VERITAS File System™ 3.4

Administrator's Guide

Solaris

November 2000
30-000057-399

VERITAS

Disclaimer

The information contained in this publication is subject to change without notice. VERITAS Software Corporation makes no warranty of any kind with regard to this manual, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. VERITAS Software Corporation shall not be liable for errors contained herein or for incidental or consequential damages in connection with the furnishing, performance, or use of this manual.

Copyright

Printed in the USA, November 2000.

VERITAS Software Corporation
1600 Plymouth St.
Mountain View, CA 94043
Phone 650–527–8000
Fax 650–527–8050
http://www.veritas.com

Contents

Preface

Introduction

The *VERITAS File System Administrator's Guide* provides information on the most important aspects of VERITAS File System™ (VxFS™) administration. This guide is for system administrators who configure and maintain UNIX systems with the VERITAS File System, and assumes that you have a:

- Basic understanding of system administration
- Working knowledge of the UNIX operating system
- General understanding of file systems

Organization

- Chapter 1, “The VERITAS File System,” introduces the features and characteristics of this product.
- Chapter 2, “Extent Attributes,” describes the policies associated with allocation of disk space.
- Chapter 3, “Online Backup,” describes the snapshot backup feature of VxFS.
- Chapter 4, “Performance and Tuning,” describes VxFS tools that optimize system performance. This section includes information on mount options.
- Chapter 5, “Application Interface,” describes ways to optimize an application for use with VxFS. This chapter includes details on cache advisories, extent sizes, and reservation of file space.
- Chapter 6, “Cluster File System,” describes configuring, monitoring, and other operations on VxFS file systems that are part of a cluster.
- Chapter 7, “Quotas,” describes VxFS methods to limit user access to file and data resources.
- Chapter 8, “Storage Checkpoints,” describes the VxFS replication technology that allows the quick and easy creation of resource-efficient file system backups.
- Chapter 9, “Quick I/O for Databases,” describes the VERITAS Quick I/O™ feature that treats preallocated files as raw character devices to increase performance.
- Chapter 10, “VERITAS QuickLog,” describes the optional product that improves the performance of log writes.
- Appendix A, “VERITAS File System Quick Start Reference,” provides information on common file system tasks and examples of typical VxFS operations.
- Appendix B, “Kernel Messages,” lists VxFS kernel error messages in numerical order and provides explanations and suggestions for dealing with these problems.
- Appendix C, “Disk Layout,” describes and illustrates the major components of VxFS disk layouts.
- The “Glossary” contains a list of terms and definitions relevant to VxFS.

Related Documents

The *VERITAS File System Installation and Configuration Guide* provides information on installation procedures and verification. Make sure that VxFS set is correctly installed on your system before using the *VERITAS File System Administrator's Guide.*

The online manual pages provide additional details on VxFS commands and utilities.

Conventions

Typeface	Usage	Examples
`monospace`	Computer output, files, directories, software elements such as command options, function names, and parameters	Read tunables from the `/etc/vx/tunefstab` file. See the `ls`(1) manual page for more information.
`monospace` (**`bold`**)	User input	# **`mount -F vxfs /h/filesys`**
italic	New terms, book titles, emphasis, variables replaced with a name or value	See the *User's Guide* for details. The variable *ncsize* determines the value of...

Symbol	Usage	Examples
%	C shell prompt	
$	Bourne / Korn shell prompt	
#	Superuser prompt (all shells)	
\	Continued input on the following line; you do not type this character	# **`mount -F vxfs`** \ **`/h/filesys`**
`[ ]`	In a command synopsis, brackets indicates an optional argument	`ls [ -a ]`
|	In a command synopsis, a vertical bar separates mutually exclusive arguments	`mount [ suid | nosuid ]`
blue text	Indicates an active hypertext link	In PDF and HTML files, click on links to move to the specified location

Getting Help

For assistance with any of the VERITAS products, contact VERITAS Technical Support:

- U.S. and Canadian Customers: 1-800-342-0652
- International: +1-650-527-8555
- Email: support@veritas.com

For license information:

- Phone: 1-650-527-4265
- Email: license@veritas.com
- Fax: 1-650-527-8428

For software updates:

- Phone: 1-650-527-2549
- Email: swupdate@veritas.com

For additional information about VERITAS and VERITAS products, visit the Web site at:

http://www.veritas.com

For software updates and additional technical support information, such as TechNotes, product alerts, and hardware compatibility lists, visit the VERITAS Technical Support Web site at:

http://support.veritas.com

The VERITAS File System 1

Introduction

VxFS is an extent based, intent logging file system. VxFS is geared toward UNIX environments that require high performance and availability and deal with large amounts of data.

The following topics are covered in this chapter:

VxFS Features

This chapter provides an overview of major VxFS features that are described in detail in later chapters. Basic features include:

- Extent based allocation
- Extent attributes
- Fast file system recovery
- Access control lists (ACLs)

The VxFS Advanced feature set offers these additional features:

- Online administration
- Online backup
- Enhanced application interface
- Enhanced mount options
- Improved synchronous write performance
- Support for file systems up to 1 terabyte in size
- Support for files up to 1 terabyte in size (up to two terabytes for sparse files)
- Enhanced I/O performance
- Support for quotas
- Support for improved database performance
- Support for Storage Checkpoints
- Support for cluster file systems
- Support for improved network file server (NFS) performance through use of VERITAS QuickLog™
- VxFS supports all UFS file system features and facilities except for the linking, removing, or renaming of "`.`" and "`..`" directory entries. Such operations may disrupt file system sanity.

