources for a replay cache that can hold the negotiated number of slot responses and can guarantee "at most once" semantics with the resultant bounded replay cache. The sessions implementation is discussed in section 2.4. RHEL6.1 and later versions support sessions. Clustered Data ONTAP 8.1 and later versions support sessions too.

Directory delegations are not supported by any Linux client at this time.

pNFS is a feature in NFSv4.1. It is designed to improve performance because the metadata is isolated from the data and the control paths. This means that the client talks to the metadata server (MDS), and once the MDS provides the file-layout information to the pNFS-supported client, the client then communicates with the data servers directly. Note that by using pNFS, volumes that are remote

to clients in the NetApp Cluster namespace will have a direct data path access. RHEL6.2 and RHEL6.3 versions were labeled as the "Tech Preview" kernels for pNFS. The RHEL6.4 kernel is the GA version for pNFS. Clustered Data ONTAP 8.1 and later versions support pNFS. For more information on pNFS, refer to TR-4063.

To mount a file system over pNFS on a pNFS-supported client:

```
mount -t nfs -o vers=4.1 <NetApp Storage IP>:/path01 /mnt/pNFSpath01
```

3.7 Mount Option Examples

We provide the following examples as a basis for beginning your experimentation. Start with an example that closely matches your scenario, then thoroughly test the performance and reliability of your application while refining the mount options you selected.

The following mount options are best-practice recommendations for most of the file systems mounted over NFS. Here is an example of mount options that are reasonable defaults. In fact, on many newer Linux distributions, these are the default mount options.

Best Practice

```
mount -o rw,bg,vers=3,tcp,timeo=600,rsize=65536,wsize=65536,hard,intr
```

Note: `intr` is deprecated in RHEL6.x. It is hard coded to `nointr`.

Using the `bg` option means our client will be able to finish booting without waiting for appliances 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 32KB read and write size. Using `timeo=600` is a good default for TCP mounts.

When mounting a group of home directories over a WAN, you might try the following best practice.

Best Practice

```
mount -o rw,bg,vers=3,nosuid,tcp,timeo=600,retrans=2,rsize=65536,
    wsize=65536,hard,intr,nosuid
```

Note: `intr` is deprecated in RHEL6.x. It is hard coded to `nointr`.

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 that 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 65536, are more appropriate. The `timeo=600` option allows the TCP protocol a long time to attempt recovery before the RPC client interferes.

When mounting an appliance from an anonymous FTP or HTTP server, use the following best practice.

Best Practice

```
mount -o ro,fg,vers=3,tcp,timeo=600,retrans=2,rsize=65536,wsize=65536,
    hard,nointr,nocto,actimeo=600
```

Note: `intr` is deprecated in RHEL6.x. It is hard coded to `nointr`.

Here we use the `fg` option so 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. 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.

Again, the `fg` option enables NFS file systems to be 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 enables 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 database 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).

Best Practice

NetApp strongly recommends using NFS version 3 over TCP. In slower networks use TCP instead. Avoid using the soft mount option. Try the special mount options if you need an extra boost in performance.

4 Performance

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

4.1 Linux NFS Client Performance

The Linux NFS client runs in many different environments, from light desktop usage to a database with a dedicated private SAN. In general, the RHEL NFS client can perform as well as most other NFS clients, and better than some, in these environments. However, the default values and the best-practice recommendations for mount options need to be followed while mounting a file system from NetApp storage, and you need to observe network behavior carefully so that the Linux NFS client performs at its best.

Most often, when the subject of tuning is brought up, it is done so in a preemptive fashion, the notion being that various dials can be turned in such a way as to make the network "go faster." Although experience with a fixed and stable set of applications often yields a set of adjustments that can result in superior network performance, this sort of adjustment should not be made unless a particular problem (for example, frame loss or degraded throughput compared to expected norms) has been observed and diagnosed. Simply turning up settings (for example, making buffer/queue lengths larger, reducing interrupt latency, and so on) can actually have a detrimental effect on throughput in many cases.

Take, for example, the buffer bloat problem, in which ever-increasing buffer queue depths result in TCP connections that have congestion windows larger than the link would otherwise allow (due to deep buffering), but they also have huge RTT values since the frames spend so long in the queue. This effectively breaks TCP congestion avoidance.

With that said, when should you tune? The first step in making adjustments to your network stack is to observe a problem. Administrators should monitor the various files and tools in Linux (`/proc/net/dev`, `/proc/net/snmp`, `netstat`, `dropwatch`, `etc`) for excessive frame drops or other odd conditions that signal suboptimal performance. Using the data collected from those sources, a description of the problem can be formulated along with a tuning solution to mitigate the problem. For example, an increase in UDP input errors in /proc/net/snmp indicates that one or more socket-receive queues are full when the network stack attempts to enqueue a new frame to an application's socket. This indicates that the rate at which at least one socket queue is draining is less than the rate at which new packets destined for that socket are arriving. Couple this with some application-level logging that indicates lost data at the application and you can see the need to either drain that application's socket queue faster (by optimizing the application) or increase the depth of that application's socket queue (via the `rmem_default` sysctl or the `SO_RCVBUF` socket option).

