Quantum®

StorNext® 6

Tuning Guide

6-68046-02, Rev. C

StorNext 6 Tuning Guide, 6-68046-02, March 2018, Product of USA.

Contents

Preface

This manual contains the following chapters:

- Chapter 1: StorNext File System Tuning
- Chapter 2: Allocation Session Reservation (ASR)
- Appendix A: StorNext File System Stripe Group Affinity
- Appendix B: Best Practices

Audience

This manual is written for StorNext 6 operators, system administrators, and field service engineers.

Notational Conventions

This manual uses the following conventions:

Convention	Example
User input is shown in bold monospace font.	**`./DARTinstall`**
Computer output and command line examples are shown in monospace font.	`./DARTinstall`

Convention	Example
User input variables are enclosed in angle brackets.	**http://<ip_address>/cgi-bin/stats**

For UNIX and Linux commands, the command prompt is implied.	`./DARTinstall`
	is the same as
	`# ./DARTinstall`

File and directory names, menu commands, button names, and window names are shown in bold font.	**/data/upload**
Menu names separated by arrows indicate a sequence of menus to be navigated.	**Utilities > Firmware**

The following formats indicate important information:

Note: Note emphasizes important information related to the main topic.

Caution: Caution indicates potential hazards to equipment or data.

WARNING: Warning indicates potential hazards to personal safety.

- Right side of the system - Refers to the right side as you face the component being described.
- Left side of the system - Refers to the left side as you face the component being described.
- Data sizes are reported in base 10 (decimal) rather than base 2^{10} (binary). For example:

  10,995, 116,277,769 Bytes are reported as 11.0 TB (decimal/1000). In binary, this value is 10 TiB (binary/1024).

Product Safety Statements

Quantum will not be held liable for damage arising from unauthorized use of the product. The user assumes all risk in this aspect.

This unit is engineered and manufactured to meet all safety and regulatory requirements. Be aware that improper use may result in bodily injury, damage to the equipment, or interference with other equipment.

WARNING: Before operating this product, read all instructions and warnings in this document and in the *Quantum Products System, Safety, and Regulatory Information Guide*.

ADVARSEL: Læs alle instruktioner og advarsler i dette dokument og i *Informationsvejledning vedrørende system-, sikkerheds- og lovbestemmelser for Quantum produkter, før produktet betjenes*.

AVERTISSEMENT : Avant d’utiliser ce produit, lisez toutes les instructions et les avertissements de ce document et du *Guide d’informations sur le système, la sécurité et la réglementation de Quantum*.

WARNUNG: Lesen Sie vor der Inbetriebnahme dieses Produkts alle Anleitungen und Warnungen in diesem Dokument und im *System-, Sicherheits- und Betriebsbestimmungen-Handbuch für Quantum-Produkte*.

ADVERTENCIA: Antes de hacer funcionar este producto, lea todas las instrucciones y advertencias de este documento y de la *Guía de información normativa, del sistema y de seguridad de los productos de Quantum*.

VARNING: Läs igenom alla instruktioner och varningar i detta dokument och i *Quantums produktsystem, säkerhet och reglerande informationsguide* innan denna produkt används.

ВНИМАНИЕ! Перед началом эксплуатации данного изделия прочтите все инструкции и предупреждения, приведенные в настоящем документе и в *Руководстве по системе, технике безопасности и действующим нормативам компании Quantum*.

警告：本製品を使用される前に、本書と『*Quantum製品システム、安全、規制情報ガイド*』に記載されているすべての説明と警告をお読みください。

경고: 본 제품을 작동하기 전에 본 문서와 *Quantum 제품 시스템, 안전 및 규제 정보 설명서*에 있는 모든 지침과 경고를 참조합니다.

警告：在操作本产品之前，请阅读本文档和 *Quantum 产品系统、安全和法规信息指南*中的所有说明和警告。

警告：操作此產品前，請閱讀本檔案及 *Quantum 產品系統、安全與法規資訊指南*中的指示與和警告說明。

אזהרה: לפני ההפעלה של מוצר זה, קרא את כל ההוראות והאזהרות הכלולות במסמך זה וכן *במדריך המידע בנושא מערכת, בטיחות ותקינה עבור מוצרי Quantum*.

For the most up to date information on StorNext 6, see:

http://www.quantum.com/serviceandsupport/get-help/index.aspx#contact-support

Contacts

For information about contacting Quantum, including Quantum office locations, go to:

http://www.quantum.com/aboutus/contactus/index.aspx

Comments

To provide comments or feedback about this document, or about other Quantum technical publications, send e-mail to:

doc-comments@quantum.com

Getting More Information or Help

StorageCare™, Quantum's comprehensive service approach, leverages advanced data access and diagnostics technologies with cross-environment, multi-vendor expertise to resolve backup issues faster and at lower cost.

Accelerate service issue resolution with these exclusive Quantum StorageCare services:

- **Service and Support Website** - Register products, license software, browse Quantum Learning courses, check backup software and operating system support, and locate manuals, FAQs, firmware downloads, product updates and more in one convenient location. Get started at:

  http://www.quantum.com/serviceandsupport/get-help/index.aspx#contact-support

- **eSupport** - Submit online service requests, update contact information, add attachments, and receive status updates via email. Online Service accounts are free from Quantum. That account can also be used to access Quantum's Knowledge Base, a comprehensive repository of product support information. Get started at:

  http://www.quantum.com/customercenter/

For further assistance, or for training opportunities, contact the Quantum Customer Support Center:

Region	Support Contact
North America	1-800-284-5101 (toll free) +1-720-249-5700
EMEA	+800-7826-8888 (toll free) +49 6131 324 185
Asia Pacific	+800-7826-8887 (toll free) +603-7953-3010
For worldwide support: http://www.quantum.com/serviceandsupport/get-help/index.aspx#contact-support	

Worldwide End-User Product Warranty

For more information on the Quantum Worldwide End-User Standard Limited Product Warranty:

http://www.quantum.com/serviceandsupport/warrantyinformation/index.aspx

Chapter 1: StorNext File System Tuning

This chapter contains the following topics:

Tuning Quality of Service Bandwidth Management (QBM)

Beginning with StorNext 6, the Quality of Service Bandwidth Management feature provides a way to spread I/O bandwidth among a number of clients in a controlled and configurable manner.

This feature can be used to manage all clients accessing a file system. Clients that are not configured explicitly are allocated bandwidth using default values. The classes are:

- **First Come**: Used to prioritize clients actively using bandwidth above new clients.
- **Fair Share**: Used to share I/O bandwidth equally or proportionally between identified clients.
- **Low Share**: Used in conjunction with the main classes, **First Come** and **Fair Share**.

To configure QBM, you must specify the maximum I/O bandwidth for each stripe group. Tools are supplied to assist with measuring the bandwidth.

Configuration of QBM

QBM is best thought of as allowing for two general strategies:

- **First Come Strategy**: This strategy allows active clients to use all of their allotted bandwidth, possibly at the expense of new clients.
- **Fair Share Strategy**: This strategy tries to share bandwidth among all clients. The **Low Share** class is used in conjunction with the **First Come Strategy** and **Fair Share Strategy** to prioritize some clients over other non-configured clients.

Quantum recommends that you choose one strategy, although the **First Come** and **Fair Share** classes can be mixed in a configuration. Choosing a single strategy at the beginning does not preclude using a more complicated strategy later.

Consider a case in which you have 1000MB/s available and four clients that would like to do 300MB/s each. Under the First Come Strategy, the first three clients to request and use 300MB/s will retain their bandwidth, while the fourth client will have to wait for more bandwidth to become available. The **Fair Share Strategy** splits the 1000MB/s between all four clients, such that all of them would have 250MB/s available.

The configuration can be modified by updating the configuration file and then signaling the FSM to read the configuration file.

Overview of QBM

The QBM Allocator on the FSM

There are three allocation classes. The class that a client is assigned to determines how bandwidth is allocated to that client. The higher priority classes can steal bandwidth from lower priority classes.

- **First Come** (**FC**): This is the highest priority class. Clients configured for this class of service will either get all the bandwidth they are assigned or be denied the allocation if that bandwidth is not available.
- **Fair Share** (**FS**): This is the second highest priority class. Clients configured for this class of service get their configured bandwidth, or share the total bandwidth allocated to all clients in this class in proportion to their configured amounts.
- **Low Share** (**DS**): This is the third highest priority class. Clients assigned to this class of service get their configured bandwidth or share the total bandwidth allocated to all clients in this class, in proportion to their configured amounts.

Allocating Bandwidth to a Client

QBM Allocator applies the following rules when allocating bandwidth to a client. Bandwidth is allocated on a stripe group basis.

Initialization

1. Every QoS-enabled stripe group must be configured with its bandwidth capacity. This value is used to set the global free bandwidth pool for the stripe group.
2. All clients are configured with a class, and a minimum and maximum amount of bandwidth. Some clients are explicitly configured, and some use the default values. The minimum value is considered the desired value. This is the value the QBM allocator attempts to give each client.
3. The administrator can configure an optional reserve pool per class.
4. Each client must register with the QBM Allocator. If the QBM configuration has information about this client, this information determines the bandwidth assigned to the client. If the client does not have any configuration, QBM uses the default values.

The rules for processing a client’s bandwidth allocation request are given below. Each rule is applied until the client’s bandwidth request is satisfied, or all the rules have been applied.

1. Locate the registration information for this client. Determine the client’s configured minimum and maximum bandwidth, and the client’s configured class.
2. If a reserve pool is available for this class and has available bandwidth, allocate as much of the bandwidth from this reserve pool as possible.
3. Try to allocate the minimum value from the stripe group’s global free bandwidth pool.
4. If there are clients with more than their minimum bandwidth allocation, take that extra bandwidth away from those clients. Start with the lowest priority clients to the highest priority clients.
5. Steal bandwidth from lower priority clients. Take only what is needed to satisfy the request. If a lower priority client does not have enough to satisfy the remaining needed amount, take all the bandwidth

except what is considered the absolute minimum amount (currently set at 32K) from that client. Take from the lowest priority to the highest priority that is below the requesting client's priority. That is, you cannot steal from your own priority.

6. The FS, and DS classes are sharing classes. If the algorithm has taken bandwidth from any of these classes, redistribute the total bandwidth in those classes. Redistribute the bandwidth in proportion to what the client's minimum bandwidth requirement is, or the client's requested amount. If the client requested less than the client's minimum, redistribute in proportion to that lesser amount. Send the new bandwidth allocations to all affected clients.
7. All clients always get at least the absolute minimum bandwidth amount, even if the total bandwidth is oversubscribed. This prevents clients from freezing up when doing I/O if they have 0 bandwidth allocated.

Freeing Bandwidth on a Client

When a client's bandwidth has been freed, the following rules are used to return the bandwidth. Each rule is applied until all freed bandwidth is returned.

1. If this bandwidth was taken from the class reserve, give it back to the reserve.
2. If there is any oversubscription of bandwidth, used the freed bandwidth to eliminate the oversubscription.
3. If there are any clients that have an allocation less than their requested minimum, give the bandwidth to the highest priority client that currently has bandwidth below its minimum, while trying to satisfy the oldest request first. All clients must be at their minimum bandwidth before the next rule is applied.
4. Give the bandwidth to the highest priority client requesting more bandwidth (above the minimum.)
5. Put the bandwidth into the free pool.

Clients Requesting Less Bandwidth

A client requesting less bandwidth is always granted the request. Use the Freeing Bandwidth on a Client above algorithm.

Clients Requesting More Bandwidth

A client that requests more bandwidth can only get more than its minimum bandwidth if there is bandwidth available in the free pool or reserve pool, or if lower priority clients have extra bandwidth. The following rules apply.

1. Get the bandwidth from the free pool if there is some bandwidth available (above a minimum allocation amount, currently 256K). Give that amount to the client.
2. Take bandwidth from lower priority clients that have extra bandwidth (above their configured minimum requested amount.)
3. If there is no bandwidth available, mark the client's current allocation as wanting more bandwidth and return the client's request with **VOP_EAGAIN**.

Clients Requesting to Restore Bandwidth

Assume that a client has previously asked for less bandwidth than its configured minimum bandwidth. But now the client has determined that it needs to get as much as it can, up to the configured minimum bandwidth. Sometimes, a client may not ask for the configured minimum bandwidth because of class bandwidth sharing. The client may have received an asynchronous message from the QBM Allocator that its bandwidth was reduced (possibly because of sharing).

The last amount the QBM Allocator told it could use is the amount it should ask for when it does the restore. The amount needed is the difference between the current allocation and the requested restore amount. The allocation rules for obtaining the needed amount is the same as for the section Allocating Bandwidth to a Client on page 3.

QBM on the Client

Gating I/O on the client

When gating (throttling) of I/O to a client becomes necessary, the leaky bucket algorithm is used.

- The client’s allocated bandwidth is divided into time periods.
- A token represents a single byte of I/O per time period. Each I/O byte sent to or received from the output device takes one token out of the bucket.
- All I/Os are all or nothing — QBM never performs partial I/Os. If the bucket is empty, no more I/O can be sent to the SG associated with the empty bucket. In this case, the I/O thread is blocked and put on a sleep queue.
- When the bucket is replenished, the sleep queue is checked. If there is a sleeping thread for which there now are enough tokens to satisfy the associated I/O request, the thread is awakened and allowed to perform its I/O operation.
- All I/Os are done in the order they are received. After a thread blocks on a bucket, no other threads can perform any I/O associated with that bucket until the first blocked thread has enough tokens to satisfy its I/O request.

Dynamic Bandwidth Requests

QBM running on a client keeps statistics on usage over two time periods, called the **Fast** and **Slow** periods. The **Fast** period is 1second, and the **Slow** period is 30 seconds. These time periods were selected because they seemed to provide adequate intervals for testing the client’s I/O rate.

The **Fast** period is used to determine if a client needs to restore the bandwidth it was originally given after it has asked for less bandwidth. The heuristic used to determine if a client needs to restore bandwidth is to test if the client has had to sleep at least 50% of the time periods in the **Fast** period. If it has had to sleep at least 50% of the time periods and it has asked for less than what it was allocated, it sends a **Restore_BW** request to the FSM. The bandwidth is then immediately restored, without waiting for a response from the FSM. Using this algorithm, it can take up to 1.44 seconds for the client to determine that it needs to restore the original bandwidth (BW).

The **Slow** period is used to determine if QBM needs to increase or decrease its current bandwidth allocation. The heuristic for asking the FSM to increase the allocated bandwidth is if the client is using at least 80% of its currently allocated bandwidth during the entire **Slow** period. If it has been using at least 80%, the client asks for a 30% increase in the rate. The FSM may grant the entire requested amount of increase, it may grant a partial increase, or it may not grant any increase. The client must wait for the response before the bandwidth allotment is changed. If the FSM did not grant any increase, the request to increase will cause the FSM to set a flag to wait for available bandwidth at a later time. If the client is using less than 60% of its allocated bandwidth during the **Slow** period, it sends a **DEC_BW** request to the FSM asking for a 10% decrease in bandwidth. This request is granted by the FSM. The client immediately decreases its bandwidth rate by 10%.

StorNext File System Tuning

The StorNext File System (SNFS) provides extremely high performance for widely varying scenarios. Many factors determine the level of performance you will realize. In particular, the performance characteristics of the underlying storage system are the most critical factors. However, other components such as the Metadata Network and MDC systems also have a significant effect on performance.

Furthermore, file size mix and application I/O characteristics may also present specific performance requirements, so SNFS provides a wide variety of tunable settings to achieve optimal performance. It is usually best to use the default SNFS settings, because these are designed to provide optimal performance under most scenarios. However, this guide discusses circumstances in which special settings may offer a performance benefit.

Note: The configuration file examples in this guide show both the .cfgx (XML) format used by StorNext for Linux and the .cfg format used by Windows.

For information about locating sample configuration files, see Example FSM Configuration File on page 64.

Note: StorNext supports the use of clients running certain older versions of StorNext. When configuring these clients, refer to the *StorNext Tuning Guide* appropriate for the StorNext version they are running.

Note: If StorNext Appliances are being used, the tuning recommendations found in the appliance-specific documentation should be used in preference to the information in this guide.

StorNext File System Thin Provisioned Capabilities

With thin provisioned storage, you may need the ability to unmap space on storage. For example, if the space is over-provisioned and shared by multiple devices, it could be “over-allocated” and writes to the storage fail, even though the storage and corresponding file system claim there is available space. This usually occurs when the actual storage space is completely mapped; the storage maps the space when it is written and it is never unmapped without intervention by the file system or the administrator. When files are removed, the space is not unmapped by StorNext versions prior to StorNext 6.

StorNext 6 provides two file system level mechanisms that unmap free space in the file system

Note: This functionality is available on Linux MDCs and with the QXS series storage. Quantum recommends you not over-provision your StorNext volumes.

At File System Creation

Beginning with StorNext 6, the **`cvmkfs`** command automatically detects if the LUNs in each Stripe Group (SG) are thin provisioned and QXS series devices. This is done for all the SGs so the MDC needs to have access to all of the storage configured in the file system to do the thin provisioned work.

The storage is notified that all of the file system free space is now available on each LUN. So, if the LUN has previously been written to and thereby contained mappings, these are all “unmapped” allowing the storage that was consumed to be available to other devices. The metadata and journal are written to so they are either re-mapped or left mapped during the run of the **`cvmkfs`** command. If the command is invoked with the **`-e`** option or the **`-r`** option and the file system is not managed, the unmap work is skipped for all stripe groups that can hold user data. The thin provision work is still done for all other stripe groups, for example, metadata only SGs.

The **`-T`** option causes the **`cvmkfs`** command to skip all thin provision work on all stripegroups. This is useful if the administrator knows all the space is already unmapped and the command is failing since some LUNs are not available. Each LUN in each SG that is thin provisioned has a pagesize and maximum unmap size. All the LUNs in a SG must have the same sizes for each. If not, the **`cvmkfs`** command fails. This failure can be bypassed with the **`-T`** option but then all thin provision unmap work is skipped on all SGs.

Note: Do not configure SGs using LUNs with different pagesizes or maximum unmap sizes in the same SG.

Unmapping Free Space

Beginning with StorNext 6, the **`cvfsck`** command has been supplemented to perform thin provision unmap operations of free space for a given file system. The machine running the command must have access to all of the LUNs in the file system in order to unmap space on them. This is done by executing the following commands:

```
cvadmin -e 'stop <fsname>'
cvfsck -U <fsname>
cvadmin -e 'start <fsname>'
```

This unmaps all free space in the file system. If the file system has the **`AllocSessionReservationSize`** parameter set to non-zero and there are active sessions, any chunks that are reserved for Allocation Sessions, are not unmapped.

To unmap **ALL** free space including the session chunks, execute the following commands to stop all writes and make sure all data is flushed:

```
cvadmin -e 'stop <fsname>'
cvadmin -e 'start <fsname>'
/bin/ls <mount point>
sleep 2
cvadmin -e 'stop <fsname>'
cvfsck -U <fsname>
cvadmin -e 'start <fsname>'
```

Unmapping Free Space After Adding a Stripe Group

Beginning with StorNext 6, after adding a Stripe Group with **cvupdatefs** or with **sgadd**, execute the **cvfsck -U *<fsname>*** command as indicated in the Unmapping Free Space on the previous page section to unmap any existing mapping for that SG as well as all the others.

Determining the Relationship Between Mapped Space and Free Space

Administrators can compare the free/allocated space on a given Stripe Group with the amount of unmapped/mapped space on that Stripe Group. To do so, execute the following command:

```
cvadmin -e 'select <fsname>;show'
```

Note the amount of free/allocated space on a given Stripe Group.

Then, execute the following command on each LUN in that SG and add up all of the unmapped/mapped space for each LUN:

```
sn_dmap
```

Some space on LUNs in not available to the file system so do not expect exact matches in the totals.

Note: Mapped space occurs when unmapped blocks are written and that allocated space in the file system may not all be written. This occurs with optimistic allocation and pre-allocation that is not necessarily entirely written. So unmapped space is typically higher than free space when a new file system is written to. As files are removed, the unmapped space will not increase as the free space increases. If the free space is significantly larger than the unmapped space, execute the **cvfsck -U** command to increase the unmapped space.

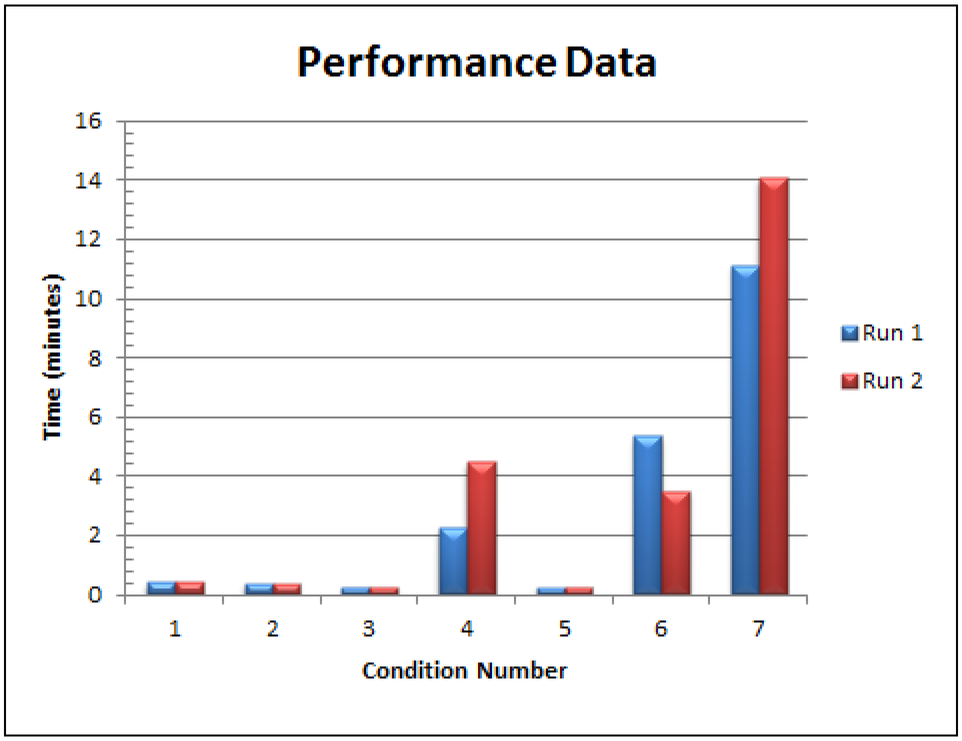
Performance Metrics

The performance of **`cvfsck -U <fsname>`** to unmap thin provisioned LUNs varies significantly. Multiple performance measurements were conducted using **`cvfsck -U <fsname>`** under the following seven conditions:

Note: The system environment consisted of a 14 TB file system containing 3 LUNs (each could consume up to 4.7 TBs).

