Quantum

File System Tuning Guide

StorNext

6-67367-04 Rev A

StorNext File System Tuning Guide, 6-67367-04 Rev A, March 2012, Product of USA.

Contents

Chapter 1

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 38.

The Underlying Storage System

The performance characteristics of the underlying storage system are the most critical factors for file system 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 such as the EMC CX series and the various Engenio 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 cache, and then deferring the operation of actually writing the data to the physical disks. This results in a great performance improvement for small 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. Check with your RAID vendor to make sure that write-back caching is safe to use.

Minimal I/O latency is critically important for metadata stripe groups to achieve high metadata operations throughput. This is because metadata operations involve a very high rate of small writes to the metadata disk, so disk latency is the critical performance factor. Write-back caching can be an effective approach to minimizing I/O latency and optimizing metadata operations throughput. This is easily observed in the hourly File System Manager (FSM) statistics reports in the **cvlog** file. For example, here is a message line from the **cvlog** file:

```
PIO HiPriWr SUMMARY SnmsMetaDisk0 sysavg/350 sysmin/333
sysmax/367
```

This statistics message reports average, minimum, and maximum write latency (in microseconds) for the reporting period. If the observed average latency exceeds 500 microseconds, peak metadata operation throughput will be degraded. For example, create operations may be around 2000 per second when metadata disk latency is below 500 microseconds. However, if metadata disk latency is around 5 milliseconds, create operations per second may be degraded to 200 or worse.

Another typical write caching approach is a "write-through." 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 some cases, large write I/O operations can also benefit from caching. However, some SNFS customers observe maximum large I/O throughput by disabling caching. While this may be beneficial for special large I/O scenarios, it severely degrades small I/O performance; therefore, it is suboptimal for general-purpose file system performance.

RAID Read-Ahead Caching

RAID read-ahead caching is a very effective way to improve sequential read performance for both small (buffered) and large (DMA) I/O operations. When this setting is utilized, the RAID controller pre-fetches disk blocks for sequential read operations. Therefore, subsequent application read operations benefit from cache speed throughput, which is faster than the physical disk throughput.

This is particularly important for concurrent file streams and mixed I/O streams, because read-ahead significantly reduces disk head movement that otherwise severely impacts performance.

While read-ahead caching improves sequential read performance, it does not help highly transactional performance. Furthermore, some SNFS customers actually observe maximum large sequential read throughput by disabling caching. While disabling read-ahead is beneficial in these unusual cases, it severely degrades typical scenarios. Therefore, it is unsuitable for most environments.

RAID Level, Segment Size, and Stripe Size

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.

The best RAID level to use for high I/O throughput is usually RAID 5. The stripe size is determined by the product of the number of disks in the RAID group and the segment size. For example, a 4+1 RAID 5 group with 64K segment size results in a 256K stripe size. The stripe size is a very critical factor for write performance because I/Os smaller than the stripe size may incur a read/modify/write penalty. It is best to configure RAID 5 settings with no more than 512K stripe size to avoid the read/modify/write penalty. The read/modify/write penalty is most noticeable in the absence of “write-back” caching being performed by the RAID controller.

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.

Note: If available, RAID 3 can offer better sequential performance compared to RAID 5 for streaming-intense production. RAID 5 works well for general purpose file systems but suffers compared to RAID 3 in raw streaming throughput. RAID 3 and RAID 5 offer the same redundancy. The difference is that RAID 3 dedicates a drive to parity and RAID 5 spreads it out across all drives. If RAID 3 is supported on a RAID, it should be considered, especially for customers that do a lot of media playback from their RAID.

RAID 1 mirroring is the best RAID level for metadata and journal storage 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 ordep 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. There are few RAIDs that support RAID 10, but if IOpS is the goal, they should be seriously considered.

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**.

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.

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 Direct Memory Access (DMA) I/O Transfer section.

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 **mount command** settings **auto_dma_read_length** and **auto_dma_write_length**, described in the Mount Command Options on page 25.

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 in this document.

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 Mount Command Options on page 25.