Following this process of observation/diagnose/tune iteratively, a set of tunings can be derived that, for a fixed environment, can eventually be deployed preemptively. The point, however, is that initial and ongoing observation is always the key to proper tuning of a network stack.

When using Gigabit Ethernet, verify that both ends of every link have enabled full flow control. `ethtool` can be used to check and enable flow control on RHEL clients. Some switches, particularly midrange Cisco® switches, do not support flow control in both directions. Discuss support for full flow control with your switch vendor so that your gigabit NIC and routers support it properly. In the new 10Gbe network cards, autonegotiation is turned OFF and flow control is set to FULL.

Further, if you use Linux NFS clients and storage together on an unrouted network, consider using jumbo frames to improve the performance of your application. Consult your switch's command reference to verify that 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. Ideally the README file on the network interface driver provides more accurate information about the jumbo frame MTU size. The same value has to be set on the switch ports and on the NetApp storage. When using jumbo frames on more complex networks, verify that every link in the network between your client and server supports them and have the support enabled.

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

For example, on a clean 10 Gigabit Ethernet network, a single Linux client can send up to 1.2GB/sec on the network to an FAS6280. If there is other network traffic or packet loss, write performance from a Linux client on NFS over TCP, performance should remain reasonable. Read performance depends on the size and speed of the client's and the storage's memory. Caching on the storage and on the client plays a big role to improve throughput by increasing the number of IOPs required by the application. The bandwidth is determined by the block size times the number of IOPs and is measured in MB/sec. Due to high concurrency by the application, the network interfaces on the NetApp storage may require aggregation of network ports to handle the high throughput requirement. Aggregating two 10Gbe ports would provide 2.4GB/sec of bandwidth.

Best Practice

NetApp recommends having a clean network over TCP. Verify that the network cards always negotiate the fastest settings and that the NIC drivers are up to date.

4.2 Diagnosing Performance Problems with the Linux NFS 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 that can impact client performance.

Checking for network packet loss is the first thing to do 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.

To check for retransmissions, `nfsstat -c` can be used at the shell prompt. At the top of the output, it lists 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 performance.

NFS over TCP does not show up network problems as clearly as UDP and performs better in the face of packet loss. If the TCP mounts run faster than the UDP mounts, that's a sure indication that the network between the clients and the storage 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 can be viewed 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 an appliance'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.

- If the end-to-end connection between the 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 1 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.
- 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.
- The client's RPC layer allocates a socket for each mount. If by any means these sockets use input and output buffers that are too small on systems that use large rsize or wsize or generate a large number of NFS operations in a short period, then there is a high chance of packet drop. Therefore we highly recommend increasing the size of these buffers as documented in section 6 or using a new kernel that autotunes the socket buffer settings.

If the issues are resolved and still have poor performance, then attempt end-to-end performance testing between one of the 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. If the network is full duplex, run iPerf tests in both directions concurrently so that the network is capable of handling a full load of traffic in both directions simultaneously.

One more piece of network advice: Become familiar with network snooping tools such as `tcpdump` and `ethereal`. In RHEL5, ethereal and tethereal are replaced by wireshark and tshark. On NetApp storage, `pktt` generates trace files in tcpdump format that can be analyzed later on a client. These tools provide the last word in what is really happening on the network between the clients and the storage.

Best Practice

NetApp recommends running both `tcpdump` on a client and `pktt` on the storage at the same time and comparing the traces to determine where the problem lies.

Several options must be specified explicitly to collect clean network traces with tcpdump. Verify that 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 that can set it to zero if there is a need to check 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.

Examples: `tcpdump host 172.17.32.100 -s 256 -w xyz.trc`

In addition, always use filtering to capture just the traffic between the client and the server. Again, this reduces the likelihood that `tcpdump` or the 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. In the RHEL6.x kernel you can also use the "dropwatch" utility to monitor and record any dropped packets.

```
[root@ibmx3650-svl25 ~]# dropwatch -l kas
Initalizing kallsyms db
dropwatch> start
Enabling monitoring...
Kernel monitoring activated.
Issue Ctrl-C to stop monitoring
1 drops at netlink_unicast+251
2 drops at ip_forward+288
1 drops at nf_hook_slow+eb
1 drops at netlink_destroy_callback+19
1 drops at tcp_v4_rcv+aa
1 drops at unix_stream_recvmsg+30d
1 drops at unix_stream_recvmsg+30d
1 drops at ip_forward+288
1 drops at ip_rcv_finish+199
7 drops at nf_hook_slow+eb
1 drops at __brk_limit+1e2a13d7
2 drops at nf_hook_slow+eb
10 drops at nf_hook_slow+eb
4 drops at nf_hook_slow+eb
5 drops at nf_hook_slow+eb
1 drops at ip_forward+288
1 drops at ip_rcv_finish+199
2 drops at ip_forward+288
^CGot a stop message
dropwatch>
```

To find out if the application is competing for CPU resources with the NFS client, RPC client, or network layer on your client system, the `top` program can be used. The `rpciod` process at the top of the listing is an indicator that NFS operations are dominating the CPU on your system. In addition, if the system CPU percentage increases significantly when the application accesses NFS data, this also can indicate a CPU shortage. In many cases, adding more CPUs or faster CPUs helps.