1. After the initial **`cvmkfs`**.
2. After writing many files and mapping about 8.1 TB.
3. After filling the file system.
4. After removing ½ the files leaving 7 TB used.
5. After re-filling the file system.
6. After removing a different set of files …about ½ the files leaving 7.2 TB used.
7. After removing all the files.

Condition When cvfsck –U Ran	Run 1	Run 2
1	0 min 24.8 secs	0 min 24.7 secs
2	0 min 22.4 secs	0 min 21.8 secs
3	0 min 15.3 secs	0 min 15.5 secs
4	2 min 15 secs	4 min 28 secs
5	0 min 15 secs	0 min 15.5 secs
6	5 min 20 secs	3 min 29 secs
7	11 min 7 secs	14 min 2 secs



The results indicate the performance of `cvfsck -U <fsname>` to unmap thin provisioned LUNs varies significantly. Additionally, the unmap operations in the system continue for several seconds, as they continue to run in the background.

Expand a StorNext LUN

Until recently, the only supported way to expand storage in a StorNext file system was to add a stripe group. New array technology now makes it possible in some cases to expand a LUN by adding more storage to the underlying volume. This topic provides the procedure for expanding a LUN. LUN expansion is supported beginning with StorNext 6.

Note: For the purposes of this topic, the terms **volume** and **LUN** are used interchangeably.

Introduction

Having the array manage the allocation and mapping of the blocks onto the physical storage is commonly known as thin-provisioning. On a new thin-provisioned volume all the blocks (LBAs – Logical Block Addresses) are unmapped and unallocated. Only when an LBA or LBA range is first written, is it allocated and mapped. The allocation and mapping is done in 4 MiByte pages.

Thin-provisioning can offer some advantages:

- If a drive fails and is replaced, the array need only reconstruct mapped LBAs.
- Volumes do not have to be statically carved from underlying RAID groups and can be dynamically expanded.
- Volumes can be over-provisioned.
- Array utilities like volume copy and snapshots can be more efficient.
- Volumes can be dynamically expanded.

Thin-provisioning also has some disadvantages:

- The array mapping mechanism does not know when the file system deletes blocks so an unmap or trim operation has to be done to free mapped/allocated blocks.
- Because the array handles the allocation, it essentially nullifies any optimizations that the StorNext allocator has made.
- All volumes in a storage pool are allocated on a first write basis so a checkerboard allocation among streams is assured.
- Performance is inconsistent.

Caveats for Volume Expansion

- Clients and FSM must be running at StorNext 6.0 (and later).
- All clients must have multipath configured appropriately.
- Windows and Linux clients can dynamically resize the StorNext file system without unmounting. With other clients, it may be necessary to unmount the file system prior to resize, and then remount the file system after resizing.
- If a StorNext stripe group has more than one volume, all the volumes in the stripe group must be expanded to the same size.

The Volume Expansion Process

The volume expansion process consists of several tasks that need to be completed in sequence. This is a bottom up process that begins in the array and pushes up into the FSM and clients.

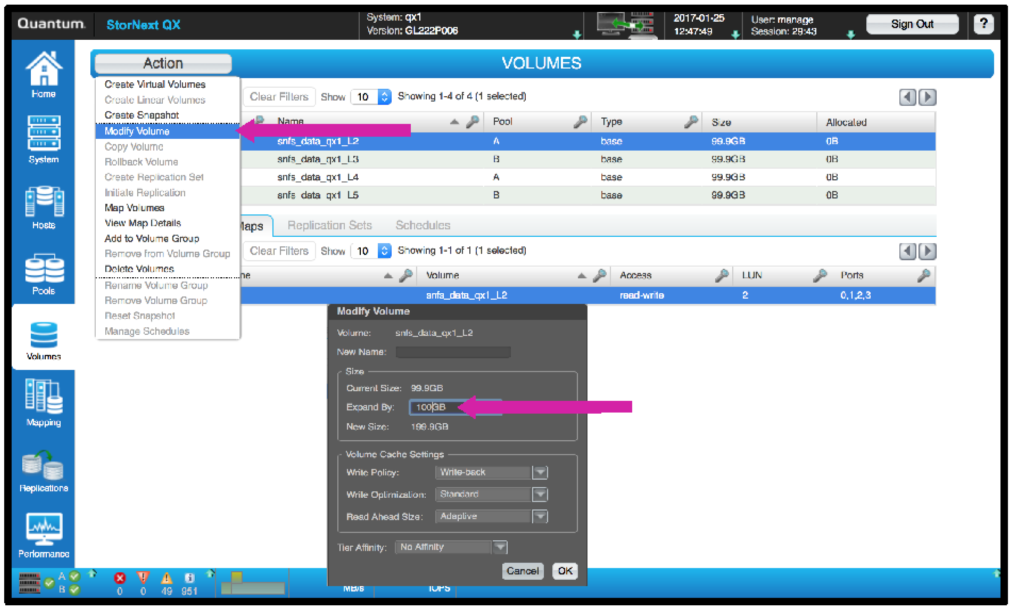
1. Expand the Volume on the QXS Array on the next page
2. Unmount StorNext file systems on non-Linux, Windows, or StorNext clients that are running StorNext 5.4.0.1 (or earlier)
3. Update Linux to the New Size on the next page
4. Rewrite the StorNext Label(s) to Reflect the Size Change on page 13
   a. Update the file system configuration to reflect the size change.
   b. Instantiate the new configuration in the FSM.

5. Resize the StorNext Stripe Group on an Active File System on page 14
6. Resize the StorNext Stripe Group on an Inactive File System on page 15

Expand the Volume on the QXS Array

This task assumes basic knowledge of the QXS array Web Based Interface (WBI). In this example, we will expand a StorNext data volume by 100 GBytes, effectively doubling its size.

1. Log in to the array and select volumes from the left column.
2. Select the volume to expand.
3. Click below on the selected volume
4. In the **Action** menu, click **Modify Volume**.
5. In the **Expand By** field, input the desired amount.
6. Click **OK**.



System: qx1
Version: GL222P006
2017-01-25
12:47:49
User: manage
Session: 29:43

Showing 1-4 of 4 (1 selected)

Name	Pool	Type	Size	Allocated
snfs_data_qx1_L2	A	base	99.9GB	0B
snfs_data_qx1_L3	B	base	99.9GB	0B
snfs_data_qx1_L4	A	base	99.9GB	0B
snfs_data_qx1_L5	B	base	99.9GB	0B

Showing 1-1 of 1 (1 selected)

Volume	Access	LUN	Ports
snfs_data_qx1_L2	read-write	2	0,1,2,3

Modify Volume
Volume: snfs_data_qx1_L2
New Name:
Size
Current Size: 99.9GB
Expand By: 100GB
New Size: 199.9GB
Volume Cache Settings
Write Policy: Write-back
Write Optimization: Standard
Read Ahead Size: Adaptive
Tier Affinity: No Affinity

Note: StorNext requires all the volumes in a stripe group to be the same size. If there is more than one volume that you need to expand, do so now.

Caution: Over-provisioning is not recommended because unmap/trim is not fully supported in StorNext. Verify that your volumes do not exceed the physical storage available.

Update Linux to the New Size

The Linux block device stack hot-plug infrastructure does not automatically update the size of SCSI device when it is changed. You must explicitly force a rescan of the SCSI devices so they incorporate the new size.

A utility, **sn_scsi_resize**, initiates a rescan of all SCSI block devices based on their DM/Multipath instantiation. The utility, **sn_scsi_resize**, also executes a StorNext "disks refresh" command via **cvadmin**.

```
/usr/cvfs/bin/sn_scsi_resize
```

Important

On a StorNext appliance, you must execute the utility, **sn_scsi_resize**, on both nodes of the HA pair.

Rewrite the StorNext Label(s) to Reflect the Size Change

1. Execute the command, **`cvlabel -l`**; you have resized **snfs_data_qx1_L2**.

   Note: The **Sectors** field and the **Maximum sectors** field are different. The **Sectors** field refers to the value from the disk label, and the **Maximum sectors** field refers to the actual size of the volume.

```
per1-# cvlabel -l
/dev/mapper/mpatha [Quantum StorNext QX G22x] SNFS-EFI " snfs_data_qx1_
L2"Sectors: 195287007.
Sector Size: 512. Maximum sectors: 390592479. Stripebreadth: 0.
/dev/mapper/mpathb [Quantum StorNext QX G22x] SNFS-EFI "snfs_data_qx1_
L3"Sectors: 195287007.
Sector Size: 512. Maximum sectors: 195287007. Stripebreadth: 0.
/dev/mapper/mpathc [Quantum StorNext QX G22x] SNFS-EFI "snfs_data_qx1_
L4"Sectors: 390592479.
Sector Size: 512. Maximum sectors: 390592479. Stripebreadth: 0.
/dev/mapper/mpathd [Quantum StorNext QX G22x] SNFS-EFI "snfs_meta_qx1_
L5"Sectors: 195287007.
Sector Size: 512. Maximum sectors: 195287007. Stripebreadth: 0.
```

2. The new size is 390592479; you must re-label the volume with the new size.

```
per1-# cvlabel -c | grep mpatha >/tmp/label
per1-# cat /tmp/label
snfs_data_qx1_L2 /dev/mapper/mpatha 195287007 EFI 0 # host 0 lun 2 sectors
195287007 sector_size 512
inquiry [Quantum StorNext QX G22x] serial 600C0FF0001BE35788018F5801000000
```

3. Edit the label file and change the sectors field to reflect the new size.

```
per1-# cat /tmp/label
snfs_data_qx1_L2 /dev/mapper/mpatha 3907010703 EFI 0 # host 0 lun 2 sectors
195287007 sector_size 512
inquiry [Quantum StorNext QX G22x] serial 600C0FF0001BE35788018F5801000000
```

4. Rewrite the label.

```
per1-# cvlabel -r /tmp/label
*WARNING* This program will over-write volume labels on the devices specified
in the file "/tmp/label". After execution, the devices will only be usable by
the StorNext. You will have to re-partition the devices to use them on a
different file system.
Do you want to proceed? (Y / N) -> y
/dev/mapper/mpatha [Quantum StorNext QX G22x] SNFS-EFI "snfs_data_qx1_L2"
Controller '208000C0FF1A60DA', Serial '600C0FF0001BE35788018F5801000000',
Sector Size 512, Sectors 195287007 (100.0GB), Max 390592479, Stripebreadth 0,
GUID 9ab6057c-e6fe-11e6-ba37-0024e8636400 [Mon Jan 30 09:13:07 2017
00:24:e8:63:64:00]
```

The disk **/dev/mapper/mpatha** has a valid label, and is changing from 195287007 to 390592479 sectors.

```
Do you want to re-label it SNFS-EFI - Name: snfs_data_qx1_L2 Sectors:
390592479 (Y / N) -> y
New Volume Label -Device: /dev/mapper/mpatha SNFS Label: snfs_data_qx1_L2
Sectors: 390592479.
Done. 1 source lines. 1 labels.
Requesting disk rescan .
```

5. Verify the label change.

```
per1-# cvlabel -l | grep mpatha
/dev/mapper/mpatha [Quantum StorNext QX G22x] SNFS-EFI "snfs_data_qx1_
L2"Sectors: 390592479. Sector Size: 512. Maximum sectors: 390592479.
Stripebreadth: 0.
```

Resize the StorNext Stripe Group on an Active File System

On an active file system, execute the command, **`sgmanage`**, to resize an active file system. You must provide the file system name and the stripe group name.

```
per1-# sgmanage --resize -f jhb -g DataFiles0
fs: jhb Active: yes Port: 48299 Pid: 16383 IP: 10.65.186.149
Stripe Group 1 [DataFiles0]
Status : Up, AllocTrue
Total blocks : 24410112
Free blocks : 24410112
Reserved blocks : 1082880
Allocated blocks : 0 - (0.00% full)
Disk stripes : Read: Enabled, Write: Enabled
Node: 0, Name: snfs_data_qx1_L2, Label: QXDATA
Sectors: 195287007
Resize FSM jhb SG DataFiles0 [1]
New sector count: 390592479
Are you sure you want to continue [y/n]? y
Stripe Group 1 [DataFiles0]
Status : Up, AllocTrue
Total blocks : 48823296
Free blocks : 48823296
Reserved blocks : 1082880
Allocated blocks : 0 - (0.00% full)
Disk stripes : Read: Enabled, Write: Enabled
Node: 0, Name: snfs_data_qx1_L2, Label: GENERIC_390592479_512
Sectors: 390592479
```

The command, **`sgmanage`**, updates the on-disk configuration file, the file system configuration file **/usr/cvfs/config/<*fsname*>.cfgx** and the internal FSM tables.

Resize the StorNext Stripe Group on an Inactive File System

On an inactive file system, execute the command **`cvupdatefs -s <`*`fsname`*`>`**.

1. Edit the file system configuration file, **/usr/cvfs/config/<*fsname*>.cfgx**, so that the new size is reflected in the appropriate stripe group(s).
2. Execute the following command:

```
cvupdatefs –s <fsname>
```

Check an Active File System for an Updated Size

If the stripe group is a StorNext data stripe group, the new size should be reflected by the command, **df**. Below are examples of the output from the command, **df**, before and after for the mounted file system.

Before:

```
per1-# df -B512 /jhb
Filesystem 512B-blocks Used Available Use% Mounted on
jhb 195280896 8663040 186617856 5% /jhb
```

After:

```
per1-# df -B512 /jhb
Filesystem 512B-blocks Used Available Use% Mounted on
jhb 390586368 8663040 381923328 3% /jhb
```

If the stripe group is a StorNext metadata stripe group, you can see the new size by executing the command, **`cvadmin`**; the option is **`show <stripe_group_name>`** or **`disks`**.

Verify the File System is Accessible from Your Client(s)

1. Verify the command output for, **cvlablel –L** does not report unusable on the expanded LUN.
2. In cases where the LUN reports unusable, run the command, **/usr/cvfs/bin/sn_scsi_resize**, on that client.
3. Clients, such as Ubuntu 14.04, require the multi-path driver be restarted, **service multipath -tools restart**, and **/usr/cvfs/bin/sn_scsi_resize** to acknowledge the updated LUN size.
4. Reboot the system if the size is incorrect, if operations are hanging or unresponsive, or if operations are failing.

The Underlying Storage System

The performance characteristics of the underlying storage system are the most critical factors for file system performance. Depending on an environment's use cases, differing performance characteristics may be more important. For very large, sequential, file access, storage throughput will be an important factor. For smaller file access, or random file access, storage access time and I/O latency will be an important factor.

Metadata access is small, random I/O, with most I/O 4KB in size. As such, storage access time and I/O latency are the key factors when tuning for StorNext file operation performance.

Solid state drive, or SSD, has shown advantages when optimizing storage for metadata performance due to its very low I/O latency and high rate of operations per second. Choosing solid state drive within RAID storage provides a good mix of resiliency and small, random I/O performance.

Typically, RAID storage systems provide many tuning options for cache settings, RAID level, segment size, stripe size, and so on.

RAID Cache Configuration

The single most important RAID tuning component is the cache configuration. This is particularly true for small I/O operations. Contemporary RAID systems provide excellent small I/O performance with properly tuned caching. So, for the best general purpose performance characteristics, it is crucial to utilize the RAID system caching as fully as possible.

For example, write-back caching is absolutely essential for metadata stripe groups to achieve high metadata operations throughput.

However, there are a few drawbacks to consider as well. For example, read-ahead caching improves sequential read performance but might reduce random performance. Write-back caching is critical for small write performance but may limit peak large I/O throughput.

⚠ **Caution:** Some RAID systems cannot safely support write-back caching without risk of data loss, which is not suitable for critical data such as file system metadata.

Consequently, this is an area that requires an understanding of application I/O requirements. As a general rule, RAID system caching is critically important for most applications, so it is the first place to focus tuning attention.

RAID Write-Back Caching

Write-back caching dramatically reduces latency in small write operations. This is accomplished by returning a successful reply as soon as data is written into RAID cache, thus allowing the RAID to immediately acknowledge completion of the write I/O operation as soon as the data has been captured into the RAID's cache. Simultaneous to write into cache operations, the RAID writes previously cached data onto the targeted disk LUN storage. The result is minimal I/O latency and thus great performance improvement for small write I/O operations.

Many contemporary RAID systems protect against write-back cache data loss due to power or component failure. This is accomplished through various techniques including redundancy, battery backup, battery-backed memory, and controller mirroring. To prevent data corruption, it is important to ensure that these systems are working properly. It is particularly catastrophic if file system metadata is corrupted, because complete file system loss could result.

⚠ **Caution:** If the array uses write-back caching, Quantum requires that the cache is battery-backed.

Minimal I/O latency is critically important to file system performance whenever the file system processes a large number of files of smaller file sizes. Each file processed requires a metadata small write operation and as discussed above many small write operations I/O latency is improved with RAID write-Back caching enabled. This is easily observed in the hourly File System Manager (FSM) statistics reports in qustats log files: the “PIO Write HiPri” statistic reports average, minimum, and maximum write latency (in microseconds) for the reporting period. If the observed average latency exceeds 0.5 milliseconds, peak metadata operation throughput will be degraded. For example, create operations may be around 2000 per second when metadata disk latency is below 0.5 milliseconds. However, create operations may fall to less than 200 per second when metadata disk latency is around 5 milliseconds.

In contrast to Write-Back caching, Write-Through caching eliminates use of the cache for writes. This approach involves synchronous writes to the physical disk before returning a successful reply for the I/O operation. The write-through approach exhibits much worse latency than write-back caching; therefore, small I/O performance (such as metadata operations) is severely impacted. It is important to determine which write caching approach is employed, because the performance observed will differ greatly for small write I/O operations.

In most cases, enabling Write-Back RAID caching improves file system performance regardless of whether small or large file sizes are being processed. However, in rare instances for some customers, depending on the type of data and RAID equipment and when larger file sizes are being processed, disabling RAID caching maximizes SNFS file system performance.

Most dual controller disk arrays typically use a "write cache mirroring" mechanism to protect against a controller failure. The "write cache mirroring" mechanism is important to ensure data integrity when failover is enabled. However, there is typically a performance impact when using "write cache mirroring". The performance impact varies greatly depending on the I/O workload. Depending on the customers' performance and reliability needs, some customers disable "write cache mirroring" in the array controller's cache settings; disabling "write cache mirroring" can subject the array to both single points of failure as well as data corruption. Quantum's best practice is to enable "write cache mirroring". For LUNs containing metadata, "write cache mirroring" must always be enabled.

Kinds of Stripe Groups

StorNext uses Stripe Groups to separate data with different characteristics onto different LUNs. Every StorNext file system has three kinds of Stripe Groups.

- **Metadata Stripe Groups** hold the file system metadata: the file name and attributes for every file in the file system. Metadata is typically very small and accessed in a random pattern.
- **Journal Stripe Groups** hold the StorNext Journal: the sequential changes to the file system metadata. Journal data is typically a series of small sequential writes and reads.
- **User Data Stripe Groups** hold the content of files. User data access patterns depend heavily on the customer workflow, but typical StorNext use is of large files sequentially read and written. Users can define multiple User Data Stripe Groups with different characteristics and assign data to those Stripe Groups with Affinities; see StorNext File System Stripe Group Affinity on page 88.

Because the typical access patterns for Metadata and User Data are different, Quantum recommends creating different Stripe Groups for Metadata and User Data. Journal data access patterns are similar enough to be placed on the Metadata Stripe Group, or Journal can be placed on its own Stripe Group.

RAID Level

Configuration settings such as RAID level, segment size, and stripe size are very important and *cannot be changed after put into production*, so it is critical to determine appropriate settings during initial configuration.

Quantum recommends Metadata and Journal Strips Groups use RAID 1 because it is most optimal for very small I/O sizes. Quantum recommends using fibre channel or SAS disks (as opposed to SATA) for metadata and journal due to the higher IOPS performance and reliability. It is also very important to allocate entire physical disks for the Metadata and Journal LUNs in order to avoid bandwidth contention with other I/O traffic. Metadata and Journal storage requires very high IOPS rates (low latency) for optimal performance, so contention can severely impact IOPS (and latency) and thus overall performance. If Journal I/O exceeds 1ms average latency, you will observe significant performance degradation.

Note: For Metadata, RAID 1 works well, but RAID 10 (a stripe of mirrors) offers advantages. If IOPS is the primary need of the file system, RAID 10 supports additional performance by adding additional mirror pairs to the stripe. (The minimum is 4 disks, but 6 or 8 are possible). While RAID 1 has the performance of one drive (or slightly better than one drive), RAID 10 offers the performance of RAID 0 and the security of RAID 1. This suits the small and highly random nature of metadata.

Quantum recommends User Data Stripe Groups use RAID 5 for high throughput, with resilience in case of disk error. A 4+1 RAID 5 group would logically process data on four disks, and another disk for parity.

Some storage vendors now provide RAID 6 capability for improved reliability over RAID 5. This may be particularly valuable for SATA disks where bit error rates can lead to disk problems. However, RAID 6 typically incurs a performance penalty compared to RAID 5, particularly for writes. Check with your storage vendor for RAID 5 versus RAID 6 recommendations.

Segment Size and Stripe Size

The stripe size is the sum of the segment sizes of the data disks in the RAID group. For example, a 4+1 RAID 5 group (four data disks plus one parity) with 64kB segment sizes creates a stripe group with a 256kB stripe size. The stripe size is a critical factor for write performance. Writes smaller than the stripe size incur the read/modify/write penalty, described more fully below. Quantum recommends a stripe size of 512kB or smaller.

The RAID stripe size configuration should typically match the `SNFS StripeBreadth` configuration setting when multiple LUNs are utilized in a stripe group. However, in some cases it might be optimal to configure the `SNFS StripeBreadth` as a multiple of the RAID stripe size, such as when the RAID stripe size is small but the user's I/O sizes are very large. However, this will be suboptimal for small I/O performance, so may not be suitable for general purpose usage.

To help the reader visualize the read/modify/write penalty, it may be helpful to understand that the RAID can only actually write data onto the disks in a full stripe sized packet of data. Write operations to the RAID that are not an exact fit of one or more stripe-sized segments, requires that the last, or only, stripe segment be read first from the disks. Then the last, or only portion, of the write data is overlaid onto the read stripe segment. Finally, the data is written back out onto the RAID disks in a single full stripe segment. When RAID caching has been disabled (no Write-Back caching), these read/modify/write operations will require a read of the stripe data segment into host memory before the data can be properly merged and written back out. This is the worst case scenario from a performance standpoint. The read/modify/write penalty is most noticeable in the absence of “write-back” caching being performed by the RAID controller.

It can be useful to use a tool such as **lmdd** to help determine the storage system performance characteristics and choose optimal settings. For example, varying the stripe size and running lmdd with a range of I/O sizes might be useful to determine an optimal stripe size multiple to configure the SNFS **StripeBreadth**.

The deviceparams File

This file is used to control the I/O scheduler, and control the scheduler's queue depth.

For more information about this file, see the **deviceparams** man page, or the StorNext Man Pages Reference Guide.