NFS / CIFS

It is best to isolate NFS and/or CIFS traffic off of the metadata network to eliminate contention that will impact performance. For optimal performance it is necessary to use 1000BaseT instead of 100BaseT. On NFS clients, use the `vers=3`, `rsize=262144` and `wsize=262144` mount options, and use TCP mounts instead of UDP. When possible, it is also best to utilize TCP Offload capabilities as well as jumbo frames.

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-detect** is the correct setting. Some managed switches allow setting **speed/duplex** (for example `1000Mb/full`,) which disables **auto-detect** 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-detect` 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 Distributed 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** as of StorNext 4.0.

Reverse Path Lookup (RPL)

Beginning with release 4.0, StorNext includes a new feature called Reverse Path Lookup (RPL). When enabled, RPL provides the following benefits:

- StorNext replication reports containing lists of files show full pathnames instead of inode numbers.
- Operations involving reverse path lookup on managed file systems containing directories with very large file counts (>50,000) perform significantly better.

RPL is automatically enabled for file systems created using StorNext 4.0 and later. File systems created with StorNext releases prior to 4.0 do not have RPL enabled.

RPL can be turned on for these file systems by running the command `cvupdatefs -L`. However, there are possible side effects to dynamically enabling RPL, including the following:

- Extensive downtime to populate existing inodes with RPL information
- Increased metadata space usage (running `cvupdatefs -L` may result in as much as double the amount used)
- Decreased performance for certain inode-related operations

Therefore, consider carefully when deciding whether to enable RPL for file systems created with StorNext releases prior to 4.0.

SNFS and Virus Checking

Virus-checking software can severely degrade the performance of any file system, including SNFS. If you have anti-virus software running on a Windows Server 2003 or Windows XP machine, Quantum recommends configuring the software so that it does NOT check SNFS.

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. Either 100BaseT or 1000BaseT is required, but for a dedicated Metadata Network there is usually no benefit from using 1000BaseT over 100BaseT. Neither TCP offload nor are jumbo frames 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 is critical to ensure that **speed/duplex** settings are correct, as this will severely impact performance. Most of the time **auto-detect** is the correct setting. Some managed switches allow setting **speed/duplex**, such as **100Mb/full**, which disables **auto-detect** 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-detect** but the host is set to **100Mb/full**, you will observe a high error rate and extremely poor performance. On Linux the **ethtool** tool can be very useful to investigate and adjust **speed/duplex** settings.

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.
- A separate, dedicated switched Ethernet LAN is mandatory for the metadata network if 100 Mbit/s or slower networking hardware is used.
- StorNext does not support file system metadata on the same network as iSCSI, NFS, CIFS, or VLAN data when 100 Mbit/s or slower networking hardware is used.

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.

Some metadata operations such as file creation can be CPU intensive, and benefit from increased CPU power. The MDC platform is important in these scenarios because lower clock- speed CPUs such as Sparc degrade performance.

Other operations can benefit greatly from increased memory, such as directory traversal. SNFS provides three config file settings that can be used to realize performance gains from increased memory: `BufferCacheSize`, `InodeCacheSize`, and `ThreadPoolSize`.

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.

FSM Configuration File Settings

The following FSM configuration file settings are explained in greater detail in the `cvfs_config` man page. For a sample FSM configuration file, see Example FSM Configuration File on page 38.

The examples in the following sections are excerpted from the sample configuration file from Example FSM Configuration File on page 38.

Stripe Groups

Splitting apart data, metadata, and journal into separate stripe groups is usually the most important performance tactic. The **create**, **remove**, and **allocate** (e.g., write) operations are very sensitive to I/O latency of the journal stripe group. Configuring a separate stripe group for journal greatly benefits the speed of these operations because disk seek latency is minimized. 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 doesn't 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.

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 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. *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

This setting consumes up to 2X bytes of memory times the number specified. 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 16MB to 128MB for a new file system and can be increased to 256MB or 512MB as a file system grows. A higher setting is more effective if the CPU is not heavily loaded.