There are certain cases in which 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. Using the "strace" utility is another way of gathering kernel-level information on any behavior of system calls that the client communicates with the NetApp storage.

```
[root@ibmx3650-svl26 ~]# strace mount
execve("/bin/mount", ["mount"], [/* 28 vars */]) = 0
brk(0)                                  = 0x1b367000
mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x2aaeb3484000
uname({sys="Linux", node="ibmx3650-svl26.iop.eng.netapp.com", ...}) = 0
access("/etc/ld.so.preload", R_OK)      = -1 ENOENT (No such file or directory)
open("/etc/ld.so.cache", O_RDONLY)      = 3
fstat(3, {st_mode=S_IFREG|0644, st_size=182290, ...}) = 0
mmap(NULL, 182290, PROT_READ, MAP_PRIVATE, 3, 0) = 0x2aaeb3485000
close(3)                                = 0
open("/lib64/libblkid.so.1", O_RDONLY)  = 3
read(3, "\177ELF\2\1\1\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\0008@/?\0\0\0"..., 832) = 832
fstat(3, {st_mode=S_IFREG|0755, st_size=44224, ...}) = 0
mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x2aaeb34b2000
mmap(0x3f2f400000, 2137216, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) =
0x3f2f400000
```

4.3 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 that the server is sluggish or that there are network problems. If you know the server is down, this indicates that 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 storage.

4.4 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 set up your client correctly and your network is clean but you still suffer from performance or reliability problems, contact experts to help you proceed further.

Currently, there is no single 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. NetApp customers can also search the support database for Linux-related issues. NetApp 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 fixed. If you have ever complained about the Linux NFS client, here is your opportunity to do something about it.

If you find a problem with the Linux NFS client, report it to Red Hat. Red Hat supports an online bug database based on bugzilla. You can access Red Hat's bugzilla instance at http://bugzilla.Red Hat.com/.

When filing a BURT that relates to Linux client misbehavior with an appliance, report:

- The Linux distribution and the Linux kernel release (for example, RHEL6.1)
- The client's kernel configuration (cat /etc/Red Hat-release 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 storage, such as how busy the network is, how many switches and routers there are, the 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` or `cat /proc/meminfo` on RHEL clients to collect most of this
- Include a network trace and/or a dump of debugging messages using `strace`

Most importantly, 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.

Best Practice

If you cannot find what you need in this paper or from other resources, contact Red Hat or NetApp for information on specific solutions on NetApp storage.

5 Additional Services That May Affect NFS Behavior

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

5.1 Telling Time

The clock on your Linux clients must remain synchronized with your storage 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 storage and clients to update their time using this service. After you properly configure a network time service, you can find more information on enabling NTP on your storage in the "Data ONTAP System Administrator's Guide."

Linux distributions usually come with a prebuilt network time protocol (NTP) 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, verify that 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 has difficulty maintaining synchronization with your time servers, you may need to create a new drift file. Verify that 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 works correctly.

Next, 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.

Always keep the date, time, and time zone on your appliance and clients synchronized. Not only will you enable any time-based caching on your clients to work correctly, but you will also make debugging easier by aligning time stamps in client logs and on client network trace events with the appliance's message log and pktt traces.

5.2 Security

Today, most versions of the Linux NFS client support only two types of authentication: AUTH_NULL and AUTH_UNIX. Linux distributions based on 2.6 kernels support Kerberos 5, just as Solaris does today via RPCSEC GSS. Later versions of the NFS protocol (for example NFSv4) support a wide variety of authentication and security models, including Kerberos 5, 5p (privacy), and 5i (integrity). NFSv3 and NFSv4 support Kerberos but there is a difference in the stack in the way they are implemented. NFSv3 has portmap, mount, NFS, NLM, and NSM. Kerberos RPC calls apply only to NFS that uses port 2049. The rest of the ports that are used by remaining services do not communicate over Kerberos; they use AUTH_SYS., though NFSv4 only has a single port 2049 of NFS. Therefore Kerberos is more fine grained in the NFSv4 stack compared to NFSv3. The newer versions of the RHEL6.x kernel support DES and AES128/256 encryption types. NFSv4 is firewall friendly too because it communicates over one single port 2049 with the caveat that callbacks require a separate port to be opened on the client. NFSv4.1 removes this restriction with the introduction of sessions. The callback paths are also over Kerberos in RHEL6.x kernels.