The I/O throughput of Linux Kernel 2.6.10 (SLES10 and later and RHEL5 and later) can be increased by adjusting the default I/O settings.

Note: SLES 10 is not supported in StorNext 5.

Beginning with the 2.6 kernel, the Linux I/O scheduler can be changed to control how the kernel does reads and writes. There are four types of I/O scheduler available in most versions of Linux kernel 2.6.10 and higher:

- The completely fair queuing scheduler (CFQ)
- The no operation scheduler (NOOP)
- The deadline scheduler (DEADLINE)
- The anticipatory scheduler (ANTICIPATORY)

Note: ANTICIPATORY is not present in SLES 11 SP2.

The default scheduler in most distributions is the completely fair queuing (CFQ). Experimentation displays that the deadline scheduler provides the best improvement.

Increasing the number of outstanding requests has been shown to provide a performance benefit:

```
nr_requests=4096
```

In addition, there are three Linux kernel parameters that can be tuned for increased performance:

1. The **minimal preemption qranularity** variable for CPU bound tasks.

   ```
   kernel.sched_min_granularity_ns = 10ms

   echo 10000000 > /proc/sys/kernel/sched_min_granularity_ns
   ```

2. The **wake-up preemption qranularity** variable. Increasing this variable reduces wake-up preemption, reducing disturbance of computer bound tasks. Lowering it improves wake-up latency and throughput for latency of critical tasks.

   ```
   kernel.sched_wakeup_granularity_ns = 15ms

   echo 15000000 > /proc/sys/kernel/sched_wakeup_granularity_ns
   ```

3. The `vm.dirty_background_ratio` variable contains 10, which is a percentage of total system memory, the number of pages at which the `pbflush` background writeback daemon will start writing out dirty data. However, for fast RAID based disk system, this may cause large flushes of dirty memory pages. Increasing this value will result in less frequent flushes.

```
vm.dirty_ratio = 40% RAM

sysctl vm.dirty_background_ratio = 40
```

For additional details, see the command `deviceparams(4)` in the StorNext Man Pages Reference Guide and also see StorNext Product Bulletin 50.

File Size Mix and Application I/O Characteristics

It is always valuable to understand the file size mix of the target dataset as well as the application I/O characteristics. This includes the number of concurrent streams, proportion of read versus write streams, I/O size, sequential versus random, Network File System (NFS) or Common Internet File System (CIFS) access, and so on.

For example, if the dataset is dominated by small or large files, various settings can be optimized for the target size range.

Similarly, it might be beneficial to optimize for particular application I/O characteristics. For example, to optimize for sequential 1MB I/O size it would be beneficial to configure a stripe group with four 4+1 RAID 5 LUNs with 256K stripe size.

However, optimizing for random I/O performance can incur a performance trade-off with sequential I/O.

Furthermore, NFS and CIFS access have special requirements to consider as described in the section, Direct Memory Access (DMA) I/O Transfer below.

Direct Memory Access (DMA) I/O Transfer

To achieve the highest possible large sequential I/O transfer throughput, SNFS provides DMA-based I/O. To utilize DMA I/O, the application must issue its reads and writes of sufficient size and alignment. This is called well-formed I/O. See the **mountcommand** settings **auto_dma_read_length** and **auto_dma_write_length**, described in the The Metadata Controller System on page 24.

Buffer Cache

Reads and writes that aren't well-formed utilize the SNFS buffer cache. This also includes NFS or CIFS-based traffic because the NFS and CIFS daemons defeat well-formed I/Os issued by the application.

There are several configuration parameters that affect buffer cache performance. The most critical is the RAID cache configuration because buffered I/O is usually smaller than the RAID stripe size, and therefore incurs a read/modify/write penalty. It might also be possible to match the RAID stripe size to the buffer cache

I/O size. However, it is typically most important to optimize the RAID cache configuration settings described earlier.

It is usually best to configure the RAID stripe size no greater than 256K for optimal small file buffer cache performance.

For more buffer cache configuration settings, see The Metadata Controller System on page 24.

Network File System (NFS) in StorNext

StorNext supports NFS version 3 (NFSv3) and NFS version 4 (NFSv4) with some limitations. For additional information, see **Network File System (NFS) Support in StorNext**, in the *StorNext Compatibility Guide* available online at http://www.quantum.com/snsdocs, and also the Appliance Controller Compatibility Guide.

NFS / CIFS

It is best to isolate NFS and/or CIFS traffic off of the metadata network to eliminate contention that will impact performance. On NFS clients, use the `rsize=1048576` and `wsize=1048576` mount options. When possible, it is also best to utilize TCP Offload capabilities as well as jumbo frames.

Note: Jumbo frames should only be configured when all of the relevant networking components in the environment support them.

Note: When Jumbo frames are used, the MTU on the Ethernet interface should be configured to an appropriate size. Typically, the correct value is 9000, but may vary depending on your networking equipment. Refer to the documentation for your network adapter.

It is best practice to have clients directly attached to the same network switch as the NFS or CIFS server. Any routing required for NFS or CIFS traffic incurs additional latency that impacts performance.

It is critical to make sure the **speed/duplex** settings are correct, because this severely impacts performance. Most of the time **auto-negotiation** is the correct setting for the ethernet interface used for the NFS or CIFS traffic.

Whether **auto-negotiation** is the correct setting depends on the ethernet switch capabilities that the ethernet interface connects to. Some managed switches cannot negotiate the **auto-negotiation** capability with a host ethernet interface and instead allow setting **speed/duplex** (for example `1000Mb/full`,) which disables **auto-negotiation** and requires the host to be set exactly the same. However, if the settings do not match between switch and host, it severely impacts performance. For example, if the switch is set to `auto-negotiation` but the host is set to `1000Mb/full`, you will observe a high error rate along with extremely poor performance. On Linux, the `ethtool` tool can be very useful to investigate and adjust **speed/duplex** settings.

If performance requirements cannot be achieved with NFS or CIFS, consider using a StorNext LAN client or fibre-channel attached client.

It can be useful to use a tool such as **netperf** to help verify network performance characteristics.

The NFS subtree_check Option

Although supported in previous StorNext releases, the `subtree_check` option (which controls NFS checks on a file handle being within an exported subdirectory of a file system) is no longer supported.

Reverse Path Lookup (RPL)

All file systems have the Reverse Path Lookup (RPL) feature enabled. Any file system that did not have RPL enabled prior to upgrading will have it enabled as part of the metadata conversion process which runs the first time StorNext 5 is used. StorNext uses RPL in the following ways:

- Replication uses RPL when generating reports.
- Storage Manager uses RPL as a way to quickly generate full path names.
- The **Directory Quotas** feature requires RPL.
- The command **`cvadmin repof`** uses RPL to display the full pathname of open files.
- Some StorNext log messages use RPL to display full path names.

In previous releases, enabling RPL on file systems created with pre-4.0 versions of StorNext could have negative side effects:

- Extensive downtime to populate existing inodes with RPL information.
- Increased metadata space usage.
- Decreased performance for certain inode-related operations.

However, the metadata conversion process allows RPL to always be stored efficiently and these negative side effects do not apply.

SNFS and Virus Checking

Virus-checking software can severely degrade the performance of any file system, including SNFS. However, shared file systems such as SNFS are particularly susceptible because virus scanning can be configured on all clients, causing a multiplier effect.

For optimal performance, Quantum recommends turning off virus checking on the SNFS file system.

The Metadata Network

As with any client/server protocol, SNFS performance is subject to the limitations of the underlying network. Therefore, it is recommended that you use a dedicated Metadata Network to avoid contention with other network traffic. Neither TCP offload nor jumbo frames are required.

It is best practice to have all SNFS clients directly attached to the same network switch as the MDC systems. Any routing required for metadata traffic will incur additional latency that impacts performance.

It can be useful to use a tool like **netperf** to help verify the Metadata Network performance characteristics. For example, if **netperf -t TCP_RR -H <host>** reports less than 4,000 transactions per second capacity, a performance penalty may be incurred. You can also use the **netstat** tool to identify tcp retransmissions impacting performance. The cvadmin “latency-test” tool is also useful for measuring network latency.

Note the following configuration requirements for the metadata network:

- In cases where gigabit networking hardware is used and maximum StorNext performance is required, a separate, dedicated switched Ethernet LAN is recommended for the StorNext metadata network. If maximum StorNext performance is not required, shared gigabit networking is acceptable.

The Metadata Controller System

The CPU power and memory capacity of the MDC System are important performance factors, as well as the number of file systems hosted per system. In order to ensure fast response time it is necessary to use dedicated systems, limit the number of file systems hosted per system (maximum 8), and have an adequate CPU and memory. Refer to the *StorNext User’s Guide* for limits on the number of files per file system and per database instance.

Some metadata operations such as file creation can be CPU intensive, and benefit from increased CPU power.

Other operations can benefit greatly from increased memory, such as directory traversal. SNFS provides two config file settings that can be used to realize performance gains from increased memory:

- `BufferCacheSize`
- `InodeCacheSize`

However, it is critical that the MDC system have enough physical memory available to ensure that the FSM process doesn’t get swapped out. Otherwise, severe performance degradation and system instability can result.

The operating system on the metadata controller must always be run in U.S. English. On Windows systems, this is done by setting the system locale to U.S. English.

Caution: As the File System Manager (FSM) supports over 1000 clients (with more than 1000 file requests per client), the resource limits of your MDC may be exhausted with additional load from other processes. Exceeding the file descriptor limit will cause errors to your system. Quantum recommends you **not** run additional applications on the MDC.

FSM Configuration File Settings

The following FSM configuration file settings are explained in greater detail in the `snfs.cfgx` file and **`snfs_config(5)`** man pages, which are available in the *StorNext Man Pages Reference Guide* posted here http://www.quantum.com/snsdocs.

Please refer there for setting details and an example file. For a sample FSM configuration file, see Example FSM Configuration File on page 64.

Stripe Groups

Splitting apart data, metadata, and journal into separate stripe groups is usually the most important performance tactic. The **create**, **remove**, and **allocate** (for example, write) operations are very sensitive to I/O latency of the journal stripe group. However, if **create**, **remove**, and **allocate** performance aren't critical, it is okay to share a stripe group for both metadata and journal, but be sure to set the exclusive property on the stripe group so it does not get allocated for data as well.

Note: It is recommended that you have only a single metadata stripe group. For increased performance, use multiple LUNs (2 or 4) for the stripe group.

RAID 1 mirroring is optimal for metadata and journal storage. Utilizing the write-back caching feature of the RAID system (as described previously) is critical to optimizing performance of the journal and metadata stripe groups. Quantum recommends mapping no more than one LUN per RAID 1 set.

Example (Linux)

```
<stripeGroup index="0" name="MetaFiles" status="up" stripeBreadth="262144"
read="true" write="true" metadata="true" journal="false" userdata="false"
realTimeIOs="200" realTimeIOsReserve="1" realTimeMB="200" realTimeMBReserve="1"
realTimeTokenTimeout="0" multipathMethod="rotate">
   <disk index="0" diskLabel="CvfsDisk0" diskType="MetaDrive"/>
</stripeGroup>
<stripeGroup index="1" name="JournFiles" status="up" stripeBreadth="262144"
read="true" write="true" metadata="false" journal="true" userdata="false"
realTimeIOs="0" realTimeIOsReserve="0" realTimeMB="0" realTimeMBReserve="0"
realTimeTokenTimeout="0" multipathMethod="rotate">
   <disk index="0" diskLabel="CvfsDisk1" diskType="JournalDrive"/>
</stripeGroup>
<stripeGroup index="4" name="RegularFiles" status="up" stripeBreadth="262144"
read="true" write="true" metadata="false" journal="false" userdata="true"
realTimeIOs="0" realTimeIOsReserve="0" realTimeMB="0" realTimeMBReserve="0"
realTimeTokenTimeout="0" multipathMethod="rotate">
   <disk index="0" diskLabel="CvfsDisk14" diskType="DataDrive"/>
   <disk index="1" diskLabel="CvfsDisk15" diskType="DataDrive"/>
   <disk index="2" diskLabel="CvfsDisk16" diskType="DataDrive"/>
   <disk index="3" diskLabel="CvfsDisk17" diskType="DataDrive"/>
</stripeGroup>
```

Example (Windows)

```
[StripeGroup MetaFiles]
Status Up
StripeBreadth 256K
Metadata Yes
Journal No
Exclusive Yes
Read Enabled
Write Enabled
Rtmb 200
Rtios 200
RtmbReserve 1
RtiosReserve 1
RtTokenTimeout 0
MultiPathMethod Rotate
Node CvfsDisk0 0

[StripeGroup JournFiles]
Status Up
StripeBreadth 256K
Metadata No
Journal Yes
Exclusive Yes
Read Enabled
Write Enabled
Rtmb 0
Rtios 0
RtmbReserve 0
RtiosReserve 0
RtTokenTimeout 0
MultiPathMethod Rotate
Node CvfsDisk1 0

[StripeGroup RegularFiles]
Status Up
StripeBreadth 256K
Metadata No
```

```
Journal No
Exclusive No
Read Enabled
Write Enabled
Rtmb 0
Rtios 0
RtmbReserve 0
RtiosReserve 0
RtTokenTimeout 0
MultiPathMethod Rotate
Node CvfsDisk14 0
Node CvfsDisk15 1
Node CvfsDisk16 2
Node CvfsDisk17 3
```

Affinities

Affinities are another stripe group feature that can be very beneficial. Affinities can direct file allocation to appropriate stripe groups according to performance requirements. For example, stripe groups can be set up with unique hardware characteristics such as fast disk versus slow disk, or wide stripe versus narrow stripe. Affinities can then be employed to steer files to the appropriate stripe group.

For optimal performance, files that are accessed using large DMA-based I/O could be steered to wide-stripe stripe groups. Less performance-critical files could be steered to slow disk stripe groups. Small files could be steered clear of large files, or to narrow-stripe stripe groups.

Example (Linux)

```
<stripeGroup index="3" name="AudioFiles" status="up" stripeBreadth="1048576"
read="true" write="true" metadata="false" journal="false" userdata="true"
realTimeIOs="0" realTimeIOsReserve="0" realTimeMB="0" realTimeMBReserve="0"
realTimeTokenTimeout="0" multipathMethod="rotate">
   <affinities exclusive="true">
      <affinity>Audio</affinity>
   </affinities>
   <disk index="0" diskLabel="CvfsDisk10" diskType="AudioDrive"/>
   <disk index="1" diskLabel="CvfsDisk11" diskType="AudioDrive"/>
   <disk index="2" diskLabel="CvfsDisk12" diskType="AudioDrive"/>
   <disk index="3" diskLabel="CvfsDisk13" diskType="AudioDrive"/>
</stripeGroup>
```

Example (Windows)

```
[StripeGroup AudioFiles]
Status Up
StripeBreadth 1M
Metadata No
Journal No
Exclusive Yes
Read Enabled
Write Enabled
Rtmb 0
Rtios 0
RtmbReserve 0
RtiosReserve 0
RtTokenTimeout 0
MultiPathMethod Rotate
Node CvfsDisk10 0
Node CvfsDisk11 1
Node CvfsDisk12 2
Node CvfsDisk13 3
Affinity Audio
```

Note: Affinity names cannot be longer than eight characters.

StripeBreadth

This setting should match the RAID stripe size or be a multiple of the RAID stripe size. Matching the RAID stripe size is usually the most optimal setting. However, depending on the RAID performance characteristics and application I/O size, it might be beneficial to use a multiple or integer fraction of the RAID stripe size. For example, if the RAID stripe size is 256K, the stripe group contains 4 LUNs, and the application to be optimized uses DMA I/O with 8MB block size, a `StripeBreadth` setting of 2MB might be optimal. In this example the 8MB application I/O is issued as 4 concurrent 2MB I/Os to the RAID. This concurrency can provide up to a 4X performance increase. This StripeBreadth typically requires some experimentation to determine the RAID characteristics. The `lmdd` utility can be very helpful. Note that this setting is not adjustable after initial file system creation.

Optimal range for the `StripeBreadth` setting is 128K to multiple megabytes, but this varies widely.

Note: This setting cannot be changed after being put into production, so its important to choose the setting carefully during initial configuration.

Example (Linux)

```
<stripeGroup index="2" name="VideoFiles" status="up" stripeBreadth="4194304"
read="true" write="true" metadata="false" journal="false" userdata="true"
realTimeIOs="0" realTimeIOsReserve="0" realTimeMB="0" realTimeMBReserve="0"
```

```
realTimeTokenTimeout="0" multipathMethod="rotate">
   <affinities exclusive="true">
      <affinity>Video</affinity>
   </affinities>
   <disk index="0" diskLabel="CvfsDisk2" diskType="VideoDrive"/>
   <disk index="1" diskLabel="CvfsDisk3" diskType="VideoDrive"/>
   <disk index="2" diskLabel="CvfsDisk4" diskType="VideoDrive"/>
   <disk index="3" diskLabel="CvfsDisk5" diskType="VideoDrive"/>
   <disk index="4" diskLabel="CvfsDisk6" diskType="VideoDrive"/>
   <disk index="5" diskLabel="CvfsDisk7" diskType="VideoDrive"/>
   <disk index="6" diskLabel="CvfsDisk8" diskType="VideoDrive"/>
   <disk index="7" diskLabel="CvfsDisk9" diskType="VideoDrive"/>
</stripeGroup>
```

Example (Windows)

```
[StripeGroup VideoFiles]
Status Up
StripeBreadth 4M
Metadata No
Journal No
Exclusive Yes
Read Enabled
Write Enabled
Rtmb 0
Rtios 0
RtmbReserve 0
RtiosReserve 0
RtTokenTimeout 0
MultiPathMethod Rotate
Node CvfsDisk2 0
Node CvfsDisk3 1
Node CvfsDisk4 2
Node CvfsDisk5 3
Node CvfsDisk6 4
Node CvfsDisk7 5
Node CvfsDisk8 6
Node CvfsDisk9 7
Affinity Video
```

BufferCacheSize

Increasing this value can reduce latency of any metadata operation by performing a **hot cache** access to directory blocks, inode information, and other metadata info. This is about 10 to 1000 times faster than I/O. It is especially important to increase this setting if metadata I/O latency is high, (for example, more than 2ms average latency). Quantum recommends sizing this according to how much memory is available; more is better. Optimal settings for `BufferCacheSize` range from 32MB to 8GB for a new file system and can be increased up to 500GB as a file system grows. A higher setting is more effective if the CPU is not heavily loaded.

When the value of `BufferCacheSize` is greater than 1GB, SNFS uses compressed buffers to maximize the amount of cached metadata. The effective size of the cache is as follows:

If the `BufferCacheSize` is less than or equal to 1GB, then:

Effective Cache Size = `BufferCacheSize`

If the `BufferCacheSize` is greater than 1GB, then:

Effective Cache Size = (`BufferCacheSize` - 512MB) * 2.5

The value 2.5 in the above formula represents a typical level of compression. This factor may be somewhat lower or higher, depending on the complexity of the file system metadata.

Note: Configuring a large value of `BufferCacheSize` will increase the memory footprint of the FSM process. If this process crashes, a core file will be generated that will consume disk space proportional to its size.

Example (Linux)

```
<bufferCacheSize>268435456</bufferCacheSize>
```

Example (Windows)

```
BufferCacheSize 256MB
```

In StorNext, the default value for the `BufferCacheSize` parameter in the file system configuration file changed from **32 MB** to **256 MB**. While uncommon, if a file system configuration file is missing this parameter, the new value will be in effect. This may improve performance; however, the FSM process will use more memory than it did with previous releases (up to 400 MB).

To avoid the increased memory utilization, the `BufferCacheSize` parameter may be added to the file system configuration file with the old default values.

Example (Linux)

```
<bufferCacheSize>33554432</bufferCacheSize>
```

Example (Windows)

```
BufferCacheSize 32M
```

InodeCacheSize

This setting consumes about 1400 bytes of memory times the number specified. Increasing this value can reduce latency of any metadata operation by performing a hot cache access to inode information instead of an I/O to get inode info from disk, about 100 to 1000 times faster. It is especially important to increase this setting if metadata I/O latency is high, (for example, more than 2ms average latency). You should try to size this according to the sum number of working set files for all clients. Optimal settings for `InodeCacheSize` range from 16K to 128K for a new file system and can be increased to 256K or 512K as a file system grows. A higher setting is more effective if the CPU is not heavily loaded. For best performance, the `InodeCacheSize` should be at least 1024 times the number of megabytes allocated to the journal. For example, for a 64MB journal, the `inodeCacheSize` should be at least 64K.

Example (Linux)

```
<inodeCacheSize>131072</inodeCacheSize>
```

Example (Windows)

```
InodeCacheSize 128K
```

In StorNext, the default value for the `InodeCacheSize` parameter in the file system configuration file changed from **32768** to **131072**. While uncommon, if a file system configuration file is missing this parameter, the new value will be in effect. This may improve performance; however, the FSM process will use more memory than it did with previous releases (up to 400 MB).

To avoid the increased memory utilization, the `InodeCacheSize` parameter may be added to the file system configuration file with the old default values.

Example (Linux)

```
<inodeCacheSize>32768</inodeCacheSize>
```

Example (Windows)

```
InodeCacheSize 32K
```

FsBlockSize

Beginning with StorNext 5, all SNFS file systems use a File System Block Size of 4KB. This is the optimal value and is no longer tunable. Any file systems created with pre-5 versions of StorNext having larger File System Block Sizes will be automatically converted to use 4KB the first time the file system is started with StorNext 5.

JournalSize

Beginning with StorNext 5, the recommended setting for `JournalSize` is 64Mbytes.

Increasing the `JournalSize` beyond 64Mbytes may be beneficial for workloads where many large size directories are being created, or removed at the same time. For example, workloads dealing with 100 thousand files in a directory and several directories at once will see improved throughput with a larger journal.

The downside of a larger journal size is potentially longer FSM startup and failover times.

Using a value less than 64Mbytes may improve failover time but reduce file system performance. Values less than 16Mbytes are not recommended.

Note: Journal replay has been optimized with StorNext 5 so a 64Mbytes journal will often replay significantly faster with StorNext 5 than a 16Mbytes journal did with prior releases.

A file system created with a pre-5 version of StorNext may have been configured with a small `JournalSize`. This is true for file systems created on Windows MDCs where the old default size of the journal was 4Mbytes. Journals of this size will continue to function with StorNext 5, but will experience a performance benefit if the size is increased to 64Mbytes. This can be adjusted using the cvupdatefs utility. For more information, see the command **`cvupdatefs`** in the *StorNext MAN Pages Reference Guide*.

If a file system previously had been configured with a `JournalSize` larger than 64Mbytes, there is no reason to reduce it to 64Mbytes when upgrading to StorNext 5.