Example (Linux)

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

Example (Windows)

```
BufferCacheSize 32M
```

InodeCacheSize

This setting consumes about 800 to 1000 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.

Example (Linux)

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

Example (Windows)

```
InodeCacheSize 32K
```

ThreadPoolSize

This setting consumes up to 512 KB memory times the number specified. Increasing this value can improve concurrency of metadata operations. For example, if many client processes are executing concurrently, the thread pool can become exhausted by I/O wait time. Increasing the thread pool size permits hot cache operations to be processed that would otherwise be backed up behind the I/O-bound operations. There are various O/S limits to the number of threads that can cause fatal problems for the FSM daemon, so it's not a good idea to set this setting too high. A range from 32 to 128 is recommended, depending on the amount of available memory. It is recommended to size it according to the **max threads** FSM hourly statistic reported in the `cvlog` file. Optimal settings for `ThreadPoolSize` range from 32K to 128K.

Note: **ThreadPoolSize** should be adjusted until the Max Threads hourly statistic no longer tops out at one half of the value of **ThreadPoolSize**. Therefore, a setting of 32 is too small when 16 is seen in the hourly logs. This value does not have to be a power of 2, but it should be even.

Example (Linux)

```
<threadPoolSize>32</threadPoolSize>
```

Example (Windows)

```
ThreadPoolSize 32
```

FsBlockSize

The `FsBlockSize` (FSB), metadata disk size, and `JournalSize` settings all work together. For example, the `FsBlockSize` must be set correctly in order for the metadata sizing to be correct. `JournalSize` is also dependent on the `FsBlockSize`.

For `FsBlockSize` the optimal settings for both performance and space utilization are in the range of 16K or 64K.Settings greater than 64K are not recommended because performance will be adversely impacted due to inefficient metadata I/O operations. Values less than 16K are not recommended in most scenarios because startup and failover time may be adversely impacted. Setting `FsBlockSize` to higher values is important for multiterabyte file systems for optimal startup and failover time.

Note: This is particularly true for slow CPU clock speed metadata servers such as Sparc. However, values greater than 16K can severely consume metadata space in cases where the file-to-directory ratio is low (e.g., less than 100 to 1).

For metadata disk size, you must have a *minimum* of 25 GB, with more space allocated depending on the number of files per directory and the size of your file system.

The following table shows suggested `FsBlockSize` (FSB) settings and metadata disk space based on the average number of files per directory and file system size. The amount of disk space listed for metadata is *in addition* to the 25 GB minimum amount. Use this table to determine the setting for your configuration.

Average No. of Files Per Directory	File System SIze: Less Than 10TB	File System Size: 10TB or Larger
Less than 10	FSB: 16KB Metadata: 32 GB per 1M files	FSB: 64KB Metadata: 128 GB per 1M files
10-100	FSB: 16KB Metadata: 8 GB per 1M files	FSB: 64KB Metadata: 32 GB per 1M files

Average No. of Files Per Directory	File System SIze: Less Than 10TB	File System Size: 10TB or Larger
100-1000	FSB: 64KB Metadata: 8 GB per 1M files	FSB: 64KB Metadata: 8 GB per 1M files
1000 +	FSB: 64KB Metadata: 4 GB per 1M files	FSB: 64KB Metadata: 4 GB per 1M files

This setting is not adjustable after initial file system creation, so it is very important to give it careful consideration during initial configuration.

Example (Linux)

```
<config configVersion="0" name="example"
fsBlockSize="16384" journalSize="16777216">
```

Example (Windows)

```
FsBlockSize 16K
```

JournalSize

The optimal settings for `JournalSize` are in the range between 16M and 64M, depending on the `FsBlockSize`. Avoid values greater than 64M due to potentially severe impacts on startup and failover times. Values at the higher end of the 16M-64M range may improve performance of metadata operations in some cases, although at the cost of slower startup and failover time.

The following table shows recommended settings. Choose the setting that corresponds to your configuration.

FsBlockSize	JournalSize
16KB	16MB
64KB	64MB