To maintain the overall security of your Linux clients, check for and install the latest security updates from Red Hat. NFS over TCP avoids any reordering of IP fragments caused from packets larger than the maximum transfer unit. This helps in crossing firewalls.

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. Check if there are any rules in `/etc/hosts.deny` that could prevent communication between your client and server.

In a firewall setup the client needs access to port 2049 and possibly to 111. If you use NFSv3, the client also needs access to the rpc.mountd and rpc.statd ports. For NFSv3 in RHEL, the client ports can be forced by "uncommenting" the "LOCKD_TCPPORT=32803" and "STATD_PORT=662" lines in the "/etc/sysconfig/nfs" file. This in turn allows the host firewall (iptables) as well as the network firewall to remain secure by allowing only those ports for NFS access. This configuration change requires a reboot to take effect. The server, which is the NetApp storage in this case, needs access to the rpc.statd callback port on the client (NFSv3) and/or the NFSv4 callback port.

Note that the NFSv4 callback port on the Linux NFS client is accessed by creating a file `/etc/modprobe.d/options-local.conf` containing the line `options nfs callback_tcpport=<portnumber>` and then rebooting in RHEL6.x. However, with NFSv4.1 the callback path uses port 2049 with the help of sessions introduced in this minor version of NFSv4.

Best Practice

Therefore it is highly recommended to use NFSv4.1 over NFSv4 as becomes more easy to configure and manage firewalls through a single NFS port.

There has always been a requirement to support more than 16 groups for any user. Moving to Kerberos is a natural choice because RPCSEC_GSS allows more than 16 groups. But with the changes in Data ONTAP, the user can be a member of more than 16 groups without using Kerberos. We recommend that if you want to use NFS with Kerberos you use NFS over TCP. For more information on Kerberos refer to TR-4073.

You can enable more than 16 groups on the storage system with the hidden options. Once these options are set, the client mounts the exported file system from the storage. For example, say that user1 is part of 50 groups and is trying to log in from a Linux client. The Linux client definitely has the limitation of 16 groups, beyond which it would allow or truncate. When user1 tries to log in to access the NFS share, UIDs and the GIDs are transferred to the storage system. Only 16 groups get to the system. Once the system gets the UID, it uses the same UID and talks to the local group file, NIS, or LDAP to verify that this UID is indeed with 16 groups or part of some more groups, since this is a limitation at the protocol level. Then the local group file (which is not used for practical reasons and is not recommended), NIS, or LDAP sends the system the actual list of GIDs that user1 is part of; in this case it would be 50 GIDs. This additional checking of UID is done by Data ONTAP/clustered Data ONTAP to get around this group limitation without moving to a Kerberos environment.

5.3 Network Lock Manager

The NFS version 3 protocol uses separate side-band protocols to manage file locking. On Linux 2.6 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 of the reasons network stack initialization can be delayed are slow NIC devices, slow DHCP service, and 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 an appliance 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 RHEL clients 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.

Check that netfs is running at the proper init levels:

```
/sbin/chkconfig --list netfs
```

Netfs should be running at init levels 3 and 5.

If netfs is not running at the proper init levels:

```
/sbin/chkconfig --levels 35 netfs on
```

Check that portmap is running at the proper init levels:

```
/sbin/chkconfig --list portmap
```

Portmap should be running at init levels 3 and 5.

If portmap is not running at the proper init levels, set it so it will run at the proper levels:

```
/sbin/chkconfig --levels 35 portmap on.
```

Check for the portmap daemon:

```
$ ps -ef |grep portmap
```

Portmap should be running and owned by the user rpc.

If portmap is not running, start it:

```
/etc/init.d/portmap start
```

Check that nfslock is running at the proper init levels:

```
/sbin/chkconfig --list nfslock
```

nfslock should be running at init levels 3 and 5

If nfslock is not running at the proper init levels, set it so it will run at the proper levels:

```
/sbin/chkconfig --levels 35 nfslock on
```

Check for nfslock:

```
$ ps -ef |grep rpc.statd
```

The daemon rpc.statd should be running and owned by the user rpcuser.

The problem of nfslock (rpc.statd) not running has been encountered many times on 2.4

kernels.

If nfslock is not running, start it:

```
/etc/init.d/nfslock start
```

`rpc.statd` uses `gethostbyname()` to determine the client's name, but lockd (in the Linux kernel) uses `uname -n`. By changing the HOSTNAME= fully qualified domain name, that means that lockd will then use an FQDN when contacting the storage. If a `lnx_node1.iop.eng.netapp.com` and also a `lnx_node5.ppe.iop.eng.netapp.com` are contacting the same NetApp storage, the storage will be able to correctly distinguish the locks owned by each client. Therefore, we recommend using the fully qualified name in `/etc/sysconfig/network`. In addition to this, `sm_mon -l` or `lock break` on the storage will also clear the locks on the storage, which will fix the lock recovery problem.