Example (Linux)

```
<config configVersion="0" name="example" fsBlockSize="4096"
journalSize="67108864">
```

Example (Windows)

```
JournalSize 64M
```

SNFS Tools

The **snfsdefrag** tool is very useful to identify and correct file extent fragmentation. Reducing extent fragmentation can be very beneficial for performance. You can use this utility to determine whether files are fragmented, and if so, fix them.

Note: Beginning with StorNext 6, use the **`sgoffload`** command instead of the **`snfsdefrag`** command. The **`sgoffload`** command moves extents belonging to files that are currently in use (open). The **`sgoffload`** command also informs the client to suspend I/O for a time, moves the data, then informs the client to refresh the location of the data and resume I/O.

Qustats

The **qustats** are measuring overall metadata statistics, physical I/O, VOP statistics and client specific VOP statistics.

The overall metadata statistics include journal and cache information. All of these can be affected by changing the configuration parameters for a file system. Examples are increasing the journal size, and increasing cache sizes.

The physical I/O statistics show number and speed of disk I/Os. Poor numbers can indicate hardware problems or over-subscribed disks.

The VOP statistics show what requests SNFS clients making to the MDCs, which can show where workflow changes may improve performance.

The Client specific VOP statistics can show which clients are generating the VOP requests. Below are examples of **qustat** operations.

- Print the current stats to stdout:

```
# qustat -g <file_system_name>
```

- Print a description of a particular stat:

```
# qustat -g <file_system_name> -D "<stat_name>"
```

```
# qustat -g kernel -m client -D "<stat_name>"
```

Note: Use * for stat name to print descriptions on all stats.

For additional information on **qustat**, see the **qustat** man page.

There are a large number of **qustat** counters available in the output file, most are debugging counters and are not useful in measuring or tuning the file system. The items in the table below have been identified as the most interesting counters. All other counters can be ignored.

Table 1: Qustat Counters

Name	Type	Description
`Cache Stats`	`Buffer Cache` (`Buf Hits` and `Buf Misses`)	If hit rate is low, FSM `BufferCacheSize` may need to be increased. The number of hits and misses are reported.
	`Inode Cache` (`ICa Hits` and `ICa Misses`)	If hit rate is low, `InodeCacheSize` may need to be increased. The number of hits and misses are reported.
`PhysIO Stats`	`Read/Write`	Physical metadata I/O statistics
	`max`	The maximum time to service a metadata read request.
	`average`	The average time to service a metadata read request.

Name	Type	Description
`VOP Stats`		File and Directory operations
	`Create and Remove`	File create and remove operations
	`Cnt`	The count of operations in an hour
	`Mkdir and Rmdir`	Directory create and remove operations
	`Cnt`	The count of operations in an hour.
	`Rename`	File and directory rename/mv operations
	`Cnt`	The count of operations in an hour
	`Open and Close`	File open and close operations
	`Cnt`	The count of operations in an hour
	`ReadDir`	The application `readdir()` operation, `ls` or `dir` operations. These are heavyweight and should be minimized wherever possible
	`Cnt`	The count of operations in an hour
	`DirAttr`	Gather attributes for a directory listing (Windows only)
	Cnt	The count of operations in an hour
	Get Attr and Set Attr	Attribute updates and queries, touch, stat, implicit stat calls
	Cnt	The count of operations in an hour
`Client VOP Stats`	n/a	Per client VOP stats are available to determine which client may be putting a load on the MDC

The **`qustat`** command also supports the client module. The client is the StorNext file system driver that runs in the Linux kernel. Unlike the **`cvdb`** PERF traces, which track individual I/O operations, the qustat statisics group like operations into buckets and track minimum, maximum and average duration times. In the following output, one can see that the statistics show global counters for all file systems as well as counters for individual file system. In this example, only a single file system is shown.

This shows qustats from a distributed LAN client:

```
[root@snt931782-client ~]# /usr/cvfs/bin/qustat -m client

# QuStat Rev 6.0.5
# Host snt931782-client

# Module client

# Group kernel

# Recorded time_t=1508371304 2017-10-18 19:01:44 CDT
```

The following tables for VFSOPS and VNOPS table keep track of meta data operations. These statistics are also available from the FSM.

```
# Table 1: Global.VFSOPS

# Last Reset: Secs=10301 time_t=1508361003 2017-10-18 16:10:03 CDT

# NAME TYP COUNT MIN MAX TOT/LVL AVG

Mount TIM 1 67758 67758 67758 67758
```

```
# Table 2: Global.VNOPS

# Last Reset: Secs=10301 time_t=1508361003 2017-10-18 16:10:03 CDT

# NAME TYP COUNT MIN MAX TOT/LVL AVG

Lookup TIM 661 92 4898 373940 566

Lookup Misses TIM 7905 62 44199 1374966 174

Create TIM 10516 129 23416 4075220 388

Open TIM 327417 1 3998 2178607 7

Close TIM 336102 2 31739 2101414 6

Close Last Ref TIM 1997 2 31744 772709 387

Delete TIM 1819 96 3389 309565 170

Truncate TIM 264565 1 14961 71115308 269

Read Calls TIM 20367864 7 45887 528922580 26

Read Bytes SUM 20367864 0 2097152 77962752530 3828

Write Calls TIM 1775285 30 17892702 158410978 89

Write Bytes SUM 1775285 1 1048576 23395107202 13178
```

```
# Table 3: Global.BCACHE
# Last Reset: Secs=10301 time_t=1508361003 2017-10-18 16:10:03 CDT
# NAME TYP COUNT MIN MAX TOT/LVL AVG
Bwait TIM 61763 6 1407592 14869715 241
buf thread TIM 365524 22 26084 45011889 123
fluser ran TIM 34204 1 12493 1594197 47
fluser ndone LVL 33916 1 1000 3 0
Wflush Rv CNT 108 1 1 108 1
Wflush Hi CNT 12 1 1 12 1
BC hits CNT 22579903 1 1 22579903 1
BC miss CNT 1937475 1 1 1937475 1
BC dirty LVL 697770 1 2731 2 0
File dirty LVL 697770 1 4096 4095 0
MI dirty LVL 697770 0 4095 4094 0
Rsvd grntd LVL 233115 33554432 295239680 295239680 0
Rsvd reqs LVL 454646 4096 1048576 1048576 0
Synca bufs LVL 6990 1 206 2 0
Synca cvps LVL 1513 1 2711 1 0
Synca clos LVL 78 1 1434 1 0
Synca runs TIM 5132 2 1051476 33998868 6625
Synca skipc LVL 192 0 3 1 0
nodechg que LVL 2 1 2 1 0
fsminput msgs TIM 4061 7 851 132216 33
```

```
# Table 4: Global.Proxy
# Last Reset: Secs=10301 time_t=1508361003 2017-10-18 16:10:03 CDT
# NAME TYP COUNT MIN MAX TOT/LVL AVG
Proxy Rd Bytes SUM 162439 4096 65536 9862545408 60715
Proxy Wt Bytes SUM 180754 4096 65536 10765529088 59559
```

```
# Table 5: fs.snfs1.vnops
# Last Reset: Secs=10300 time_t=1508361004 2017-10-18 16:10:04 CDT
# NAME TYP COUNT MIN MAX TOT/LVL AVG
Lookup TIM 661 93 4898 373991 566
Lookup Misses TIM 7905 62 44199 1375575 174
Create TIM 10516 129 23416 4075946 388
Open TIM 327417 1 3998 2211737 7
Close TIM 336102 2 31740 2117677 6
Close Last Ref TIM 1997 2 31744 772799 387
Delete TIM 1819 96 3389 309701 170
Truncate TIM 264565 1 14961 71129062 269
Read Calls TIM 20367864 7 45887 528168288 26
Read Bytes SUM 20367864 0 2097152 77962752530 3828
Write Calls TIM 1775285 30 17892702 158340561 89
Write Bytes SUM 1775285 1 1048576 23395107202 13178
```

The remaining tables show I/O statistics.

- Table 6 shows proxy activity to the snfs1 proxy server.
- Table 7 has no statistics because there is no direct attachment on this client.
- Table 8 shows the I/O requests for sg1. Multiple stripe groups would be represented by multiple tables.

```
# Table 6: proxy.fs.snfs1.server.10.65.191.209.client.10.65.191.198
# Last Reset: Secs=9919 time_t=1508361385 2017-10-18 16:16:25 CDT
# NAME TYP COUNT MIN MAX TOT/LVL AVG
Proxy Rd Bytes SUM 158880 4096 65536 9799798784 61681
Proxy Wt Bytes SUM 170305 4096 65536 10493296640 61615
Proxy Path Use LVL 329185 4096 268435456 65536 0
Proxy Read TO LVL 1 15 15 15 0
Proxy Write TO LVL 1 30 30 30 0
Proxy Read ET LVL 158880 233 41955 636 0
Proxy Write ET LVL 170305 207 1428277 1360 0
```

```
# Table 7: fs.snfs1.sg.sg1.io.san
# Last Reset: Secs=10300 time_t=1508361004 2017-10-18 16:10:04 CDT
# NAME TYP COUNT MIN MAX TOT/LVL AVG
```

```
# Table 8: fs.snfs1.sg.sg1.io.lan
# Last Reset: Secs=10300 time_t=1508361004 2017-10-18 16:10:04 CDT
# NAME TYP COUNT MIN MAX TOT/LVL AVG
Rd Prep Dev TIM 153051 14 2113 7215424 47
Rd children SUM 153051 1 2 162439 1
Rd Time Dev TIM 162439 252 41994 109354167 673
Rd Bytes Dev SUM 162439 4096 65536 9862545408 60715
Wr Prep Dev TIM 170365 12 26049 22994916 135
WR children SUM 170365 1 5 180751 1
Wrt Time Dev TIM 180754 222 2132752 98767663114 546420
Wrt Bytes Dev SUM 180747 4096 65536 10765398016 59561
```

These client statistics are from a gateway server.

- Table 6 with the .san suffix shows I/O initiated by this client.
- Table 7 with the .lan suffix shows the I/O done on behalf of the LAN client.
- Table 8 with the .gw suffix shows gateway server statistics. If a file system has multiple stripe groups, each stripe group has its own table for .san and .lan.

```
[root@snt931782-mdc lcostello]# qustat -m client
# QuStat Rev 6.0.5
# Host snt931782-mdc
# Module client
# Group kernel
# Recorded time_t=1508371406 2017-10-18 19:03:26 CDT
# Table 1: Global.VFSOPS
# Last Reset: Secs=10883 time_t=1508360523 2017-10-18 16:02:03 CDT
# NAME TYP COUNT MIN MAX TOT/LVL AVG
```

```
Mount TIM 2 2654315 2654315 3182277 1591139
```

```
# Table 2: Global.VNOPS
# Last Reset: Secs=10883 time_t=1508360523 2017-10-18 16:02:03 CDT
# NAME TYP COUNT MIN MAX TOT/LVL AVG
Lookup TIM 1 0 0 701 701
Lookup Misses TIM 559 37 2107 59086 106
Create TIM 7 1107 1107 5119 731
Open TIM 4 0 170 471 118
Close TIM 4 1 1 4 1
Close Last Ref TIM 8 0 0 7 1
Truncate TIM 1 1030 1030 1030 1030
Read Calls TIM 20480101 0 7748 16023087 1
Read Bytes SUM 20480101 0 1048576 10590617600 517
Write Calls TIM 10100 111 1421872 22913372 2269
Write Bytes SUM 10100 1048576 1048576 10590617600 1048576
```

```
# Table 3: Global.BCACHE
# Last Reset: Secs=10883 time_t=1508360523 2017-10-18 16:02:03 CDT
# NAME TYP COUNT MIN MAX TOT/LVL AVG
Bwait TIM 17004 2 1408738 19988474 1176
buf thread TIM 321598 1 2259516 193362992 601
fluser ran TIM 8430 0 594 74473 9
fluser ndone LVL 8426 1 2600 24 0
Wflush Rv CNT 23 1 1 23 1
Wflush Hi CNT 8 1 1 8 1
BC hits CNT 20483196 1 1 20483196 1
BC miss CNT 321600 1 1 321600 1
BC dirty LVL 161600 1 2732 15 0
File dirty LVL 161600 1 4096 4095 0
```

```
MI dirty LVL 161600 0 4095 4094 0
Rsvd grntd LVL 29 33554432 295239680 295239680 0
Rsvd fail CNT 1 1 1 1 1
Rsvd reqs LVL 4366 1048576 1048576 1048576 0
Synca bufs LVL 2 15 1248 15 0
Synca cvps LVL 19 1 1 1 0
Synca clos LVL 4 1 1 1 0
Synca runs TIM 5356 2 1158 32696 6
Synca skipc LVL 1 0 0 0 0
fsminput msgs TIM 4753 5 651 108166 23
```

```
# Table 4: Global.Proxy
# Last Reset: Secs=10883 time_t=1508360523 2017-10-18 16:02:03 CDT
# NAME TYP COUNT MIN MAX TOT/LVL AVG
Proxy Rd Bytes SUM 162439 4096 65536 9862545408 60715
Proxy Wt Bytes SUM 180754 4096 65536 10765529088 59559
```

```
# Table 5: fs.snfs1.vnops
# Last Reset: Secs=10024 time_t=1508361382 2017-10-18 16:16:22 CDT
# NAME TYP COUNT MIN MAX TOT/LVL AVG
Lookup Misses TIM 548 37 2107 57945 106
Create TIM 1 1107 1107 1107 1107
Open TIM 4 0 171 473 118
Close TIM 4 1 1 4 1
Truncate TIM 1 1030 1030 1030 1030
Read Calls TIM 20480101 0 7748 15287864 1
Read Bytes SUM 20480101 0 1048576 10590617600 517
Write Calls TIM 10100 111 1421872 22912369 2269
Write Bytes SUM 10100 1048576 1048576 10590617600 1048576
```

```
# Table 6: fs.snfs1.sg.sg1.io.san
# Last Reset: Secs=10021 time_t=1508361385 2017-10-18 16:16:25 CDT
# NAME TYP COUNT MIN MAX TOT/LVL AVG
Rd Prep Dev TIM 160002 0 363 249350 2
Rd children SUM 160002 1 1 160002 1
Rd Time Dev TIM 160002 198 10632 94776162 592
Rd Bytes Dev SUM 160002 4096 65536 10485768192 65535
Wr Prep Dev TIM 161579 0 225473 4458791 28
WR children SUM 161595 1 1 161595 1
Wrt Time Dev TIM 161597 461 2362857 28655861015 177329
Wrt Bytes Dev SUM 161537 65536 65536 10585047040 65527
```

```
# Table 7: fs.snfs1.sg.sg1.io.gw
# Last Reset: Secs=10021 time_t=1508361385 2017-10-18 16:16:25 CDT
# NAME TYP COUNT MIN MAX TOT/LVL AVG
Rd Time Dev TIM 158880 175 9487 62297771 392
Rd Bytes Dev SUM 158863 4096 65536 9799311360 61684
Wrt Time Dev TIM 170302 178 1417335 493715525 2899
Wrt Bytes Dev SUM 170298 4096 65536 10492895232 61615
```

```
# Table 8: proxy.fs.snfs1.server.10.65.191.209.client.10.65.191.198
# Last Reset: Secs=10021 time_t=1508361385 2017-10-18 16:16:25 CDT
# NAME TYP COUNT MIN MAX TOT/LVL AVG
Proxy Rd Bytes SUM 158880 4096 65536 9799798784 61681
Proxy Wt Bytes SUM 170305 4096 65536 10493296640 61615
Proxy SrvBufWt TIM 42957 4 1407875 15298331 356
Proxy Srv Ltcy TIM 329167 4 1407925 29497078 90
```

SNFS supports the Windows **Perfmon** utility (see Windows Performance Monitor Counters on page 61). This provides many useful statistics counters for the SNFS client component. Run **rmperfreg.exe** and **instperfreg.exe** to set up the required registry settings. Next, call **cvdb -P**. After these steps, the SNFS counters should be visible to the **WindowsPerfmon** utility. If not, check the Windows Application Event log for errors.

The **cvcp** utility is a higher performance alternative to commands such as **cp** and **tar**. The **cvcp** utility achieves high performance by using threads, large I/O buffers, preallocation, stripe alignment, DMA I/O transfer, and Bulk Create. Also, the **cvcp** utility uses the SNFS External API for preallocation and stripe alignment. In the directory-to-directory copy mode (for example, **cvcp source_dir destination_dir**,) **cvcp** conditionally uses the **Bulk Create** API to provide a dramatic small file copy performance boost. However, it will not use **Bulk Create** in some scenarios, such as non-root invocation, managed file systems, quotas, or Windows security. When **Bulk Create** is utilized, it significantly boosts performance by reducing the number of metadata operations issued. For example, up to 20 files can be created all with a single metadata operation. For more information, see the **cvcp** man page.

The **cvmkfile** utility provides a command line tool to utilize valuable SNFS performance features. These features include preallocation, stripe alignment, and affinities. See the **cvmkfile** man page.

The **Lmdd** utility is very useful to measure raw LUN performance as well as varied I/O transfer sizes. It is part of the **lmbench** package and is available from http://sourceforge.net.

The **cvdbset** utility has a special “Perf” trace flag that is very useful to analyze I/O performance. For example: `cvdbset perf`

Then, you can use **cvdb -g** to collect trace information such as this:

```
PERF: Device Write 41 MB/s IOs 2 exts 1 offs 0x0 len 0x400000 mics 95589 ino 0x5

PERF: VFS Write EofDmaAlgn 41 MB/s offs 0x0 len 0x400000 mics 95618 ino 0x5
```

The “PERF: Device” trace displays throughput measured for the device I/O. It also displays the number of I/Os into which it was broken, and the number of extents (sequence of consecutive filesystem blocks).

The “PERF: VFS” trace displays throughput measured for the read or write system call and significant aspects of the I/O, including:

- Dma: DMA
- Buf: Buffered
- Eof: File extended
- Algn: Well-formed DMA I/O
- Shr: File is shared by another client
- Rt: File is real time
- Zr: Hole in file was zeroed

Both traces also report file offset, I/O size, latency (mics), and inode number.

Sample use cases:

- Verify that I/O properties are as expected.

  You can use the VFS trace to ensure that the displayed properties are consistent with expectations, such as being well formed; buffered versus DMA; shared/non-shared; or I/O size. If a small I/O is being performed DMA, performance will be poor. If DMA I/O is not well formed, it requires an extra data copy and may even be broken into small chunks. Zeroing holes in files has a performance impact.
- Determine if metadata operations are impacting performance.
- If VFS throughput is inconsistent or significantly less than Device throughput, it might be caused by metadata operations. In that case, it would be useful to display “fsmtoken,” “fsmvnops,” and “fsmdmig” traces in addition to “perf.”
- Identify disk performance issues.
- If Device throughput is inconsistent or less than expected, it might indicate a slow disk in a stripe group, or that RAID tuning is necessary.
- Identify file fragmentation.
- If the extent count “exts” is high, it might indicate a fragmentation problem.This causes the device I/Os to be broken into smaller chunks, which can significantly impact throughput.
- Identify read/modify/write condition.

If buffered VFS writes are causing Device reads, it might be beneficial to match I/O request size to a multiple of the “cachebufsize” (default 64KB; see **mount_cvfs** man page). Another way to avoid this is by truncating the file before writing.

The **cvadmin** command includes a **latency-test** utility for measuring the latency between an FSM and one or more SNFS clients. This utility causes small messages to be exchanged between the FSM and clients as quickly as possible for a brief period of time, and reports the average time it took for each message to receive a response.

The **latency-test** command has the following syntax:

```
latency-test <index-number> [ <seconds> ]

latency-test all [ <seconds> ]
```

If an **index-number** is specified, the test is run between the currently-selected FSM and the specified client. (Client index numbers are displayed by the **cvadmin who** command). If **all** is specified, the test is run against each client in turn.

The test is run for 2 seconds, unless a value for seconds is specified.

Here is a sample run:

```
snadmin (lsi) > latency-test

Test started on client 1 (bigsky-node2)... latency 55us
Test started on client 2 (k4)... latency 163us
```

There is no rule-of-thumb for “good” or “bad” latency values. The observed latency for GbE is less than 60 microseconds. Latency can be affected by CPU load or SNFS load on either system, by unrelated Ethernet traffic, or other factors. However, for otherwise idle systems, differences in latency between different systems can indicate differences in hardware performance. (In the example above, the difference is a Gigabit Ethernet and faster CPU versus a 100BaseT Ethernet and a slower CPU.) Differences in latency over time for the same system can indicate new hardware problems, such as a network interface going bad.

If a latency test has been run for a particular client, the **cvadmin who long** command includes the test results in its output, along with information about when the test was last run.

Mount Command Options

The following SNFS mount command settings are explained in greater detail in the **mount_cvfs** man page.

The default size of the client buffer cache varies by platform and main memory size, and ranges between 32MB and 256MB. And, by default, each buffer is 64K so the cache contains between 512 and 4096 buffers. In general, increasing the size of the buffer cache will not improve performance for streaming reads and writes. However, a large cache helps greatly in cases of multiple concurrent streams, and where files are being written and subsequently read. Buffer cache size is adjusted with the buffercachecap setting.

The buffer cache I/O size is adjusted using the **cachebufsize** setting. The default setting is usually optimal; however, sometimes performance can be improved by increasing this setting to match the RAID 5 stripe size.

Note: In prior releases of StorNext, using a large `cachebufsize` setting could decrease small, random I/O READ performance. However, in StorNext 5, the buffer cache has been modified to avoid this issue.

The **cachebufsize** parameter is a mount option and can be unique for every client that mounts the file system.

Buffer cache read-ahead can be adjusted with the **buffercache_readahead** setting. When the system detects that a file is being read in its entirety, several buffer cache I/O daemons pre-fetch data from the file in the background for improved performance. The default setting is optimal in most scenarios.

The **auto_dma_read_length** and **auto_dma_write_length** settings determine the minimum transfer size where direct DMA I/O is performed instead of using the buffer cache for well-formed I/O. These settings can be useful when performance degradation is observed for small DMA I/O sizes compared to buffer cache.

For example, if buffer cache I/O throughput is 200 MB/sec but 512K DMA I/O size observes only 100MB/sec, it would be useful to determine which DMA I/O size matches the buffer cache performance and adjust **auto_dma_read_length** and **auto_dma_write_length** accordingly. The **lmdd** utility is handy here.

The **dircachesize** option sets the size of the directory information cache on the client. This cache can dramatically improve the speed of readdir operations by reducing metadata network message traffic between the SNFS client and FSM. Increasing this value improves performance in scenarios where very large directories are not observing the benefit of the client directory cache.

SNFS External API

The SNFS External API might be useful in some scenarios because it offers programmatic use of special SNFS performance capabilities such as affinities, preallocation, and quality of service. For more information, see the “Quality of Service” topic of the *StorNext File System API Guide* posted online at http://www.quantum.com/snsdocs.

Optimistic Allocation

Note: It is no longer recommended that the **`InodeExpand`** parameters (**`InodeExpandMin`**, **`InodeExpandMax`** and **`InodeExpandInc`**) be changed from their default values. These settings are provided for compatibility when upgrading file systems.

The `InodeExpand` values are still honored if they are in the .cfgx file, but the StorNext GUI does not allow these values to be set. Also, when converting from .cfg to .cfgx files, if the `InodeExpand` values in the .cfg file are found to be the default example values, these values are not set in the new .cfgx. Instead, the new formula is used.

How Optimistic Allocation Works

The `InodeExpand` values come into play whenever a write to disk is done, and works as an "optimistic allocator." It is referred to as “optimistic” because it works under the assumption that where there is one allocation, there will be another, so it allocates more than you asked for believing that you'll use the over-allocated space soon.

There are three ways to do a DMA I/O:

- By having an I/O larger than `auto_dma_write_length` (or `auto_dma_read_length`, but that does not cause an allocation so it will be ignored for this case)
- Doing a write to a file that was opened with `O_DIRECT`
- Opening a file for writes that's already open for writes by another client (commonly referred to as "shared write mode" which requires all I/Os go straight to disk to maintain coherency between the clients)

The first allocation is the larger of the `InodeExpandMin` or the actual IO size. For example, if the `InodeExpandMin` is 2MB and the initial IO is 1MB, the file gets a 2MB allocation. However, if the initial IO was 3MB and the `InodeExpandMin` is 2MB, the file gets only a 3MB allocation.

In both cases, the `InodeExpandMin` value is saved in an internal data structure in the file's inode, to be used with subsequent allocations. Subsequent DMA IOs that require more space to be allocated for the file add to the `InodeExpandInc` value saved in the inode, and the allocation is the larger of this value or the IO size.

For example, if `InodeExpandMin` is 2MB and `InodeExpandInc` is 4MB and the first I/O is 1MB, then the file is initially 2MB in size. On the third 1MB I/O the file is extended by 6MB (2MB + 4MB) and is now 8MB though it only has 3MB of data in it. However, that 6MB allocation is likely contiguous and therefore the file has at most 2 fragments which is better than 8 fragments it would have had otherwise.

Assuming there are more 1MB I/Os to the file, it will continue to expand in this manner. The next DMA I/O requiring an allocation over the 8MB mark will extend the file by 10MB (2MB + 4MB + 4MB). This pattern repeats until the file's allocation value is equal to or larger than `InodeExpandMax`, at which point it's capped at `InodeExpandMax`.

This formula generally works well when it's tuned for the specific I/O pattern. If it's not tuned, with certain I/O patterns it can cause suboptimal allocations resulting in excess fragmentation or wasted space from files being over allocated.

This is especially true if there are small files created with `O_DIRECT`, or small files that are simultaneously opened by multiple clients which cause them to use an `InodeExpandMin` that's too large for them. Another possible problem is an `InodeExpandMax` that's too small, causing the file to be composed of fragments smaller than it otherwise could have been created with.

With very large files, without increasing `InodeExpandMax`, it can create fragmented files due to the relatively small size of the allocations and the large number that are needed to create a large file.

Another possible problem is an `InodeExpandInc` that's not aggressive enough, again causing a file to be created with more fragments than it could be created with, or to never reach `InodeExpandMax` because writes stop before it can be incremented to that value.

Note: Although the preceding example uses DMA I/O, the `InodeExpand` parameters apply to both DMA and non-DMA allocations.

Optimistic Allocation Formula

The following table displays the new formula:

File Size (in bytes)	Optimistic Allocation
<= 16MB	1MB
16MB to 64MB + 4 bytes	4MB
64MB + 4 bytes to 256MB + 16 bytes	16MB
256MBs + 16 bytes to 1 GB + 64 bytes	64MB
1GB + 64 bytes to 4GB + 256 bytes	256MB
4GB + 256 bytes to 16GB + 1k bytes	1GB
16GB + 1k bytes to 64GB + 4k bytes	4GB
64GB + 4k bytes to 256GB + 16k bytes	16GB
256GB + 16k bytes to 1TB + 64k bytes	64GB
1TB + 64k bytes or larger	256GB

To examine how well these allocation strategies work in your specific environment, use the `snfsdefrag` utility with the `-e` option to display the individual extents (allocations) in a file.

Below is an example output from `snfsdefrag -e testvideo2.mov`:

```
testvideo2.mov:
# group frbase fsbase fsend kbytes depth
0 7 0x0 0xa86df6 0xa86df6 16 4
1 7 0x4000 0x1fb79b0 0x1fb79e1 800 4
HOLE @ frbase 0xcc000 for 41 blocks (656 kbytes)
2 7 0x170000 0x57ca034 0x57ca03f 192 4
3 7 0x1a0000 0x3788860 0x3788867 128 4
4 7 0x1c0000 0x68f6cb4 0x68f6cff 1216 4
5 7 0x2f0000 0x70839dd 0x70839df 48 4
```

Note: Beginning with StorNext 6, use the **`sgoffload`** command instead of the **`snfsdefrag`** command. The **`sgoffload`** command moves extents belonging to files that are currently in use (open). The **`sgoffload`** command also informs the client to suspend I/O for a time, moves the data, then informs the client to refresh the location of the data and resume I/O.

Here is an explanation of the column headings:

- #: This is the extent index.
- `group`: The group column tells you which stripe group on which the extent resides. Usually it's all on the same stripe group, but not always.
- `frbase`: This is the file's logical offset
- `fsbase` and `fsend`: These are the StorNext logical start and end addresses and should be ignored.
- `kbytes`: This is the size of the extent (fragment)
- `depth`: This tells you the number of LUNs that existed in the stripe group when the file was written. If you perform bandwidth expansion, this number is the old number of LUNs before bandwidth expansion, and signifies that those files aren't taking advantage of the bandwidth expansion.

If the file is sparse, you will see "`HOLE`" displayed. Having holes in a file isn't necessarily a problem, but it does create extra fragments (one for each side of the hole). Tuning to eliminate holes can reduce fragmentation, although it does that by using more disk space.

Special Considerations for StorNext LAN Clients

As with any client/server protocol, StorNext LAN performance is subject to the limitations of the underlying network. Therefore, it is strongly recommended that you use Gigabit (1000 BaseT) or 10GbE. Both TCP Offload and jumbo frames are recommended for 10GbE performance.

Hardware Configuration

A StorNext LAN can easily saturate several Gigabit Ethernet connections with data, so take special care when selecting and configuring the switches used to interconnect StorNext LAN Clients and gateway servers. Ensure that your network switches have enough internal bandwidth to handle all of the anticipated traffic between all StorNext LAN Clients and gateway servers connected to them.

A network switch that is dropping packets will cause TCP retransmissions. This can be easily observed on both Linux and Windows platforms by using the **netstat -s** command while the StorNext LAN Client is reading or writing data to the file system. Reducing the TCP window size used by the LAN might also help with an oversubscribed network switch. The Windows client **Distributed LAN** tab and the Linux **dpserver** file contain the tuning parameter for the TCP window size. Note that the gateway must unmount and remount the StorNext file system.

It is best practice to have all StorNext gateways directly attached to the same network switch. A router between gateways could be easily overwhelmed by the data rates required.

It is critical to ensure that **speed/duplex** settings are correct, as this will severely impact performance. Most of the time **auto-negotiation** is the correct setting. Some managed switches allow setting **speed/duplex**, such as **1000Mb/full**, which disables **auto-negotiation** and requires the host to be set exactly the same. However, performance is severely impacted if the settings do not match between switch and host. For example, if the switch is set to **auto-negotiation** but the host is set to **1000Mb/full**, you will observe a high error rate and extremely poor performance. On Linux the **ethtool** command can be very useful to investigate and adjust **speed/duplex** settings.

In some cases, TCP offload seems to cause problems with the StorNext LAN Clients by miscalculating checksums under heavy loads. This is indicated by **bad segments** indicated in the output of **netstat -s**. On Linux, the TCP offload state can be queried by running **ethtool -k**, and modified by running **ethtool -K**. On Windows it is configured through the **Advanced** tab of the configuration properties for a network interface.

The internal bus bandwidth of a StorNext Gateway Server can also place a limit on performance. A basic PCI- or PCI-X-based workstation might not have enough bus bandwidth to run multiple Gigabit Ethernet NICs at full speed; PCI Express is recommended but not required.

Similarly, the performance characteristics of NICs can vary widely and ultimately limit the performance of the StorNext LAN Client. For example, some NICs might be able to transmit or receive each packet at Gigabit speeds, but not be able to sustain the maximum needed packet rate. An inexpensive 32-bit NIC plugged into a 64-bit PCI-X slot is incapable of fully utilizing the host's bus bandwidth.

It can be useful to use a tool like **netperf** to help verify the performance characteristics of each StorNext LAN Client. (When using **netperf**, on a system with multiple NICs, take care to specify the right IP addresses in order to ensure the network being tested is the one you use for the StorNext LAN. For example, if **netperf -t TCP_RR -H <host>** reports less than 4,000 transactions per second capacity, a performance penalty might be incurred. Multiple copies of **netperf** can also be run in parallel to determine the performance characteristics of multiple NICs.

Software Tuning and Configuration

Full line speed is typically achievable for small configurations without tuning. However, when scaling to larger configurations (for example, 10GbE), tuning is often required for maximum performance. The following contains a set of tunables for such environments. The applicability of each tunable will depend on ecosystem details and application behavior. In some cases, experimentation may be required to determine optimal values.

Modify the grub.conf File

A fix is required prior to installing the StorNext file system and Storage Manager. This fix avoids potential clock jumps and temporary system freezes on some systems. This fix only affects RedHat Linux releases 6.0, 6.1, 6.2 and their CentOS counterparts on systems with Intel processor versions identified by the internal codename **Nehalem**.

Follow the steps below to modify the **grub.conf** file so that the Intel sleep state is disabled. Making this change could result in increased power consumption, but it helps prevent problems which result in system hangs due to processor power transition.

1. For the above systems, prior to installation, add the following text to the "kernel" line in `/boot/grub/`:

   ```
   grub.conf:idle=poll intel_idle.max_cstate=0 processor.max_cstate=1
   ```

2. Reboot the system for the change to take effect.

When creating a new file system, use Stripe Breadth values of 512K or larger

Where to Set This	On the StorNext MDC
How to Set This	Use the StorNext GUI, the Windows File System Cfg tool, or run the command **`sndcfgedit`**.
How This Helps	While the value of `transfer_buffer_size_kb` in the dpserver file determines the maximum transfer size used by DLC, this is artificially capped when the Stripe Breadth is a smaller value. Therefore, using a 512K Stripe Breadth allows maximum value of `transfer_buffer_size_kb` (512K) to be in effect.
Notes and Caveats	Using a value larger than 512K will not improve performance over 512K for DLC. The Stripe Breadth of a Stripe Group cannot be changed after a file system has been created without re-initializing the file system. While a value of 512K or larger is optimal for DLC network transfers, depending on the RAID type and configuration, it may not be optimal for SAN disk I/O.

Use the maximum value (1024) for transfer_buffer_size_kb in the dpservers file

Where to Set This	On Gateway systems
How to Set This	Run the command **`sndpscfg -e`** or **`sndpscfg -E`** *fsname*
How This Helps	The optional **transfer_buffer_size_kb** keyword specifies the size in Kilobytes of the socket transfer buffers used for Proxy Client I/O. The default value is 256 and values between 32 and 1024 are allowed. **Note:** The DLC is more efficient when larger transfer buffers are used. Quantum recommends you use **1024** for this setting to maximize performance for DLC clients.
Notes and Caveats	The transfer buffer size is artificially capped when smaller stripe breadths are used. See When creating a new file system, use Stripe Breadth values of 512K or larger on the previous page. Also, using a large values for `transfer_buffer_size_kb` and `transfer_buffer_count` can lead to significant memory utilization on the gateway systems since each proxy connection can use up to the product of the two values. This can become a real issue when there are many DLC clients and/or multiple file systems.

Use a larger than default value for transfer_buffer_count in the dpservers file

Where to Set This	On Gateway systems
How to Set This	Run the command **`sndpscfg -e`** or **`sndpscfg -E`** *fsname* and set `transfer_buffer_count` to some value greater than 16.
How This Helps	Having additional buffers may allow for better pipe-lining and processing multiple requests from a single client.
Notes and Caveats	Using a large values for `transfer_buffer_size_kb` and `transfer_buffer_count` can lead to significant memory utilization on the gateway systems since each proxy connection can use up to the product of the two values. This can become a real issue when there are many DLC clients and/or multiple file systems.

Use the default value (`0`) for tcp_window_size_kb in the dpservers file

Where to Set This	On Gateway systems
How to Set This	Run the command **`sndpscfg -e`** or **`sndpscfg -E`** *fsname*
How This Helps	TCP performance is limited when the advertised receive window is less than the delay-bandwidth product of the underlying network. The optional `tcp_window_size_kb` keyword specifies the size in Kilobytes of the TCP window used for Proxy Client I/O connections. The default value is **0**, and values between **0** and **16384** are allowed. The setting of **0** has a special meaning, which is that no change is made to the default system value. This allows Linux autotuning to adjust the receive buffer size and TCP window size dynamically for each connection. Quantum recommends this setting when auto-tuning is enabled, which is the default for recent Linux versions. **Note:** You must have auto-tuning enabled on your Linux host.
Notes and Caveats	In order for this to be effective, systems must have TCP Window Scaling (RFC1323) enabled. See additional tunables below. Very low-end networking components may not support or have resource capacity to handle large TCP windows.

Set the cache buffer size to 512K in the file system mount options

Where to Set This	On DLC clients
How to Set This	On **Linux**: Edit the file `/etc/fstab` and add the option `cachebufsize=512k` for the StorNext file system. On **Solaris**: Edit the file `/etc/vfstab` and add the option `cachebufsize=512k` for the StorNext file system. On **Windows**: Open the **Client Configuration** tool, double-click the file system, navigate to the **Advanced Cache Options** tab, and set the `Individual Buffer Size` to `512K`.
How This Helps	Using larger cache buffer sizes allows the file system to make larger requests to gateway systems when buffered I/O is used. A value of 512K allows transfers up to the maximum value of `transfer_buffer_size_kb`.
Notes and Caveats	Increasing the cache buffer size may negatively impact the performance of small, random I/Os. For this tunable to be effective, `transfer_buffer_size_kb` and stripe breadth should also be tuned. See above.

Note: The following information applies to systems configured with Appliance Controller.

The default StorNext file system mount options are not optimized for Appliance Controller. If Appliance Controller performance is not optimal, you may need to tune StorNext to optimize performance.

We recommend tuning the following.

cachebufsize

The cache buffer size is the amount of data that will be processed as a single block from the StorNext file system.

- Recommended setting for sequential I/O: 256 KB
- Recommended setting for random I/O (such as database applications): typical read/write size

> **Example**
>
> If your typical read/write size is 8 KB, set the cache buffer size to 8 KB.

- Default setting: 64 KB

> **Important**
>
> - Keep in mind that when you set the cache buffer size value, the StorNext file system will read/write the entire buffer size. So even if you are modifying a file that is only 4 KB, if you set the cache buffer size to 256 KB, the system will read/write the entire 256 KB data block.
> - When you increase the cache buffer size, you must also increase the buffer cache cap. See buffercachecap below.
> - You must optimize the raid array performance. See The Underlying Storage System in the *StorNext Documentation Center*.

buffercachecap

The buffer cache cap is the total amount of memory reserved for caching data. The reserved cache memory is shared by all mount points with the same cache size.

- Recommended Setting (dependent upon the amount of available memory): 4096 MB or 8192 MB
- Default Setting: 256 MB

When reserving cache memory, you need to take into account all processes that run on your system, such as smbd and FSM, so that you don't oversubscribe your system.

dircachesize

The directory cache size sets the size of the directory information cache on the Appliance Controller System. By increasing this value, the Appliance Controller System is able to keep more directory structure data in memory, dramatically improving the speed of **`readdir`** operations by reducing metadata network message traffic between it and FSM.

- Recommend Setting: 32 MB
- Default Setting: 10 MB

buffercache_iods

The buffer cache I/O daemons setting defines the number of background daemons used for performing buffer cache I/O.

- Recommended Setting: 16
- Default Setting: 8

Use large values for “auto_dma” settings in the file system mount options

Where to Set This	On Linux and Windows DLC clients
How to Set This	On **Linux**: Edit the file `/etc/fstab` and add the options `auto_dma_read_length=2g`, `auto_dma_write_length=2g` for the StorNext file system. On **Solaris**: Does not apply. On **Windows**: Open the **Client Configuration** tool, double-click the file system, navigate to the **Advanced Cache Options** tab, and set the `Auto-DMA Read Size` and `Auto-DMA Write Size` to 2G.
How This Helps	By default, StorNext uses DMA when performing large, well-formed I/Os. This is typically a performance win for SAN-client access, but the additional latencies in LAN client often cause DMA to under-perform buffered I/O. By increasing the `auto_dma` settings, LAN client will used buffered I/O in more cases.
Notes and Caveats	With these settings in place, additional CPU and memory bandwidth utilization may occur when large, well-formed I/Os are used compared with allowing such requests to use DMA. On modern systems with increased memory bandwidth and CPU cores, this additional overhead often does not have a significant impact.

Enable TCP Window Scaling (RFC1323)

Where to Set This	On DLC clients and gateway systems

How to Set This	On **Linux**: Modern versions of Linux have TCP window scaling enabled by default. However, the value can be set explicitly using the **`sysctl`** command. For example, **`sysctl -w net.ipv4.tcp_window_scaling=1`**. To determine the correct kernel parameter to adjust, refer to the documentation for your version of Linux. On **Solaris**: Newer versions of Solaris have TCP window scaling enabled by default. However, the value can be set explicitly using the **`ndd`** command. For example, **`ndd -set /dev/tcp tcp_wscale_always 1`**. To determine the correct kernel parameter to adjust, refer to the documentation for your version of Solaris. On **Windows**: For Vista and newer, TCP window scaling is enabled by default. For previous version of Windows including Windows 2003 and Window XP, add or set the `DWORD` key: `HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\Tcpip\Parameters\Tcp1323Opts` To the value of `3`.
How This Helps	When Window scaling is enabled, the advertised TCP receive window can be larger than 64K. This is required to achieve maximum per-connection bandwidth when high-speed or high-latency networks are used.
Notes and Caveats	Window scaling must be enabled on both ends of a connection to be effective. In the case of DLC, this means the gateway and LAN client.

Increase the system maximum TCP Window size

Where to Set This	On DLC clients and gateway systems

How to Set This	On **Linux**: Run the **`sysctl`** command to adjust `rmem_max` and `wmem_max`.
	
	For example, **`sysctl`** `-w net.core.rmem_max=4194304`**`sysctl`** `-w net.core.wmem_max=4194304`
	
	The exact syntax may vary by Linux version. For details, refer to the documentation for your version of Linux.
	
	On **Solaris**: Run **ndd**. For example, **`ndd`** `-set /dev/tcp tcp_max_buf  4194304`
	
	The exact syntax may vary by Solaris version. For details, refer to the documentation for your version of Solaris.
	
	On **Windows**: Systems running Vista or newer do not require adjustment. For older versions of Windows, add or set the `DWORD` keys:
	`HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\Tcpip\Parameters\TcpWindowSize`
	
	`KEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\Tcpip\Parameters\GlobalMaxTcpWindowSize`
	
	These should both be set to a value of `4MB` (`0x400000` or `4194304`) or greater.

How This Helps	High-speed and high-latency networks require large TCP windows to achieve full bandwidth. The `tcp_window_size_kb` tunable is supposed to set the window for the connection but it can be capped to a smaller value due to the system-imposed maximum.
Notes and Caveats	Most modern operating systems set the maximum to a large enough value by default.

Use Jumbo Frames (aka large MTU)

Where to Set This	On DLC clients and gateway systems
How to Set This	On **Linux**: To modify the setting temporarily, use the **`ifconfig`** command. For example, **`ifconfig`** `en6 mtu 9000` or to configure Jumbo frames permanently, manually edit the setting by adding `MTU=9000` in `/etc/sysconfig/network-script/ifcfg-<iface>` or use the network configuration GUI. Refer to the documentation for your version of Linux for details. On **Solaris**: This is typically done by configuring the setting `accept-jumbo=1` in the interface configuration. Refer to your Solaris documentation. On **Windows**: Refer to the driver documentation for your NIC.

How This Helps	For large data transfers, Jumbo frames reduce the number of Ethernet frames that need to be processed and increase the percentage of payload data on the wire. This results in higher payload bandwidth and lower CPU overhead.
Notes and Caveats	To be effective, Jumbo frames must be supported and enabled on all networking hardware and software components between the sender and receiver. Not all Jumbo frame implementations are compatible. When specifying a larger MTU, depending on the context, the value may need to include the header. For example, `9216` versus `9000`. The use of jumbo frames may slightly increase latency when performing very small I/O. The benefit of reduced CPU utilization may not be significant if TCP offload is also enabled.

Use TCP offload features

Where to Set This	On DLC clients and gateway systems
How to Set This	refer to the documentation for your NIC driver.
How This Helps	TCP processing is very CPU intensive. Most modern NICs contain special hardware for offloading some or all TCP processing. This may include: • **Transmit Checksum Offload** • **Receive Checksum Offload** • **Large Segment Offload** • **Full TCP Stack Offload** (on Windows this is called **Chimney**)
Notes and Caveats	Use of offload features may cause system instability or degraded performance in some cases. In most cases, full TCP stack offload cannot be used when host-based software firewalls (such as Windows Firewall) are enabled. When Full TCP stack offload is used, TCP connections will use congestion control and other TCP algorithms that are hard-wired in the NIC which may be sub-optimal depending on the TCP stack being used by the other end of the connection.