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 always using a fully qualified domain name when setting up a client's nodename during installation. With multihomed hosts and aliased hostnames, you can use the `rpc.statd -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 verify that 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 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 NFSv3 can affect the performance of your application. The NFSv3 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. Note that this is not the same as the `nolock` mount option in Linux. This mount option disables NLM calls by the client, but the client continues to use aggressive cache purging and flushing. Essentially this is the opposite of what Solaris does when `llock` is in effect.

During a storage takeover/giveback from panic/reboot or during a LIF migrate in clustered Data ONTAP, the surviving partner assumes the identity of the dead partner to the MAC address level when a node dies in the HA pair. This takeover is a simulated reboot of the dead partner. (Clients accessing the live partner are unaffected.) The takeover replays the NVRAM log for the failed partner and starts the services up for the failed partner. One service is a piece of code that supports locking for all protocols, including CIFS, NFSv2v3, and NFSv4, where NLM (the network lock manager protocol) is specific to NFS that provides locking support for NFSv2/v3 clients. So the newly

"rebooted" server sends a reboot notification to any NFS client holding open locks and allows the client to reclaim its locks before enabling new lock acquisition.

NetApp handles the lock a little aggressively here. Although we assume that the storage giveback will complete within 30 seconds, we do not give out new lock requests during that time but queue them up. We hold the locks for 45 seconds to allow the original clients to reclaim the locks; otherwise we give them away to new clients in the queue requesting for those thereafter. This is standard recovery of lock state for NFS; it is transparent to the applications and managed completely by the NFS client and NetApp storage. However, it recently was discovered that the Linux clients never requested a resend of the locks once the grace period on the NetApp storage expired. This caused the lock request to fail with an ENOLCK error on the client, a problem that is fixed in RHEL6.4.

With NFSv4.x, the chance that these failures with locks will occur is lower because it is a stateful protocol and all state information is stored on both the client and the server when they are active and recovered mutually in the event of an outage. In NFSv4.x, ***nfsd*** is the only daemon required to start the nfs service. Ancillary protocols like ***portmapd***, ***mountd, lockd,*** and ***statd*** are no longer present. With the elimination of these adjunct protocols, the locking mechanism is streamlined and an Oracle Database faces fewer challenges when recovering locks on startup.

NFSv4.x locks provide a time-bounded grant of control over file state to an NFS client. During the lease interval, the NFS server might not grant conflicting control to another client. Holding a lease allows a client to assume that its lock will remain valid for a server-specified, renewable time interval. The client is responsible for contacting the NFS server to refresh the lease to maintain a lock at the end of every lease interval. (The lease interval defaults to 30 seconds in NetApp Data ONTAP.)

Lease expiration is considered a failure in communications between the client and server, requiring recovery. The server assumes that the client has failed and may allow other clients to acquire the same lock. If the NFS server fails, on reboot it waits the full lease interval for clients to reclaim locks before allowing new lock requests. Leases enable cache consistency and are kept short to prevent delays in normal operations. Longer lock lease intervals reduce lease refreshes. Earlier RHEL5.x clients were chatty because of the overhead caused from the lease check communication that happens between NetApp storage and NFS clients. However, the newer RHE6.x clients are less chatty and thus there is lower overhead.

Leases protect against loss of locking state by the client. A client normally exists in one of two states: Either all locks are correct or all are lost. The refresh of any lock by a client validates all locks held by the client. This reduces the number of lease refreshes by a client from one per lock each lease interval to one per client each lease interval.

6 Tuning Options

The tuning options listed in this section do not apply to all types of workloads. Some tuning options work for random workloads while others help sequential workloads. It is very important that you have adequate information about the application workload and access pattern before applying any of these changes to any of the RHEL clients. NetApp always recommends using the default values on the RHEL clients. Tuning of the default parameters on the RHEL clients should be done in consultation with the appropriate technical reports and subject matter experts on specific applications that run on RHEL clients and integrate with NetApp storage.

6.1 TCP Tuning

Ideally, in an NFS environment that uses TCP for transport, enlarging the transport socket and memory buffers that the Linux client uses for NFS traffic helps reduce resource contention on the client, reduces performance variance, and improves maximum data and operation throughput. In the newer 2.6. kernels, the client automatically chooses an optimal socket buffer size based on the system memory. However, in high-latency environments over WAN or across data centers that span metropolitan areas where file transfer is slow over fast connections, these values need to be calculated appropriately.

The following parameters are responsible for default and maximum receive TCP socket buffers in bytes:

```
net.core.rmem_default = 4194304
net.core.rmem_max = 4194304
```

The following parameters are responsible for default and maximum send TCP socket buffers in bytes:

```
net.core.wmem_default = 4194304
net.core.wmem_max = 4194304
```

The values in the previous examples were received from a RHEL5.4 client with 32GB of system memory. These values would be different on different RHEL versions and values depend on the client system's memory size.

As mentioned above, the default values are always used for "_max," but in high-latency environments the following steps need to taken to calculate the "_max" values.

- We can measure the round-trip latency between two locations that are geographically apart by "pinging" from one location to another.

  From the client at Location A: "ping –s <1514-100> <Server IP address>". The round-trip value comes to 50ms, or .050 secs.

- The maximum TCP window size in bytes = Bandwidth-in-bits-per-second * Round-trip-latency-in-seconds = TCP window size in bits / 8

  A 10GBe connection with a round-trip latency of .050 secs would come to:

  1,000,000,000 bits per sec * .050 sec = 500,000,000 bits/sec

  500,000,000/8 bits per byte = 62,500,000 bytes

- Now the "_max" values can be set as follows:

```
sysctl -w net.core.rmem_max=62500000
sysctl -w net.core.wmem_max= 62500000
sysctl -w "net.ipv4.tcp_rmem=4096 16384 62500000"
sysctl -w "net.ipv4.tcp_wmem=4096 16384 62500000"
```

Some customers have found the following settings on NetApp storage to help performance in WAN and high-performance LAN network environments. Use these settings only after thorough testing in your own environment over TCP. The following settings also apply in a large compute farm environment in which thousands of cores access the NetApp storage concurrently.

Netapp storage:

```
nfs.tcp.recvwindowsize      2097152 (2MB)
nfs.ifc.rcv.high       3145728 (1.5 times of the tcp.recvwindowsize)
nfs.ifc.xmt.high            64
```

However, TCP must be preferred over WAN connections or in other high-loss networks. If you use TCP in an environment that has high packet loss, you could adjust the `net.ipv4.tcp_syn_retries` parameter. The `net.ipv4.tcp_syn_retries` parameter specifies the maximum number of SYN packets to send to try to establish a TCP connection. The default is 5; the maximum is 255. The default value corresponds to a connection time of approximately 180 seconds.

6.2 Memory Tuning

`pdblush` and `bdflush` are two daemons that are responsible for flushing the dirty buffers from the RHEL client. `pdflush` kicks in when the number of free buffers hits a threshold and `bdflush` starts when the dirty buffers reach a threshold. Using the default values the writes are normally bursty in nature. The following parameters can be adequately tuned to alleviate the bursty nature of the writes.

Caution: These tunables should be verified in a test environment before applying them in production.

`vm.dirty_ratio` = 40 – The default value is set to 40%. If the value is lower than the default, then pagecache gets less dirty and it is primarily for small I/O streams. When the values are higher than the default, then the pagecache is dirtier and it is for larger I/O streams.

`vm.dirty_background_ratio` = 10 – The default value is 10%. If the value is lower, `pdflush`

starts early when the pagecache is less dirty and there are smaller I/O streams. When this value is higher than the default, then `pdflush` starts later and the pagecache gets more dirty with large I/O streams.

`vm.swappiness=60` - The default value is 60%. This controls how the Linux system reclaims mapped memory of mainly three kinds:
Anonymous memory—Swapping
Mapped file pages—Writing if dirty and freeing
System V shared memory—Swapping
Decreasing this value provides more aggressive reclaiming of unmapped memory; increasing this value leads to aggressive swapping of mapped memory; avoid swapping mapped memory; in today's world, in which client platforms have more than 4GB of system memory, it may be worthwhile to test reducing the value to 40 to reclaim clean pages instead of using swap space unnecessarily

The above options can be tried in scenarios in which the application has concurrent writers to the same file or multiple files generating continuous or bursty writes. These options may help to smooth the writes on the NetApp storage.

For more on virtual memory and slab cache tuning, refer to the following link from Red Hat: https://access.redhat.com/knowledge/docs/en-US/Red_Hat_Enterprise_Linux/6/html-single/Performance_Tuning_Guide/index.html.

6.3 Network Tuning

In today's environment, almost all NFS client and server communication happens over 10GBe. NetApp recommends the following tuning for most work workloads to improve the 10GBe performance. The following setting is usually the default for common GbE hardware.

- The TCP transfer window size on the network depends on the number of RPC slots times the rsize/wzise value. In pre-RHEL6.4 environments the `sunrpc.tcp_slot_table_entries` value is set to 16 by default. NetApp recommends manually increasing this value to 128. If a file system is mounted with the rsize/wsize value set to 32k and with the default value of 16, the TCP transfer window size equals (16 * 32). But with a 64k rsize/wsize and 128 RPC slots the TCP window size increases to (64k * 128), thus having a better payload on the wire. To make this parameter persistent across reboots, edit `/etc/init.d/netfs` to call `/sbin/sysctl -p` in the first line of the script so that `sunrpc.tcp_slot_table_entries` is set before NFS mounts any file systems. If NFS mounts the file systems before this parameter is set, the default value of 16 will be in force.

  However in RHEL6.4 this behavior has changed, as discussed in section 2.4. There is no RPC slot limit in RHEL6.4. A TCP transfer window size will be dynamically set based on the 64k rsize/wsize (recommended) value set on the mount options. In some LAN scenarios it has been noted that with a dynamic number of RPC slots in RHEL6.4, the NetApp storage may have its network buffers depleted by a flood of RPC requests from Linux clients. This only happens with file systems mounted over NFSv3.