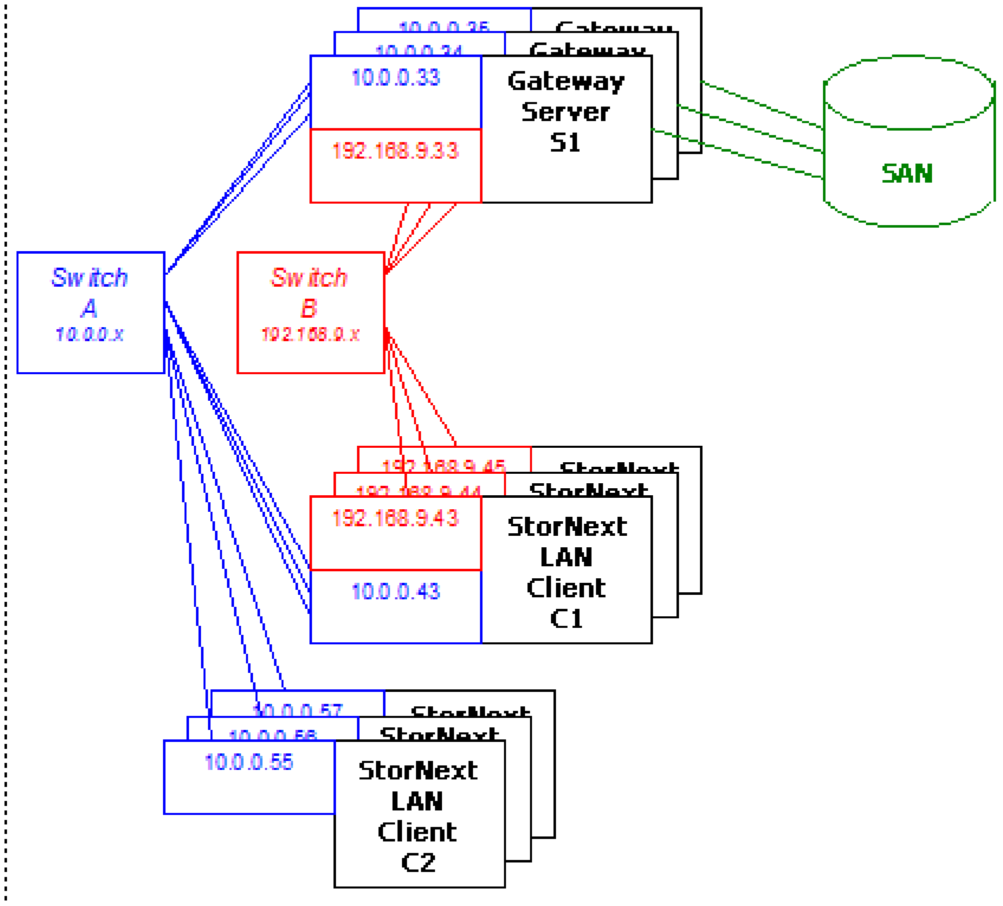
Tune Proxypath based on workload

Where to Set This	On DLC clients
How to Set This	On **Linux** or **Solaris**: Edit the file `/etc/fstab` and add the `proxypath` mount option with the appropriate value. On **Windows**: Open the **Client Configuration** tool, double-click on the file system, navigate to the **Distributed LAN** tab and select the appropriate value from the pull-down menu labeled **Proxypath Mount Option**.

How This Helps	Depending on the application profile, the default value of file sticky balance may not be appropriate. If most I/O is to one or a few files, the balance option will probably do better at load balancing to the servers.
Notes and Caveats	For additional information on **Linux** or **Solaris**, refer to the description of `proxypath` in the `mount_cvfs` man-page. For additional information on **Windows**, navigate to **Mount a StorNext File System** within StorNext Help and look the description of `proxypath`.

Network Configuration and Topology

For maximum throughput, a StorNext LAN Client can use multiple NICs on StorNext Gateway Servers. In order to take advantage of this feature, each of the NICs on a given gateway must be on a different IP subnetwork (this is a requirement of TCP/IP routing, not of SNFS - TCP/IP cannot utilize multiple NICs on the same subnetwork). An example of this is shown in the following illustration.



In the diagram there are two subnetworks: the blue subnetwork (10.0.0.x) and the red subnetwork (192.168.9.x). Servers such as S1 are connected to both the blue and red subnetworks, and can each provide up to 2 GByte/s of throughput to clients. (The three servers shown would thus provide an aggregate of 6 GByte/s.)

Clients such as C1 are also connected to both the blue and red subnetworks, and can each get up to 2 GByte/s of throughput. Clients such as C2 are connected only to the blue subnetwork, and thus get a maximum of 1 GByte/s of throughput. SNFS automatically load-balances among NICs and servers to maximize throughput for all clients.

Note: The diagram displays separate physical switches used for the two subnetworks. They can, in fact, be the same switch, provided it has sufficient internal bandwidth to handle the aggregate traffic.

Scheduling requests across multiple subnetworks and multiple servers via multiple network ports can be challenging. In particular, multiple streams of large disk read requests, because of the additional latency from disk, can lead to an imbalance of traffic across a client's network ports. In some cases, it may be possible to tune this scheduling for a particular application mix using the **`proxypath`** mount options. In other cases, changing the network configuration might help. Matching the number of server ports to the number of client ports, thereby reducing the number of path choices, has been shown to improve the performance of multiple streams of large reads.

For a detailed description of the **`proxypath`** mount options, see the **`mount_cvfs`** man page.

StorNext Gateway Servers

StorNext Gateway Servers must have sufficient memory. When a gateway server does not have sufficient memory, its performance in servicing StorNext LAN I/O requests might suffer. In some cases (particularly on Windows) it might hang.

See Gateway Server/Client Network and Memory Tuning for additional information.

StorNext Gateway Servers must also have sufficient bus bandwidth. As discussed above, the host must have sufficient bus bandwidth to operate the NICs used for StorNext LAN I/O at full speed, while at the same time operating their Fibre Channel HBAs.

Note: Quantum strongly recommends using PCI Express for gateway servers.

StorNext LAN Client vs. Legacy Network Attached Storage

StorNext provides support for legacy Network Attached Storage (NAS) protocols, including Network File System (NFS) and Common Internet File System (CIFS).

However, using StorNext LAN Clients for NAS connectivity provides several compelling advantages in the following areas:

- Performance
- Fault Tolerance
- Load Balancing
- Client Scalability
- Robustness and Stability
- Security Model Consistency

Performance

The StorNext LAN Clients outperform NFS and CIFS for single-stream I/O and provide higher aggregate bandwidth. For inferior NFS client implementations, the difference can be more than a factor of two.The The StorNext LAN Client also makes extremely efficient use of multiple NICs (even for single streams), whereas legacy NAS protocols allow only a single NIC to be used. In addition, StorNext LAN Clients communicate directly with StorNext metadata controllers instead of going through an intermediate server, thereby lowering IOP latency.

Fault Tolerance

StorNext LAN Clients handle faults transparently, where possible. If an I/O is in progress and a NIC fails, the I/O is retried on another NIC (if one is available). If a StorNext Gateway Server fails while an I/O is in flight, the I/O is retried on another server (if one is running). When faults occur, applications performing I/O will experience a delay but not an error, and no administrative intervention is required to continue operation. These fault tolerance features are automatic and require no configuration.

Load Balancing

StorNext LAN Clients automatically makes use of all available gateway servers in an active/active fashion, and evenly spreads I/O across them. If a server goes down or one is added, the load balancing system automatically adjusts to support the new configuration. For more information on load balancing options, refer to the **`cvadmin`** command in the Man Pages Reference Guide.

Consistent Security Model

StorNext LAN Clients have the same security model as StorNext SAN Clients. When CIFS and NFS are used, some security models aren’t supported. (For example, Windows ACLs are not accessible when running UNIX Samba servers.)

Windows Memory Requirements

When using StorNext with 32-bit Windows clients running older version of StorNext, the client settings may require tuning to avoid kernel memory pressure. If your environment includes such clients, refer to the **Windows Memory Requirements** section in the *StorNext Tuning Guide* for the version of StorNext being used with the systems.

Windows Performance Monitor Counters

For StorNext File Systems installed on Microsoft Windows servers, Windows provides a way to display performance for individual components in use on that StorNext system.

Performance counters are not enabled by default in the client.

Enable Performance Counters

To enable the performance counters, enter the following on the command line:

```
cvdb -P
```

The command, **cvdb -P**, also toggles the state of the counters between on and off. To clear all counters, disable the counters, and then re-enable the counters with **cvdb -P**.

View Performance Counters

To view the performance monitor counters, perform the following procedure.

Note: The following instructions apply to versions of Windows supported by StorNext (for example, Windows Vista and newer). Refer to earlier versions of the *StorNext Tuning Guide* when enabling **Windows Performance Monitor Counters** for previous versions of Windows.

1. To start the performance monitor, on the **Windows** menu, click **Start**.
2. In the **Search programs and files** dialog, type the following:

   ```
   perfmon
   ```

3. Press **Enter**.

4. Click **Add Counter**.
5. Select either **StorNext Client** or **StorNext Disk Agent**.

   Note: The **StorNext Disk Agent** counters are internal debugging/diagnostic counters used by Quantum personnel and are not helpful in performance tuning of your system.

6. Select an individual counter.
7. **(Optional)** To display additional information about the counter, enable **Show description**.

Cpuspeed Service Issue on Linux

Cpuspeed, an external Linux service on recent Intel processors, is not correctly tuned to allow StorNext to take advantage of processor speed. SUSE systems may also be impacted, as may AMD processors with similar capabilities.

On processors with a variable clockspeed (turboboost), the cpuspeed service on Red Hat controls the actual running speed of the processors based on system load.

A workload such as a heavily used FSM and probably Storage Manager does not register as something which needs a faster CPU. Disabling the cpuspeed service has been shown to double metadata performance on affected hardware.

Looking at the reported CPU clock speed by doing **cat /proc/cpuinfo** while the system is under load displays if a system is impacted by this issue.

Disable CPU Power Saving States

It is strongly recommended that CPU power saving states be disabled if they are supported by your system's CPUs. Making this change may result in increased power consumption but improves stability and performance. This applies to all systems running StorNext including clients.

Disable CPU Power Saving States on a Redhat or SUSE Linux System

For RedHat Linux (release 7 and later)

1. Add the following text to the `GRUB_CMDLINE_LINUX` line in `/etc/default/grub`:

```
intel_idle.max_cstate=0
processor.max_cstate=1
```

2. Run **`grub2-mkconfig`**.

```
# grub2-mkconfig –o /boot/grub2/grub.cfg
```

3. Reboot the system for the change to take effect.

For RedHat Linux (releases prior to 7) or SUSE Linux

1. Add the following text to the `kernel` line in `/boot/grub/menu.lst`:

```
intel_idle.max_cstate=0
processor.max_cstate=1
```

2. Reboot the system for the change to take effect.

Disable CPU Power Saving States on a Debian Linux System

To disable CPU power saving states on a Debian Linux system, perform the following procedure:

1. Modify the `GRUB_CMDLINE_LINUX` string in `/etc/default/grub` so that it contains:

```
intel_idle.max_cstate=0
processor.max_cstate=1
```

2. Run "**`update-grub`**".
3. Reboot the system for the change to take effect.

Note: Disabling CPU power saving states in the system BIOS has no effect on Linux.

In some cases, performance can also be improved by adjusting the idle kernel parameter. However, care should be taken when using certain values. For example, `idle=poll` maximizes performance but is incompatible with hyperthreading (HT) and will lead to very high power consumption. For additional information, refer to the documentation for your version of Linux.

On Windows, disable CPU power saving states by adjusting BIOS settings. Refer to system vendor documentation for additional information.

Example FSM Configuration File

On Linux, the StorNext configuration file uses an XML format (.cfgx). On Windows, the configuration file uses a text format (.cfg). However, the values contained in both files are similar.

You can locate an example StorNext configuration file in the following directory:

- Linux — **/usr/cvfs/examples/example.cfgx**
- Windows — **C:\Program Files\Stornext\config\example.cfg**

Note: If you installed StorNext in a location other than the default installation directory, the example configuration file is located in **C:\<install_directory>\config\example.cfg**

Linux Example Configuration File

Below are the contents of the StorNext example configuration file for Linux (**example.cfgx**):

```
<?xml version="1.0"?>

<configDoc xmlns="http://www.quantum.com/snfs" version="1.0">

 <config configVersion="0" name="example" fsBlockSize="4096"
journalSize="67108864">

 <globals>

 <affinityPreference>false</affinityPreference>

 <allocationStrategy>round</allocationStrategy>

 <haFsType>HaUnmonitored</haFsType>

 <bufferCacheSize>268435456</bufferCacheSize>

 <cvRootDir>/</cvRootDir>

 <storageManager>false</storageManager>

 <debug>00000000</debug>

 <dirWarp>true</dirWarp>

 <extentCountThreshold>49152</extentCountThreshold>

 <enableSpotlight>false</enableSpotlight>

 <enforceAcls>false</enforceAcls>

 <fileLocks>false</fileLocks>

 <fileLockResyncTimeOut>20</fileLockResyncTimeOut>

 <forcePerfectFit>false</forcePerfectFit>
```

```
<fsCapacityThreshold>0</fsCapacityThreshold>
<globalSuperUser>true</globalSuperUser>
<inodeCacheSize>131072</inodeCacheSize>
<inodeExpandMin>0</inodeExpandMin>
<inodeExpandInc>0</inodeExpandInc>
<inodeExpandMax>0</inodeExpandMax>
<inodeDeleteMax>0</inodeDeleteMax>
<inodeStripeWidth>0</inodeStripeWidth>
<maintenanceMode>false</maintenanceMode>
<maxLogs>4</maxLogs>
<namedStreams>false</namedStreams>
<remoteNotification>false</remoteNotification>
<renameTracking>false</renameTracking>
<reservedSpace>true</reservedSpace>
<fsmRealTime>false</fsmRealTime>
<fsmMemLocked>false</fsmMemLocked>
<opHangLimitSecs>180</opHangLimitSecs>
<perfectFitSize>131072</perfectFitSize>
<quotas>false</quotas>
<quotaHistoryDays>7</quotaHistoryDays>
<restoreJournal>false</restoreJournal>
<restoreJournalDir></restoreJournalDir>
<restoreJournalMaxHours>0</restoreJournalMaxHours>
<restoreJournalMaxMb>0</restoreJournalMaxMb>
<stripeAlignSize>-1</stripeAlignSize>
<trimOnClose>0</trimOnClose>
<useL2BufferCache>true</useL2BufferCache>
<unixDirectoryCreationModeOnWindows>755</unixDirectoryCreationModeOnWindows>
<unixIdFabricationOnWindows>false</unixIdFabricationOnWindows>
```

```
 <unixFileCreationModeOnWindows>644</unixFileCreationModeOnWindows>
 <unixNobodyUidOnWindows>60001</unixNobodyUidOnWindows>
 <unixNobodyGidOnWindows>60001</unixNobodyGidOnWindows>
 <windowsSecurity>true</windowsSecurity>
 <globalShareMode>false</globalShareMode>
 <useActiveDirectorySFU>true</useActiveDirectorySFU>
 <eventFiles>true</eventFiles>
 <eventFileDir></eventFileDir>
 <allocSessionReservationSize>0</allocSessionReservationSize>
 </globals>
 <diskTypes>
 <diskType typeName="MetaDrive" sectors="99999999" sectorSize="512"/>
 <diskType typeName="JournalDrive" sectors="99999999" sectorSize="512"/>
 <diskType typeName="VideoDrive" sectors="99999999" sectorSize="512"/>
 <diskType typeName="AudioDrive" sectors="99999999" sectorSize="512"/>
 <diskType typeName="DataDrive" sectors="99999999" sectorSize="512"/>
 </diskTypes>
 <autoAffinities/>
 <stripeGroups>
 <stripeGroup index="0" name="MetaFiles" status="up" stripeBreadth="262144"
read="true" write="true" metadata="true" journal="false" userdata="false"
realTimeIOs="0" realTimeIOsReserve="0" realTimeMB="0" realTimeMBReserve="0"
realTimeTokenTimeout="0" multipathMethod="rotate">
 <disk index="0" diskLabel="CvfsDisk0" diskType="MetaDrive" ordinal="0"/>
 </stripeGroup>
 <stripeGroup index="1" name="JournFiles" status="up" stripeBreadth="262144"
read="true" write="true" metadata="false" journal="true" userdata="false"
realTimeIOs="0" realTimeIOsReserve="0" realTimeMB="0" realTimeMBReserve="0"
realTimeTokenTimeout="0" multipathMethod="rotate">
 <disk index="0" diskLabel="CvfsDisk1" diskType="JournalDrive" ordinal="1"/>
 </stripeGroup>
 <stripeGroup index="2" name="VideoFiles" status="up" stripeBreadth="4194304"
read="true" write="true" metadata="false" journal="false" userdata="true"
realTimeIOs="0" realTimeIOsReserve="0" realTimeMB="0" realTimeMBReserve="0"
realTimeTokenTimeout="0" multipathMethod="rotate">
 <affinities exclusive="true">
 <affinity>Video</affinity>
 </affinities>
 <disk index="0" diskLabel="CvfsDisk2" diskType="VideoDrive" ordinal="2"/>
 <disk index="1" diskLabel="CvfsDisk3" diskType="VideoDrive" ordinal="3"/>
 <disk index="2" diskLabel="CvfsDisk4" diskType="VideoDrive" ordinal="4"/>
 <disk index="3" diskLabel="CvfsDisk5" diskType="VideoDrive" ordinal="5"/>
 <disk index="4" diskLabel="CvfsDisk6" diskType="VideoDrive" ordinal="6"/>
 <disk index="5" diskLabel="CvfsDisk7" diskType="VideoDrive" ordinal="7"/>
 <disk index="6" diskLabel="CvfsDisk8" diskType="VideoDrive" ordinal="8"/>
 <disk index="7" diskLabel="CvfsDisk9" diskType="VideoDrive" ordinal="9"/>
 </stripeGroup>
 <stripeGroup index="3" name="AudioFiles" status="up" stripeBreadth="1048576"
read="true" write="true" metadata="false" journal="false" userdata="true"
realTimeIOs="0" realTimeIOsReserve="0" realTimeMB="0" realTimeMBReserve="0"
realTimeTokenTimeout="0" multipathMethod="rotate">
 <affinities exclusive="true">
 <affinity>Audio</affinity>
 </affinities>
 <disk index="0" diskLabel="CvfsDisk10" diskType="AudioDrive" ordinal="10"/>
 <disk index="1" diskLabel="CvfsDisk11" diskType="AudioDrive" ordinal="11"/>
 <disk index="2" diskLabel="CvfsDisk12" diskType="AudioDrive" ordinal="12"/>
 <disk index="3" diskLabel="CvfsDisk13" diskType="AudioDrive" ordinal="13"/>
 </stripeGroup>
 <stripeGroup index="4" name="RegularFiles" status="up" stripeBreadth="262144"
read="true" write="true" metadata="false" journal="false" userdata="true"
realTimeIOs="0" realTimeIOsReserve="0" realTimeMB="0" realTimeMBReserve="0"
realTimeTokenTimeout="0" multipathMethod="rotate">
 <disk index="0" diskLabel="CvfsDisk14" diskType="DataDrive" ordinal="14"/>
```

```
  <disk index="1" diskLabel="CvfsDisk15" diskType="DataDrive" ordinal="15"/>
  <disk index="2" diskLabel="CvfsDisk16" diskType="DataDrive" ordinal="16"/>
  <disk index="3" diskLabel="CvfsDisk17" diskType="DataDrive" ordinal="17"/>
  </stripeGroup>
  </stripeGroups>
  </config>
</configDoc>
```

Windows Example Configuration File

Below are the contents of the StorNext example configuration file for Windows (**example.cfg**):

```
# Globals

AffinityPreference no
AllocationStrategy Round
HaFsType HaUnmonitored
FileLocks No
BrlResyncTimeout 20
BufferCacheSize 256M
CvRootDir /
DataMigration No
Debug 0x0
DirWarp Yes
ExtentCountThreshold 48K
EnableSpotlight No
ForcePerfectFit No
FsBlockSize 4K
GlobalSuperUser Yes
InodeCacheSize 128K
InodeExpandMin 0
InodeExpandInc 0
```

```
InodeExpandMax 0

InodeDeleteMax 0

InodeStripeWidth 0

JournalSize 64M

MaintenanceMode No

MaxLogs 4

NamedStreams No

PerfectFitSize 128K

RemoteNotification No

RenameTracking No

ReservedSpace Yes

FSMRealtime No

FSMMemlock No

OpHangLimitSecs 180

Quotas No

QuotaHistoryDays 7

RestoreJournal No

RestoreJournalMaxHours 0

RestoreJournalMaxMB 0

StripeAlignSize -1

TrimOnClose 0

UseL2BufferCache Yes

UnixDirectoryCreationModeOnWindows 0755

UnixIdFabricationOnWindows No

UnixFileCreationModeOnWindows 0644

UnixNobodyUidOnWindows 60001

UnixNobodyGidOnWindows 60001

WindowsSecurity Yes

GlobalShareMode No
```

```
UseActiveDirectorySFU Yes
EventFiles Yes
AllocSessionReservationSize 0m
# Disk Types

[DiskType MetaDrive]
Sectors 99999999
SectorSize 512
[DiskType JournalDrive]
Sectors 99999999
SectorSize 512
[DiskType VideoDrive]
Sectors 99999999
SectorSize 512
[DiskType AudioDrive]
Sectors 99999999
SectorSize 512
[DiskType DataDrive]
Sectors 99999999
SectorSize 512
# Disks

[Disk CvfsDisk0]
Type MetaDrive
Status UP
[Disk CvfsDisk1]
Type JournalDrive
Status UP
[Disk CvfsDisk2]
```

```
Type VideoDrive
Status UP
[Disk CvfsDisk3]
Type VideoDrive
Status UP
[Disk CvfsDisk4]
Type VideoDrive
Status UP
[Disk CvfsDisk5]
Type VideoDrive
Status UP
[Disk CvfsDisk6]
Type VideoDrive
Status UP
[Disk CvfsDisk7]
Type VideoDrive
Status UP
[Disk CvfsDisk8]
Type VideoDrive
Status UP
[Disk CvfsDisk9]
Type VideoDrive
Status UP
[Disk CvfsDisk10]
Type AudioDrive
Status UP
[Disk CvfsDisk11]
Type AudioDrive
Status UP
```

```
[Disk CvfsDisk12]
Type AudioDrive
Status UP
[Disk CvfsDisk13]
Type AudioDrive
Status UP
[Disk CvfsDisk14]
Type DataDrive
Status UP
[Disk CvfsDisk15]
Type DataDrive
Status UP
[Disk CvfsDisk16]
Type DataDrive
Status UP
[Disk CvfsDisk17]
Type DataDrive
Status UP
# Stripe Groups

[StripeGroup MetaFiles]
Status Up
StripeBreadth 256K
Metadata Yes
Journal No
Exclusive Yes
Read Enabled
Write Enabled
Rtmb 0
```

```
Rtios 0
RtmbReserve 0
RtiosReserve 0
RtTokenTimeout 0
MultiPathMethod Rotate
Node CvfsDisk0 0

[StripeGroup JournFiles]
Status Up
StripeBreadth 256K
Metadata No
Journal Yes
Exclusive Yes
Read Enabled
Write Enabled
Rtmb 0
Rtios 0
RtmbReserve 0
RtiosReserve 0
RtTokenTimeout 0
MultiPathMethod Rotate
Node CvfsDisk1 0

[StripeGroup VideoFiles]
Status Up
StripeBreadth 4M
Metadata No
Journal No
Exclusive No
```

```
Read Enabled
Write Enabled
Rtmb 0
Rtios 0
RtmbReserve 0
RtiosReserve 0
RtTokenTimeout 0
MultiPathMethod Rotate
Node CvfsDisk2 0
Node CvfsDisk3 1
Node CvfsDisk4 2
Node CvfsDisk5 3
Node CvfsDisk6 4
Node CvfsDisk7 5
Node CvfsDisk8 6
Node CvfsDisk9 7
Affinity Video

[StripeGroup AudioFiles]
Status Up
StripeBreadth 1M
Metadata No
Journal No
Exclusive No
Read Enabled
Write Enabled
Rtmb 0
Rtios 0
RtmbReserve 0
```

```
RtiosReserve 0
RtTokenTimeout 0
MultiPathMethod Rotate
Node CvfsDisk10 0
Node CvfsDisk11 1
Node CvfsDisk12 2
Node CvfsDisk13 3
Affinity Audio

[StripeGroup RegularFiles]
Status Up
StripeBreadth 256K
Metadata No
Journal No
Exclusive No
Read Enabled
Write Enabled
Rtmb 0
Rtios 0
RtmbReserve 0
RtiosReserve 0
RtTokenTimeout 0
MultiPathMethod Rotate
Node CvfsDisk14 0
Node CvfsDisk15 1
Node CvfsDisk16 2
Node CvfsDisk17 3
```

StorNext Storage Manager

StorNext Storage Manager uses a MySQL database for storing managed StorNext file system information.