  The following settings do not apply to file systems mounted over NFSv4.1. If this is not a WAN scenario, NetApp recommends explicitly setting the `sunrpc.tcp_slot_table_entries` value to 128 instead of allowing this parameter to take a dynamic value. The following change does not apply to a WAN setup. To make this change persistent on the RHEL6.4 client across reboots:
    - Create the file `/etc/modprobe.d/sunrpc-local.conf`.
    - Add the following entry:
      - options `sunrpc tcp_max_slot_table_entries=128`

- In recently produced platforms with multicore, enabling CPU frequency scaling helps to improve the 10Gbe performance. By default you are not able to utilize the complete frequency of each core.

```
[root@ibmx3650-svl28 ~]# cat /proc/cpuinfo|grep MHz
```

```
cpu MHz         : 1596.000
cpu MHz         : 1596.000
cpu MHz         : 1596.000
cpu MHz         : 1596.000
cpu MHz         : 1596.000
cpu MHz         : 1596.000
cpu MHz         : 1596.000
cpu MHz         : 1596.000
```

Once the governor is set to performance in the /etc/sysconfig/cpuspeed file, then the CPU frequency scaling has to be restarted.

```
[root@ibmx3650-svl28 ~]# service cpuspeed restart
Disabling performance cpu frequency scaling:               [  OK  ]
Enabling performance cpu frequency scaling:                [  OK  ]

[root@ibmx3650-svl28 ~]# cat /proc/cpuinfo|grep MHz
cpu MHz         : 2261.000
cpu MHz         : 2261.000
cpu MHz         : 2261.000
cpu MHz         : 2261.000
cpu MHz         : 2261.000
cpu MHz         : 2261.000
cpu MHz         : 2261.000
cpu MHz         : 2261.000
```

- The Linux kernel I/O schedulers play an important role in controlling disk access to NetApp storage. By default the I/O scheduler is the Completely Fair Scheduler (cfq), but, depending on the workload type, No Operation (noop) or Deadline (deadline) I/O schedulers also help improve performance.

```
[root@ibmx3650-svl28 ~]# cat /sys/block/sda/queue/scheduler
noop anticipatory deadline [cfq]
```

- The scheduler can be changed on the fly while the client is up and running. However, modifying the `/boot/grub/menu.lst` file and rebooting the Linux client make the change permanent. In the following example the I/O scheduler is now changed to `noop`.

```
[root@ibmx3650-svl28 ~]# echo noop > /sys/block/sda/queue/scheduler
[root@ibmx3650-svl28 ~]# cat /sys/block/sda/queue/scheduler
[noop] anticipatory deadline cfq
```

- In a 10Gbe or larger network pipe scenario, NetApp recommends setting the `net.core.netdev_max_backlog` value to 300000. This helps the packets to be queued in the buffer rather than dropping them and then requesting retransmission.

- The other tuning options that may help to improve the Linux client performance over 10Gbe connections are:
    a. Enable windows scaling - `sysctl -w net.ipv4.tcp_window_scaling=1`
    b. Disable irqbalance – `service irqbalance stop`
       `chkconfig irqbalance off`

6.4 Controlling File Read-Ahead in Linux

Read-ahead occurs when Linux predicts that an application may soon require file data that it has not yet requested. Such a 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.

The 2.6 Linux kernel does not support adjusting read-ahead behavior via a `sysctl` parameter. However, the read-aheads adapt more easily to automatically detecting the size of the I/O request. This change eliminates the need for treating large random I/O as sequential and all of the averaging code that exists just to support this. Tests have indicated that multithreaded sequential reads using the new read-ahead code in the 2.6.9 kernel and later is always faster (20–30%). The read-aheads in RHEL5.7 and later are set to 256, which works very well for random reads and is as much as 50% faster. `blockdev –getra <device_name>` and `blockdev –setra <device_name> 256` help to get and appropriately set the read-ahead.

As always, test your workload with these new settings before making changes to your production systems.

6.5 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:

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

The `sysrq` key is one of the best (and sometimes the only) way to determine what a machine is really doing. It is useful when a system appears to be hung or for diagnosing elusive, transient, kernel-related problems.

```
sysctl -w kernel/sysrq=1; echo t > /proc/sysrq-trigger  kernel.sysrq = 1
```

To turn this off, after the problem occurs:

```
sysctl -w kernel/sysrq=0 echo t > /proc/sysrq-trigger kernel.sysrq = 0
```

Trace messages appear in your system log, usually `/var/log/messages`. To disable them, 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.

6.6 Reporting and Monitoring Tools

The following monitoring and reporting tools can be used to collect data about CPU utilization, memory, and network statistics.