StorNext Storage Manager can run on Linux MDCs and requires a Storage Manager license. Tuning the MySQL database can improve the performance of Storage Manager and other parts StorNext.

MySQL innodb_buffer_pool_size

The InnoDB buffer pool is used to cache the data and indexes of MySQL InnoDB database tables.

Increasing the size of the InnoDB buffer pool will allow MySQL to keep more of the working database set memory resident, thereby reducing the amount of disk access required to read datasets from the file system. The InnoDB buffer pool size is determined by the parameter **`innodb_buffer_pool_size`** in the `/usr/adic/mysql/my.cnf` file.

Increasing this value can improve the performance of Storage Manager operations that require large queries. However, setting this value too high can inefficiently remove memory from the free pool that could otherwise be used by the StorNext file system or other applications, and could lead to memory starvation issues on the system. The default value value of **`innodb_buffer_pool_size`** is 8 G. The following are the default values for the StorNext Appliances:

Metadata Controller	Size
M220	10 GB
M330	5 GB
M44x	10 GB
M66x	40 GB
Xcellis	10 GB

Quantum recommends setting this value to at least 8 GB (except on the M330 where 5 GB is recommended). This value may need to be decreased if memory starvation is observed or if the **Linux Out-of-Memory (OOM)** Killer is triggered.

To change this value, modify the `/usr/adic/mysql/my.cnf` file and change the **`innodb_buffer_pool_size`** setting in the `.mysqld.group`.

Example `/usr/adic/mysql/my.cnf`:

```
[mysqld]

innodb_buffer_pool_size = 8G
```

Both Storage Manager and MySQL will need to be restarted for the change to `/usr/adic/mysql/my.cnf` to take effect by executing the following commands:

```
# adic_control stop
# adic_control start
```

StorNext High Availability

When a FSM for a file system activates, it invokes the **snactivated** script. You can tune the **snactivated** script using the file, **/usr/cvfs/config/snactivated.conf**. The **snactvated.conf** file is mirrored by **syncha** to keep the contents in sync between the HA pair.

Note: The **snactivated.conf** file is sourced by the **/usr/cvfs/lib/snactivated** shell script and must conform to **/bin/sh** syntax.

SNACTIVATED_MOUNT_TIMEOUT=<integer>

You can use this parameter to configure the number of seconds to wait for StorNext file systems to mount before starting the remainder of the StorNext services.

Note: The default timeout is 300 seconds.

SNACTIVATED_MOUNT_RETRY_SLEEP=<integer>

You can use this parameter to configure the number of seconds that **snactivated** should wait between attempting to remount StorNext file systems while waiting for all StorNext file systems to mount.

Note: The default sleep is 5 seconds.

SNACTIVATED_MANAGED_FS_DELAY=<integer>

You can use this parameter to configure the number of seconds that **snactivated** will sleep to help allow FSMs to start before starting the rest of the StorNext services using **adic_control start**.

Note: The default is the same value that is set for **SNACTIVATED_MOUNT_RETRY_SLEEP**.

SNACTIVATED_SMITH_ON_TSM_FAILURE=<integer>

You can use this parameter to determine the behavior of **snactivated** when Storage Manager is licensed and fails to start when the HaShared file system activates.

Note: By default, **snactivated** invokes a SMITH reset if Storage Manager is licensed and TSM fails to start.

- Setting this to **0** causes **snactivated** to generate a RAS message instead of invoking a SMITH reset if TSM fails to start.
- Setting this to a non-zero integer value causes **snactivated** to SMITH to allow the peer node to attempt to start TSM should TSM not startup up successfully.
- This setting has no effect if Storage Manager is not licensed.
- This setting might be necessary to prevent SMITH resets should part of the system be unavailable for maintenance.

Chapter 2: Allocation Session Reservation (ASR)

This chapter contains the following topics:

Allocation Session Reservation (ASR)

Starting with StorNext 4.2, the Allocation Session Reservation (ASR) feature provides another method of allocating space to regular files. ASR optimizes on-disk allocation behavior in work-flows (such as some rich media streaming applications) which write and read sequences of files of certain sizes in specific directories.

With ASR, file sequences in a directory are usually placed on disk based on the order in which they are written. ASR keeps these files together even if other applications are writing at the same time in different directories or from different StorNext clients. The feature also has the ability to reduce file system free space fragmentation since collections of files which are written together are then typically removed together.

The workflows which see reduced free space fragmentation are those which have concurrent applications each writing files in their own directories and using files mostly larger than 1MB each. With this kind of work-flow, when a collection of files from one application is removed, the space is freed up in big chunks which are independent of other application runs.

Some workflows will see worse performance and may also see more free space fragmentation with ASR. These workflows are those which have concurrent applications all using the same directory on the same client, or all writing the same file on different clients. Additionally, performance may be adversely affected when stripe groups are configured and used to distribute applications. See How ASR Works on the next page.

Some applications depend on stripe alignment for performance. Stripe alignment can cause the allocator to chop an allocation request to make its head and tail land on a stripe boundary. The ASR feature disables stripe alignment since the chopping can lead to even more free space fragmentation since the chopping is within ASR chunks.

Customers should run with ASR and see if performance is adversely affected. You can do this by turning On or Off by setting the size in the configuration file or via the StorNext GUI and then restarting the FSM. Then, run your application and measure performance.

The fact that files are kept together on a stripe group for the ASR chunk size may improve performance and make stripe alignment unnecessary.

The ideal situation is for a system administrator to watch the system both with and without ASR enabled. First, performance should be monitored. Second, fragmentation can be checked. There are two kinds of fragmentation:

- Fragmentation within files.
- Free space fragmentation.

Fragments within a collection of files can be counted using snfsdefrag(1), for example, `snfsdefrag -t -r -c <directory>`. This command lists all the files and the number of extents in each file, and then the total of all regular files, extents, and extents per file.

Note: Beginning with StorNext 6, use the **`sgoffload`** command instead of the **`snfsdefrag`** command. The **`sgoffload`** command moves extents belonging to files that are currently in use (open). The **`sgoffload`** command also informs the client to suspend I/O for a time, moves the data, then informs the client to refresh the location of the data and resume I/O.

The command, `cvfsck -a -f <file system>` lists free space fragments on each stripe group by chunk size, the total number of free space fragments for each stripe group, and then the total number of stripe groups and free space fragments for the entire file system. With this tool, free space fragments can be counted before and after a workflow is run. (“Workflows” should include normal administrative cleanup and modifications which occur over time.)

Administrators are encouraged to monitor their system to see how fragmentation is occurring.

The `snfsdefrag(1)` command can be run periodically to defragment files, reducing the number of fragments in those files. This usually helps reduce free space fragmentation, too.

How ASR Works

For details on how to set the “size” and enable this feature, refer to the **`snfs_config(5)`** man page and the StorNext GUI’s online help. The man page **`snfs_config(5)`** also contains an overview of how the ASR feature works.

Because this “How ASR Works” section provides more detail, before reading this section you should already be familiar with the man page contents.

Allocation Sessions

Allocation requests (which occur whenever a file is written to an area that has no actual disk space allocated,) are grouped into sessions. A chunk of space is reserved for a session. The size of the chunk is determined using the configured size and the size of the allocation request. If the allocation size is bigger than 1MB and smaller than 1/8th the configured ASR chunk size, the ASR chunk size is rounded up to be a multiple of the initial allocation request size.

There are three session types: **small**, **medium** (directory), and **large** (file). The session type is determined by the file offset and requested allocation size on a given allocation request.

- Small sessions are for sizes (offset + allocation size) smaller than 1MB.
- Medium sessions are for sizes 1MB through 1/10th of the configured ASR size.
- Large sessions are sizes bigger than medium.

Here is another way to think of these three types: small sessions collect or organize all small files into small session chunks; medium sessions collect medium-sized files by chunks using their parent directory; and large file allocations are collected into their own chunks and are allocated independently of other files.

All sessions are client specific. Multiple writers to the same directory or large file on different clients will use different sessions. Small files from different clients use different chunks by client.

Small sessions use a smaller chunk size than the configured size. The small chunk size is determined by dividing the configured size by 32.

For example, for 128 MB the small chunk size is 4 MB, and for 1 GB the small chunk size is 32 MB. Small sessions do not round the chunk size. A file can get an allocation from a small session only if the allocation request (offset + size) is less than 1MB. When users do small I/O sizes into a file, the client buffer cache coalesces these and minimizes allocation requests. If a file is larger than 1MB and is being written through the buffer cache, it will most likely have allocation on the order of 16MB or so requests (depending on the size of the buffer cache on the client and the number of concurrent users of that buffer cache).

With NFS I/O into a StorNext client, the StorNext buffer cache is used. NFS on some operating systems breaks I/O into multiple streams per file. These will arrive on the StorNext client as disjointed random writes. These are typically allocated from the same session with ASR and are not impacted if multiple streams (other files) allocate from the same stripe group. ASR can help reduce fragmentation due to these separate NFS generated streams.

Files can start using one session type and then move to another session type. A file can start with a very small allocation (small session), become larger (medium session), and end up *reserving* the session for the file. If a file has more than 10% of a medium sized chunk, it “reserves” the remainder of the session chunk it was using for itself. After a session is reserved for a file, a new session segment will be allocated for any other medium files in that directory.

Small chunks are never reserved.

When allocating subsequent pieces for a session, they are rotated around to other stripe groups that can hold user data unless `InodeStripeWidth` (ISW) is set to 0.

Note: In StorNext, rotation is not done if `InodeStripeWidth` is set to 0.

When `InodeStripeWidth` is set, chunks are rotated in a similar fashion to `InodeStripeWidth`. The direction of rotation is determined by a combination of the session key and the index of the client in the client table. The session key is based on the inode number so odd inodes will rotate in a different direction from even inodes. Directory session keys are based on the inode number of the parent directory. For additional information about `InodeStripeWidth`, refer to the **`snfs_config(5)`** man page.

Video Frame Per File Formats

Video applications typically write one frame per file and place them in their own unique directory, and then write them from the same StorNext client. The file sizes are all greater than 1MB and smaller than 50 MB each and written/allocated in one I/O operation. Each file and write land in “medium/directory” sessions.

For this kind of workflow, ASR is the ideal method to keep “streams” (a related collection of frames in one directory) together on disk, thereby preventing checker boarding between multiple concurrent streams. In addition, when a stream is removed, the space can be returned to the free space pool in big ASR pieces, reducing free space fragmentation when compared to the default allocator.

Hotspots and Locality

Suppose a file system has four data stripe groups and an ASR size of 1 GB. If four concurrent applications writing medium-sized files in four separate directories are started, they will each start with their own 1 GB piece and most likely be on different stripe groups.

Without ASR

Without ASR, the files from the four separate applications are intermingled on disk with the files from the other applications. The default allocator does not consider the directory or application in any way when carving out space. All allocation requests are treated equally. With ASR turned off and all the applications running together, any hotspot is very short lived: the size of one allocation/file. (See the following section for more information about hotspots.)

With ASR

Now consider the 4 GB chunks for the four separate directories. As the chunks are used up, ASR allocates chunks on a new SG using rotation. Given this rotation and the timings of each application, there are times when multiple writers/segments will be on a particular stripe group together. This is considered a “hotspot,” and if the application expects more throughput than the stripe group can provide, performance will be sub par.

At read time, the checker boarding on disk from the writes (when ASR is off) can cause disk head movement, and then later the removal of one application run can also cause free space fragmentation. Since ASR collects the files together for one application, the read performance of one application's data can be significantly better since there will be little to no disk head movement.

Small Session Rationale

Small files (those less than 1 MB) are placed together in small file chunks and grouped by StorNext client ID. This was done to help use the leftover pieces from the ASR size chunks and to keep the small files away from medium files. This reduces free space fragmentation over time that would be caused by the leftover pieces. Leftover pieces occur in some rare cases, such as when there are many concurrent sessions exceeding 500 sessions.

Large File Sessions and Medium Session Reservation

When an application starts writing a very large file, it typically starts writing in some units and extending the file size. For this scenario, assume the following:

- ASR is turned on, and the configured size is 1 GB.
- The application is writing in 2 MB chunks and writing a 10 GB file.
- ISW is set to 1 GB.

On the first I/O (allocation), an ASR session is created for the directory (if one does not already exist,) and space is either stolen from an expired session or a new 1 GB piece is allocated on some stripe group.

When the file size plus the request allocation size passes 100 MB, the session will be converted from a directory session to a file-specific session and reserved for this file. When the file size surpasses the ASR size, chunks are reserved using the ISW configured size.

Returning to our example, the extents for the 10 GB file should start with a 1 GB extent (assuming the first chunk was not stolen and a partial), and the remaining extents except the last one should all be 1 GB.

The following is an example of extent layout from one process actively writing in it's own directory as described above:

```
root@per2:() -> snfsdefrag -e 10g.lmdd
10g.lmdd:
# group frbase fsbase fsend kbytes depth
0 3 0x0 0xdd4028 0xde4027 1048576 1
1 4 0x40000000 0xdd488a 0xde4889 1048576 1
2 1 0x80000000 0x10f4422 0x1104421 1048576 1
3 2 0xc0000000 0x20000 0x2ffff 1048576 1
4 3 0x100000000 0xd34028 0xd44027 1048576 1
5 4 0x140000000 0xd9488a 0xda4889 1048576 1
6 1 0x180000000 0x10c4422 0x10d4421 1048576 1
7 2 0x1c0000000 0x30000 0x3ffff 1048576 1
8 3 0x200000000 0x102c028 0x103c027 1048576 1
9 4 0x240000000 0xd6c88a 0xd7c889 1048576 1
```

Below are the extent layouts of two processes writing concurrently but in their own directory:

```
root@per2:() -> lmdd of=1d/10g bs=2m move=10g & lmdd of=2d/10g bs=2m move=10g &
```

```
[1] 27866
[2] 27867
root@per2:() -> wait
snfsdefrag -e 1d/* 2d/*
10240.00 MB in 31.30 secs, 327.14 MB/sec
[1]- Done lmdd of=1d/10g bs=2m move=10g
10240.00 MB in 31.34 secs, 326.74 MB/sec
[2]+ Done lmdd of=2d/10g bs=2m move=10g
root@per2:() ->
root@per2:() -> snfsdefrag -e 1d/* 2d/*
1d/10g:
# group frbase fsbase fsend kbytes depth
0 1 0x0 0xf3c422 0xf4c421 1048576 1
1 4 0x40000000 0xd2c88a 0xd3c889 1048576 1
2 3 0x80000000 0xfcc028 0xfdc027 1048576 1
3 2 0xc0000000 0x50000 0x5ffff 1048576 1
4 1 0x100000000 0x7a0472 0x7b0471 1048576 1
5 4 0x140000000 0xc6488a 0xc74889 1048576 1
6 3 0x180000000 0xcd4028 0xce4027 1048576 1
7 2 0x1c0000000 0x70000 0x7ffff 1048576 1
8 1 0x200000000 0x75ef02 0x76ef01 1048576 1
9 4 0x240000000 0xb9488a 0xba4889 1048576 1
2d/10g:
# group frbase fsbase fsend kbytes depth
0 2 0x0 0x40000 0x4ffff 1048576 1
1 3 0x40000000 0xffc028 0x100c027 1048576 1
2 4 0x80000000 0xca488a 0xcb4889 1048576 1
3 1 0xc0000000 0xedc422 0xeec421 1048576 1
4 2 0x100000000 0x60000 0x6ffff 1048576 1
```

```
5 3 0x140000000 0xea4028 0xeb4027 1048576 1

6 4 0x180000000 0xc2c88a 0xc3c889 1048576 1

7 1 0x1c0000000 0x77f9ba 0x78f9b9 1048576 1

8 2 0x200000000 0x80000 0x8ffff 1048576 1

9 3 0x240000000 0xbe4028 0xbf4027 1048576 1
```

Finally, consider two concurrent writers in the same directory on the same client writing 10 GB files. The files will checker board until they reach 100 MBs. After that, each file will have its own large session and the checker boarding will cease.

Below is an example of two 5 GB files written in the same directory at the same time with 2MB I/Os. The output is from the `snfsdefrag -e <file>` command.

First Example

```
# group frbase fsbase fsend kbytes depth

0 1 0x0 0x18d140 0x18d23f 4096 1

1 1 0x400000 0x18d2c0 0x18d33f 2048 1

2 1 0x600000 0x18d3c0 0x18d43f 2048 1

3 1 0x800000 0x18d4c0 0x18d53f 2048 1

4 1 0xa00000 0x18d5c0 0x18d73f 6144 1

5 1 0x1000000 0x18d7c0 0x18d83f 2048 1

6 1 0x1200000 0x18d8c0 0x18d9bf 4096 1

7 1 0x1600000 0x18dbc0 0x18dcbf 4096 1

8 1 0x1a00000 0x18dfc0 0x18e4bf 20480 1

9 1 0x2e00000 0x18e8c0 0x18e9bf 4096 1

10 1 0x3200000 0x18eac0 0x18ebbf 4096 1

11 1 0x3600000 0x18ecc0 0x18f3bf 28672 1

12 1 0x5200000 0x18f9c0 0x18fdbf 16384 1

13 1 0x6200000 0x1901c0 0x19849f 536064 1

14 3 0x26d80000 0x1414028 0x1424027 1048576 1

15 4 0x66d80000 0x150f092 0x151f091 1048576 1

16 1 0xa6d80000 0x10dc6e 0x11dc6d 1048576 1

17 3 0xe6d80000 0x1334028 0x1344027 1048576 1
```

```
18 4 0x126d80000 0x8f74fe 0x8fd99d 412160 1
```

Second Example

```
# group frbase fsbase fsend kbytes depth
0 1 0x0 0x18d0c0 0x18d13f 2048 1
1 1 0x200000 0x18d240 0x18d2bf 2048 1
2 1 0x400000 0x18d340 0x18d3bf 2048 1
3 1 0x600000 0x18d440 0x18d4bf 2048 1
4 1 0x800000 0x18d540 0x18d5bf 2048 1
5 1 0xa00000 0x18d740 0x18d7bf 2048 1
6 1 0xc00000 0x18d840 0x18d8bf 2048 1
7 1 0xe00000 0x18d9c0 0x18dbbf 8192 1
8 1 0x1600000 0x18dcc0 0x18dfbf 12288 1
9 1 0x2200000 0x18e4c0 0x18e8bf 16384 1
10 1 0x3200000 0x18e9c0 0x18eabf 4096 1
11 1 0x3600000 0x18ebc0 0x18ecbf 4096 1
12 1 0x3a00000 0x18f3c0 0x18f9bf 24576 1
13 1 0x5200000 0x18fdc0 0x1901bf 16384 1
14 4 0x6200000 0x1530772 0x1540771 1048576 1
15 3 0x46200000 0x1354028 0x1364027 1048576 1
16 1 0x86200000 0x12e726 0x13e725 1048576 1
17 4 0xc6200000 0x14ed9b2 0x14fd9b1 1048576 1
18 3 0x106200000 0x1304028 0x13127a7 948224 1
```

Note: Beginning with StorNext 6, use the **`sgoffload`** command instead of the **`snfsdefrag`** command. The **`sgoffload`** command moves extents belonging to files that are currently in use (open). The **`sgoffload`** command also informs the client to suspend I/O for a time, moves the data, then informs the client to refresh the location of the data and resume I/O.

Without ASR and with concurrent writers of big files, each file typically starts on its own stripe group. The checker boarding does not occur until there are more writers than the number of data stripe groups. However, once the checker boarding starts, it will exist all the way through the file. For example, if we have two data stripe groups and four writers, all four files would checker board until the number of writers is reduced back to two or less.

Appendix A: StorNext File System Stripe Group Affinity

This appendix describes the behavior of the stripe group affinity feature in the StorNext file system, and it discusses some common use cases.

Note: This section does not discuss file systems managed by StorNext Storage Manager. There are additional restrictions on using affinities for these managed file systems.

StorNext File System Stripe Group Affinity

This topics below describe the behavior of the stripe group affinity feature in the StorNext file system, and it discusses some common use cases.

Note: This section does not discuss file systems managed by StorNext Storage Manager. There are additional restrictions on using affinities for these managed file systems.

Definitions

The following are terms and definitions are used throughout the topics below.

Stripe Group

A *stripe group* is collection of LUNs (typically disks or arrays), across which data is striped. Each stripe group also has a number of associated attributes, including affinity and exclusivity.

Affinity

An *affinity* is used to steer the allocation of a file's data onto a set of stripe groups. Affinities are referenced by their name, which may be up to eight characters long. An affinity may be assigned to a set of stripe groups, representing a named pool of space, and to a file or directory, representing the space from which space should be allocated for that file (or files created within the directory).

Exclusivity

A stripe group which has both an affinity and the exclusive attribute can have its space allocated only by files with that affinity. Files without a matching affinity cannot allocate space from an exclusive stripe group.

- Configure Affinities below
- Allocation Strategy on page 95
- Common Use Cases on page 96

Configure Affinities

Affinities for stripe groups are defined in the file system configuration file. They can be created through the StorNext GUI or by adding one or more `Affinity` lines to a `StripeGroup` section in the configuration file. A stripe group may have multiple affinities, and an affinity may be assigned to multiple stripe groups.

Affinities for files are defined in the following ways:

- Using the **`cvmkfile`** command with the '`-k`' option.
- Using the **`snfsdefrag`** command with the '`-k`' option.

Note: Beginning with StorNext 6, use the **`sgoffload`** command instead of the **`snfsdefrag`** command. The **`sgoffload`** command moves extents belonging to files that are currently in use (open). The **`sgoffload`** command also informs the client to suspend I/O for a time, moves the data, then informs the client to refresh the location of the data and resume I/O.

- Using the **`cvaffinity`** command with the '`-s`' option
- Through inheritance from the directory in which they are created
- Through the `CvApi_SetAffinity()` function, which sets affinities programmatically
- Using the **`cvmkdir`** command with the `-k` option, a directory can be created with an affinity

Auto Affinities

Auto Affinities designate the affinity (stripe group[s]) to which allocations will be targeted for all files on the file system whose name has the specified file extension. For example, files ending with `.dpx` like, `frame1.dpx`, can be assigned an affinity by mapping dpx to an affinity in the configuration file. This assignment happens when the file is created.

See the *StorNext Online Help* for more details on how to configure **Auto Affinities** using the GUI.

Automatic Affinity Capability

At creation time, a file that matches the configuration specification (see below) will have the affinity automatically assigned. If the file would inherit a parent's affinity (its parent directory has an affinity such as from: `cvmkdir -k <affinity> <dir>`), the automatic affinity from a configuration file entry overrides it. If there are no auto affinity mappings that match for a given file name, the file's affinity is not affected (left at 0 or the inherited value). One of the configuration entries can indicate "no affinity" causing a file at creation time to have its affinity set to 0, even if it would inherit its parent's affinity.

If a file already exists, its affinity is not affected by automatic affinities.

If a file is renamed or linked to another name, its affinity is not changed. This is significant since some application create a file with a temporary name and then rename the file. The file's automatic affinity mapping occurs only at create time so the name used for these applications is the temporary name. The applications typically write; therefore space is allocated using the temporary file name so there is no suffix or extension to use. The rename usually occurs after the file is finished.

The assignment is retained in the on-disk inode and exists for the life of the file. As before this feature, you can use the **`cvaffinity(1)`** command to check or change a file's affinity.

The automatic affinity capability is implemented by creating new entries in the configuration file mapping file extensions to affinities or no affinity. The affinity in each entry must match the current affinity specification, for example, 1-8 character/ASCII string. Each mapping is followed by a list of extensions. The extensions are case insensitive.

The following examples show how to configure automatic affinities in a file system's configuration file. The examples use XML syntax. The end of this section has the XML syntax converted to the ASCII configuration format.

```
<autoAffinities>
 <autoAffinity affinity="Video">
 <extension>dpx</extension>
 <extension>mov</extension>
 </autoAffinity>
 <autoAffinity affinity="Audio">
 <extension>mp3</extension>
 <extension>wav</extension>
 </autoAffinity>
 <autoAffinity affinity="Image">
 <extension>jpg</extension>
 <extension>jpeg</extension>
```

```
  </autoAffinity>
  <autoAffinity affinity="Other">
  <extension>html</extension>
  <extension>txt</extension>
  </autoAffinity>
  <noAffinity>
  <extension>c</extension>
  <extension>sh</extension>
  <extension>o</extension>
  </noAffinity>
 </autoAffinities>
```

The affinities used must also exist in the StripeGroup sections of the same configuration file. For example, the above configuration uses the affinity **`Image`**. This affinity must be present in at least one StripeGroup.

If a filename does not match any lines in the AutoAffinities section, its affinity is 0 or the inherited affinity from its parent. An entry can be used to map such files to a specific affinity by having an empty extension field. For example:

```
  </autoAffinity>
  <autoAffinity affinity="Other">
  <extension>html</extension>
  <extension>txt</extension>
  <extension></extension>
  </autoAffinity>
```

Or:

```
  <noAffinity>
  <extension>c</extension>
  <extension>sh</extension>
  <extension>o</extension>
```

```
<extension></extension>
</autoAffinity>
```

The last case is useful to override inheritable affinities for files that do not match any extension so that they get 0 as their affinity. One could map all files thereby overriding all inheritable affinities on a system with old directory trees that have affinities.

Affinity Preference

If checked in the GUI, this permits files of a particular affinity to have their allocations placed on other available stripe groups (with non-exclusive affinities) when the stripe groups of their assigned affinity do not have sufficient space. Otherwise, allocation attempts will fail with an out-of-space error.

A new global is created to alter the behavior of affinities when all the space of a StripeGroup is used or overly fragmented not allowing certain allocations. Currently, a file with an affinity must allocate space on a StripeGroup with that affinity. If those StripeGroups are full or overly fragmented, a file allocate might fail with ENOSPC. This can be thought of as affinity enforcement. The new behavior is that an allocation with an affinity would behave the same as a file without any affinity when there is no matching StripeGroup. It can allocate space on any StripeGroup with `exclusive=false`, but only after an attempt to allocate space with the affinity fails.

The new parameter is:

```
<affinityPreference>true<affinityPreference/>
```

The default behavior is false (enforcement instead). If set to true, the new behavior is to allow allocation on StripeGroups that do not match the affinity but only when there is no space available on the "preferred" StripeGroups.

Allocation Session Reservation

Affinities create File Segregation and Grouping.

If **Allocation Session Reservation** is also enabled, files that land in the same session must have the same affinity. Creating sessions with the pair, session key and affinity, causes the functionality of grouping or segregating files within a session or folder.

For example, consider a folder **my_movies** that currently has the affinity **Video**.

It was created with the command:

```
cvmkdir -k Video my_movies
```

Consider you have two StripeGroups:

- StripeGroup number 1: affinity=Video exclusive=false
- StripeGroup number 2: No affinity.

Also consider the following:

```
<AffinityPreference>true<AffinityPreference/>
```

All the files under the directory prefer StripeGroup number 1. This includes *.dpx and *.WAV files and any others. For example, you want to separate the *.dpx files and *.WAV files. Create the mapping in the configuration file shown above.

- StripeGroup number 1: Contains the **Video** affinity. StripeGroup number 2 is modified to have the **Audio** affinity.
- StripeGroup number 2: affinity Audio exclusive=false

Consider if you create a directory with no affinity and then write *.dpx and audio files, *.WAV, in that directory.

The *.dpx files correctly land on StripeGroup number 1 and the *.WAV files go to StripeGroup number 2. Any other files also go to either StripeGroup number 2 or StripeGroup number 1. Now, consider you want the voice also on StripeGroup number 1 but in a separate location from the *.dpx files. Simply add **Audio** to the affinities on StripeGroup number 1 and remove it from StripeGroup number 2. With this addition, *.WAV files will land on StripeGroup number 1, but they will be in separate sessions from the *.dpx files. They will all prefer StripeGroup number 2 until it is full.

With **Allocation Session Reservation** enabled and the above automatic affinities, *.WAV files are grouped separately from *.dpx files since they have different affinities. If you desire just file grouping and no affinity steering to different stripe groups, simply add all the affinities to each StripeGroup that allows data and set `exclusive=false` on each data StripeGroup. The non-exclusivity is needed so that files without affinities can also use the data stripe groups.

Finally, Quantum recommends to set `AffinityPreference=true`.

Old ASCII Configuration File Example

Below are the example configuration file entries in the ASCII configuration file format.

First, Affinity Preference:

```
AffinityPreference yes
```

Next, Auto or No affinity mappings shown above in XML.

```
# Auto Affinities

```

```
[AutoAffinity Video]
Extension mov
Extension dpx

[AutoAffinity Audio]
Extension wav
Extension mp3

[AutoAffinity Image]
Extension jpeg
Extension jpg

[AutoAffinity Other]
Extension txt
Extension html

# No Affinities

[NoAffinity]
Extension o
Extension sh
Extension c
```

Empty Extension example:

```
[AutoAffinity Other]
Extension txt
Extension html
Extension
```

Or:

```
[NoAffinity]

Extension o

Extension sh

Extension c
Extension
```

Allocation Strategy

StorNext has multiple allocation strategies which can be set at the file system level. These strategies control where a new file's first blocks will be allocated. Affinities modify this behavior in two ways:

- A file with an affinity is usually allocated on a stripe group with matching affinity, unless the affinity is a preferred affinity.
- A stripe group with an affinity and the exclusive attribute is used only for allocations by files with matching affinity.

Once a file has been created, StorNext attempts to keep all of its data on the same stripe group. If there is no more space on that stripe group, data may be allocated from another stripe group. The exception to this is when `InodeStripeWidth` is set to a non-zero value. For additional information about `InodeStripeWidth`, refer to the **`snfs_confg(5)`** man page.

If the file has an affinity, only stripe groups with that affinity are considered. If all stripe groups with that affinity are full, new space may not be allocated for the file, even if other stripe groups are available. The `AffinityPreference` parameter can be used to allow file allocations for an affinity that would result in ENOSPAC to allocate on other stripe groups (using an affinity of 0). See the **`snfs_config(5)`** man page for details.

When a file system with two affinities is to be managed by the Storage Manager, the GUI forces those affinities to be named tier1 and tier2. This will cause an issue if a site has an existing unmanaged file system with two affinities with different names and wants to change that file system to be managed. There is a process for converting a file system so it can be managed but it is non-trivial and time consuming. Please contact Quantum Support if this is desired.

Note: The restriction is in the StorNext GUI because of a current system limitation where affinity names must match between one managed file system and another. If a site was upgraded from a pre-4.0 version to post-4.0, the affinity names get passed along during the upgrade. For example, if prior to StorNext 4.0 the affinity names were aff1 and aff2, the GUI would restrict any new file systems to have those affinity names as opposed to tier1 and tier2.

Common Use Cases

Here are some sample use cases in which affinities are used to maximize efficiency and operation.

Using Affinities on the HaShared File

Note: StorNext File Systems prior to version 4.3, which are configured to utilize affinities on the HaShared file system, will need to reapply affinities to directories in the HaShared file system after the upgrade to version 4.3 completes.

In some instances customers have seen improved performance of the HaShared file system by separating I/O to database and metadata archive directories through the use of multiple stripe groups and SNFS stripe group affinities. The specific improvements are dependent on the overall system utilization characteristics.

When using Affinities on the HaShared file system, the **AffinityPreference** setting should be enabled in the file system configuration. This setting will allow files to be allocated to other available stripe groups (with non-exclusive affinities) when the stripe groups of their affinity do not have sufficient space. This will prevent unnecessary out-of-space errors occurring on the HaShared file system.

The following section describes options for configuring affinities in the HaShared file system.

Note: Configuring an HaShared file system with multiple stripe groups, but no affinities will still have the advantages of distributing file I/O across each of the underlying disks. For many this simplified approach may be preferable.

Key Metadata File Locations

- `/usr/adic/HAM/shared/database/metadata archives`
- `/usr/adic/HAM/shared/TSM/internal/mapping_dir`

Key Database File Locations

- `/usr/adic/HAM/shared/mysql/db`
- `/usr/adic/HAM/shared/mysql/journal`
- `/usr/adic/HAM/shared/mysql/tmp`

For configurations utilizing two data stripe groups in the HaShared file system, key database files should be assigned to one affinity and key metadata files should be assigned the other affinity. If more than two stripe groups are configured in the HaShared file system, the individual MySQL directories can be broken out into their own stripe groups with the appropriate affinity set.

WARNING: Ensure that each stripe group is provisioned appropriately to handle the desired file type. See the `snPreInstall` script for sizing calculations. Failure to provision stripe groups appropriately could result in unexpected no-space errors.

1. Configure HaShared file system to use multiple data stripe groups.
   - If this is the initial configuration of the HaShared file system, it is recommended that an exclusive metadata & journal stripe group be created along with each of the data stripe groups. Each affinity should be assigned to the desired data stripe group prior to creating the file system.
   - If the HaShared file system already exists, any additional stripe groups should be added and the desired affinities should be added to the data stripe groups. If the MDC pair has already been converted to HA, then the MDCs must be put into Config Mode before making configuration changes to the HaShared file system.

     The number of data stripe groups should be equal to, or greater than the number of affinities desired. Do not configure any of the data affinities as exclusive.
2. If the MDC pair has not yet been converted to HA, do so at this point.
   - After the HA conversion completes, put the MDC pair into Config Mode.
3. Stop Storage Manager.
4. Assign the desired affinity to each of the directories and move data to the appropriate stripe group using the following commands:

```
# find <directory> -exec cvaffinity –s <affinity key> {} \;
```

```
# snfsdefrag –m 0 –k <affinity key> -K <affinity key> -r <directory>
```

   Note: Beginning with StorNext 6, use the **`sgoffload`** command instead of the **`snfsdefrag`** command. The **`sgoffload`** command moves extents belonging to files that are currently in use (open). The **`sgoffload`** command also informs the client to suspend I/O for a time, moves the data, then informs the client to refresh the location of the data and resume I/O.
5. Start Storage Manager.
6. Exit Config Mode.

   Note: Any active metadata archive files will not be migrated at the time these steps are run, as open files will not be relocated. Once the next StorNext backup runs, subsequent metadata archive files will be allocated on the desired stripe group.

Segregating Audio and Video Files Onto Their Own Stripe Groups

To segregate audio and video files onto their own stripe groups:

One common use case is to segregate audio and video files onto their own stripe groups. Here are the steps involved in this scenario:

- Create one or more stripe groups with an AUDIO affinity and the exclusive attribute.
- Create one or more stripe groups with a VIDEO affinity and the exclusive attribute.

- Create one or more stripe groups with no affinity (for non-audio, non-video files).
- Create a directory for audio using ‘`cvmkdir -k AUDIO audio`’.
- Create a directory for video using ‘`cvmkdir -k VIDEO video`’.

Files created within the audio directory will reside only on the AUDIO stripe group. (If this stripe group fills, no more audio files can be created.)

Files created within the video directory will reside only on the VIDEO stripe group. (If this stripe group fills, no more video files can be created.)

Reserving High-Speed Disk For Critical Files

In this use case, high-speed disk usage is reserved for and limited to only critical files. Here are the steps for this scenario:

- Create a stripe group with a FAST affinity and the exclusive attribute.
- Label the critical files or directories with the FAST affinity.

The disadvantage here is that the critical files are restricted to using only the fast disk. If the fast disk fills up, the files will not have space allocated on slow disks.

To work around this limitation, you could reserve high-speed disk for critical files but also allow them to grow onto slow disks. Here are the steps for this scenario:

- Create a stripe group with a FAST affinity and the exclusive attribute.
- Create all of the critical files, pre allocating at least one block of space, with the FAST affinity, or move them using `snfsdefrag` after ensuring the files are not empty.

> **Note:** Beginning with StorNext 6, use the **`sgoffload`** command instead of the **`snfsdefrag`** command. The **`sgoffload`** command moves extents belonging to files that are currently in use (open). The **`sgoffload`** command also informs the client to suspend I/O for a time, moves the data, then informs the client to refresh the location of the data and resume I/O.

- Remove the FAST affinity from the critical files.

Alternatively, configure the `AffinityPreference` parameter. For additional information see Configure Affinities on page 89.

Because files allocate from their existing stripe group even if they no longer have a matching affinity, the critical files will continue to grow on the FAST stripe group. Once this stripe group is full, they can allocate space from other stripe groups since they do not have an affinity.

This scenario will not work if new critical files can be created later, unless there is a process to move them to the FAST stripe group, or an affinity is set on the critical files by inheritance but removed after their first allocation (to allow them to grow onto non-FAST groups).

Appendix B: Best Practices

This appendix contains some best practice recommendations for various StorNext features which you can implement to ensure optimal performance and efficiency.

HA File System Best Practices

Note: The tuning recommendations for user file systems are not all applicable to the StorNext HA file system. This is due to the specific requirements of the HA file system. This file system is not intended for streaming I/O workloads. In addition, the HA file system utilizes a small metadata working set, relative to most user file systems.

- RAID-1 and RAID-10 are recommended to maximize data safety and data IOPS performance.
  - One or more hot spares are recommended for data protection.
  - Minimum of two dedicated physical disk drives are recommended. Additional disks are beneficial to increase IOPS performance. For instance, the M660 utilizes up to six RAID-1 pairs (for example, twelve dedicated HDD) and up to four shared hot spares for the HA file system.
- Presenting unique metadata Stripe Group for the HA file system is recommended.
- Utilizing the same RAID set for metadata and data Stripe Groups is acceptable as long as multiple dedicated disk drives are provisioned to the RAID set.
- Sharing a RAID set with other file systems is not supported
- Quantum recommends a 4KB FS Block size and 4MB Stripe Breadth settings.
  - Due to the transactional nature of the HA file system I/O profile, striping is not found to provide benefit. A stripe breadth that is too small may be detrimental to performance.
- Quantum recommends utilizing RAID read-ahead capabilities.
  - Although read-ahead is not expected to benefit database performance it can be advantageous for sequential I/O operations, such backups.

Replication Best Practices

This section describes some best practices related to using the StorNext replication feature.

Replication Copies

The replication target can keep one or more copies of data. Each copy is presented as a complete directory tree for the policy. The number of copies and placement of this directory are ultimately controlled by the replication target. However, if the target does not implement policy here, the source system may request how many copies are kept and how the directories are named.

When multiple copies are kept, the older copies and current copy share files where there are no changes. This appears as extra hard links to the files. If a file is changed on the target, it affects all copies sharing the file. If a file is changed on the replication source, older copies on the target are not affected.

The best means to list which replication copies exist on a file system is running `snpolicy -listrepcopies` command. The `rmrepcopy`, `mvrepcopy` and `exportrepcopy` options should be used to manage the copies.

Replication and Deduplication

Replication can be performed on deduplicated or non-deduplicated data. Even if the source system is running deduplication, you can still replicate non-deduplicated data to the target using the `rep_dedup=off` policy parameter.

A good example of when this makes sense is replicating into a TSM relation point which is storing to tape. If deduplicated replication is used, the store to tape requires retrieving files from the blockpool. This is much more likely to stall tape drives than streaming raw file content to tape.

The tradeoff here is that all file data will be sent over the network even if the target system has already seen it. So if the limiting resource is network bandwidth and the data is amenable to deduplication, then deduplication-enabled replication into TSM may perform better.

With deduplicated replication, the file contents are deduplicated prior to replication. There is no concept of replication using deduplicated data without deduplicating the data on the source system.

Replication data is moved via a pull model, in which the target of replication asks the source system to send it data it does not yet have. For non-deduplicated replication, this will be performed over the network UNLESS the source file system is cross mounted on the target, in which case the target will use local I/O to copy the data. The number of files actively being replicated at the same time, and the size of the buffer used for I/O in the non-deduplicated data case are controlled by the `replicate_threads` and `data_buffer_size` parameters on the target system. The default for `replicate_threads` is 8, and the default for `data_buffer_size` is 4 Mbytes.

StorNext Gateway ServerPerformance

If your configuration includes StorNext LAN Clients, Quantum strongly recommends that the machines you use for your gateway servers should not also be configured as metadata controllers. The exception to this recommendation is the StorNext M660 Metadata Appliance, which is specifically manufactured to handle this workload. Doing so may not only cause performance degradation, but also expose the virtual IPs to additional vulnerability. For best performance, machines used as gateway servers should be dedicated machines.

Replication with Multiple Physical Network Interfaces

If you want to use replication with multiple physical network interfaces, you must arrange for traffic on each interface to be routed appropriately.

In cases where both the replication source and target are plugged into the same physical Ethernet switch, you can accomplish this with VLANs.

In cases where replication is over multiple WAN links, the addresses used on the source and target replication systems must route over the appropriate WAN links in order for replication to use all the links.

Deduplication Best Practices

This section describes some best practices related to using the StorNext deduplication feature.

Deduplication and File Size

Deduplication will not be beneficial on small files, nor will it provide any benefit on files using compression techniques on the content data (such as mpeg format video). In general, deduplication is maximized for files that are 64MB and larger. Deduplication performed on files below 64MB may result in sub-optimal results.

You can filter out specific files to bypass by using the `dedup_skip` policy parameter. This parameter works the same as filename expansion in a UNIX shell.

You can also skip files according to size by using the `dedup_min_size` parameter.

Deduplication and Backups

Backup streams such as tar and NetBackup can be recognized by the deduplication algorithm if the `dedup_filter` parameter on the policy is set to `true`.

In this configuration the content of the backup image is interpreted to find the content files, and these are deduplicated individually. When this this flag is not set to `true`, the backup image is treated as raw data and the backup metadata in the file will interfere with the reduction potential of the deduplication algorithm. Recognition of a backup stream is according to its contents, not the file name.

Deduplication and File Inactivity

Deduplication is performed on a file after a period of inactivity after the file is last closed, as controlled by the `dedup_age` policy parameter. It is worth tuning this parameter if your workload has regular periods of inactivity on files before they are modified again.

Note: Making the age too small can lead to the same file being deduplicated more than once.

Deduplication and System Resources

Running deduplication is a CPU and memory-intensive operation, and the backing store for deduplicated data can see a lot of random I/O, especially when retrieving truncated files.

Consequently, plan accordingly, and do not under-resource the blockpool file system or metadata system if you are striving for optimal performance.

Deduplication Parallel Streams

The number of deduplication parallel streams running is controlled by the `ingest_threads` parameter in `/usr/cvfs/config/snpolicyd.conf`.

If you are not I/O limited and have more CPU power available, increasing the stream count from the default value of 8 streams can improve throughput.

Truncation Best Practices

This section describes some best practices related to using the StorNext truncation feature.

Deduplication and Truncation

If deduplication is run without StorNext Storage Manager also storing the file contents, then `snpolicyd` can manage file truncation. If Storage Manager is also running on a directory, it becomes the engine which removes the online copy of files.

Note: Storage Manager can retrieve deleted files from tape. With deduplication, if the primary file is removed from a directory, the deduplicated copy is no longer accessible. This is a fundamental difference between the two mechanisms (truncation and deduplication) which must be understood.

If a policy is configured not to deduplicate small files, it will automatically not truncate them. It is also possible to set an independent minimum size for files to truncate, and a stub length to leave behind when a file is truncated.

Once a file is truncated by the policy daemon, the contents must be retrieved from the deduplicated storage. This can be done by reading the file, or via the `snpolicy -retrieve` command.

Note: When using the command line to run commands, the truncation policy can potentially remove the contents again before they are used, depending on how aggressive the policy is. Unlike TSM, the whole file does not have to be retrieved before I/O can proceed. The number of parallel retrieves is governed by the `event_threads` parameter in `/usr/cvfs/config/snpolicyd.conf`.

In the case where both deduplication and tape copies of data are being made, TSM is the service which performs truncation.

Tune StorNext for Small Files

This section describes configuration settings that can lead to improved performance when using Storage Manager with workflows involving many small files. Before making these adjustments, ensure that the metadata controllers have sufficient RAM to accommodate the new values.

1. In the StorNext GUI, adjust the **Buffer Cache Size** in the file system configuration to 2 GB or larger. This setting is located in the **Performance** tab under **Advanced Parameters**.
2. Adjust the `/etc/fstab` file on the primary MDC so that the entry for the StorNext file system contains:

   ```
   threads=16,buffercachecap=4096,dircachesize=128m,cvnode_max=1048576
   ```

   For example, change:

   ```
   snfs1 /stornext/snfs1 cvfs rw 0 0
   ```

   So that it reads instead:

   ```
   snfs1 /stornext/snfs1 cvfs
   rw,threads=16,buffercachecap=4096,dircachesize=128m,cvnode_max=1048576 0 0
   ```

3. Restart StorNext to have the changes take effect.