- SAR utility can provide information on CPU, iowait, and idle time.

```
[root@ibmx3650-svl42 ~]# sar 1 3
Linux 2.6.18-308.24.1.el5 (ibmx3650-svl42.iop.eng.netapp.com)   02/02/2013

06:11:27 PM       CPU     %user     %nice   %system   %iowait    %steal     %idle
06:11:28 PM       all      0.00      0.00      0.00      0.00      0.00    100.00
06:11:29 PM       all      0.00      0.00      0.00      0.00      0.00    100.00
06:11:30 PM       all      0.00      0.00      0.00      0.00      0.00    100.00
Average:          all      0.00      0.00      0.00      0.00      0.00    100.00
```

- MPIO can provide more granular information for each core on the Linux platform.

```
[root@ibmx3650-svl42 ~]# mpstat -P ALL
Linux 2.6.18-308.24.1.el5 (ibmx3650-svl42.iop.eng.netapp.com)   02/02/2013

06:12:38 PM  CPU   %user   %nice    %sys %iowait    %irq   %soft  %steal   %idle    intr/s
06:12:38 PM  all    0.01    0.03    0.02    0.05    0.00    0.01    0.00   99.88   1073.56
06:12:38 PM    0    0.01    0.04    0.01    0.12    0.00    0.00    0.00   99.82   1000.74
06:12:38 PM    1    0.03    0.04    0.02    0.01    0.01    0.00    0.00   99.88     11.07
06:12:38 PM    2    0.02    0.04    0.02    0.03    0.01    0.00    0.00   99.89     10.32
06:12:38 PM    3    0.01    0.05    0.02    0.01    0.01    0.00    0.00   99.91     10.37
06:12:38 PM    4    0.00    0.00    0.01    0.01    0.00    0.00    0.00   99.97      0.08
06:12:38 PM    5    0.02    0.00    0.01    0.01    0.00    0.00    0.00   99.95      0.04
06:12:38 PM    6    0.01    0.03    0.04    0.18    0.00    0.03    0.00   99.70     22.61
06:12:38 PM    7    0.01    0.01    0.02    0.04    0.00    0.01    0.00   99.91     18.33
```

- TOP provides information on the different processes and the CPU utilization of each of those. This also provides information on memory and swap usage.

```
[root@ibmx3650-svl42 ~]# top
top - 18:11:36 up 22 days,  8:27,  2 users,  load average: 0.00, 0.00, 0.00
Tasks: 217 total,   1 running, 215 sleeping,   0 stopped,   1 zombie
Cpu(s):  0.0%us,  0.0%sy,  0.0%ni, 99.9%id,  0.1%wa,  0.0%hi,  0.0%si,  0.0%st
Mem:  16377260k total,  9028592k used,  7348668k free,   211608k buffers
Swap: 18415608k total,        0k used, 18415608k free,  7258488k cached

  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
    1 root      15   0 10364  692  580 S  0.0  0.0   0:00.91 init
    2 root      RT  -5     0    0    0 S  0.0  0.0   0:00.22 migration/0
    3 root      34  19     0    0    0 S  0.0  0.0   0:00.00 ksoftirqd/0
    4 root      RT  -5     0    0    0 S  0.0  0.0   0:00.00 watchdog/0
```

- VMSTAT providesmemory and swap information.

```
[root@ibmx3650-svl42 ~]# vmstat
procs -----------memory---------- ---swap-- -----io---- --system-- -----cpu------
 r  b   swpd   free   buff  cache   si   so    bi    bo   in   cs us sy id wa st
 0  0      0 7347976 211620 7259064    0    0     0     1    1    2  0  0 100  0  0
```

- DSTAT and IPTRAF provide detailed information on network traffic.

```
[root@ibmx3650-svl42 ~]# dstat
----total-cpu-usage---- -dsk/total- -net/total- ---paging-- ---system--
usr sys idl wai hiq siq| read  writ| recv  send|  in   out | int   csw
  0   0 100   0   0   0|2933B   12k|   0     0 |   0     0 |1074  1059
  0   0 100   0   0   0|   0     0 | 408B 1192B|   0     0 |1012   960
  0   0 100   0   0   0|   0     0 | 345B  436B|   0     0 |1034   976
  0   0 100   0   0   0|   0     0 | 204B  436B|   0     0 |1010   963
```

Refer to the Interoperability Matrix Tool (IMT) on the NetApp Support site to validate that the exact product and feature versions described in this document are supported for your specific environment. The NetApp IMT defines the product components and versions that can be used to construct configurations that are supported by NetApp. Specific results depend on each customer's installation in accordance with published specifications.

NetApp provides no representations or warranties regarding the accuracy, reliability, or serviceability of any information or recommendations provided in this publication, or with respect to any results that may be obtained by the use of the information or observance of any recommendations provided herein. The information in this document is distributed AS IS, and the use of this information or the implementation of any recommendations or techniques herein is a customer's responsibility and depends on the customer's ability to evaluate and integrate them into the customer's operational environment. This document and the information contained herein may be used solely in connection with the NetApp products discussed in this document.

Go further, faster®

NetApp

www.netapp.com