Disk Layout Options

Three disk layout formats are available with VxFS:

Version 1

The Version 1 disk layout is the original layout used with earliest releases of VxFS.

Version 2

The Version 2 disk layout supports such features as:

- Filesets
- Dynamic inode allocation

The Version 2 layout is available with optional support for quotas.

Note The Version 3 disk layout is not supported on Solaris.

Version 4

Version 4 is the latest default disk layout with additional support for:

- Files up to 2 terabytes
- File systems up to 1 terabyte
- Access Control Lists

See Appendix C for a description of the disk layouts.

File System Performance Enhancements

UFS, the file system supplied with Solaris, uses block based allocation schemes which provide adequate random access and latency for small files but limit throughput for larger files. As a result, UFS is less than optimal for commercial environments.

VxFS addresses this file system performance issue through an alternative allocation scheme and increased user control over allocation, I/O, and caching policies. An overview of the VxFS allocation scheme is covered in the section "Extent Based Allocation" on page 5.

VxFs provides the following performance enhancements:

- Extent based allocation
- Enhanced mount options
- VERITAS Quick I/O for Databases
- Data synchronous I/O
- Direct I/O and discovered direct I/O
- Caching advisories
- Enhanced directory features
- Explicit file alignment, extent size, and preallocation controls
- Tunable I/O parameters
- Tunable indirect data extent size
- Integration with VERITAS Volume Manager™ (VxVM®)

The rest of this chapter, as well as Chapter 4, "Performance and Tuning," and Chapter 5, "Application Interface," provide more details on some of these features.

Extent Based Allocation

Disk space is allocated in 512-byte sectors to form logical blocks. VxFS supports logical block sizes of 1024, 2048, 4096, and 8192 bytes. The default block size is 1K for file systems up to 8 GB, 2K for file systems up to 16 GB, 4K for file systems up to 32 GB, and 8K for file systems beyond this size.

An *extent* is defined as one or more adjacent blocks of data within the file system. An extent is presented as an *address-length* pair, which identifies the starting block address and the length of the extent (in file system or logical blocks). VxFS allocates storage in groups of extents rather than a block at a time (as seen in UFS).

Extents allow disk I/O to take place in units of multiple blocks if storage is allocated in consecutive blocks. For sequential I/O, multiple block operations are considerably faster than block-at-a-time operations; almost all disk drives accept I/O operations of multiple blocks.

Extent allocation only slightly alters the interpretation of addressed blocks from the inode structure compared to block based inodes. The UFS inode structure contains the addresses of 12 direct blocks, one indirect block, and one double indirect block. An indirect block contains the addresses of other blocks. The UFS indirect block size is 8K and each address is 4 bytes long. UFS inodes therefore can address 12 blocks directly and up to 2048 more blocks through one indirect address.

A VxFS inode is similar to the UFS inode and references 10 direct extents, each of which are pairs of starting block addresses and lengths in blocks. The VxFS inode also points to two indirect address extents, which contain the addresses of other extents:

- The first indirect address extent is used for single indirection; each entry in the extent indicates the starting block number of an indirect data extent.
- The second indirect address extent is used for double indirection; each entry in the extent indicates the starting block number of a single indirect address extent.

Each indirect address extent is 8K long and contains 2048 entries. All indirect data extents for a file must be the same size; this size is set when the first indirect data extent is allocated and stored in the inode. Directory inodes always use an 8K indirect data extent size. By default, regular file inodes also use an 8K indirect data extent size that can be altered with `vxtunefs`; these inodes allocate the indirect data extents in clusters to simulate larger extents.

Typed Extents

Note The information in this section applies to the VxFS Version 4 disk layout.

In Version 4, VxFS introduced a new inode block map organization for indirect extents known as *typed extents*. Each entry in the block map has a typed descriptor record containing a type, offset, starting block, and number of blocks.

Indirect and data extents use this format to identify logical file offsets and physical disk locations of any given extent. The extent descriptor fields are defined as follows:

type	Uniquely identifies an extent descriptor record and defines the record's length and format.
offset	Represents the logical file offset in blocks for a given descriptor. Used to optimize lookups and eliminate hole descriptor entries.
starting block	The starting file system block of the extent.
number of blocks	The number of contiguous blocks in the extent.

- Indirect address blocks are fully typed and may have variable lengths up to a maximum and optimum size of 8K. On a fragmented file system, indirect extents may be smaller than 8K depending on space availability. VxFS always tries to obtain 8K indirect extents but resorts to smaller indirects if necessary.
- Indirect Data extents are variable in size to allow files to allocate large, contiguous extents and take full advantage of VxFS's optimized I/O.
- Holes in sparse files require no storage and are eliminated by typed records. A hole is determined by adding the offset and length of a descriptor and comparing the result with the offset of the next record.
- While there are no limits on the levels of indirection, lower levels are expected in this format since data extents have variable lengths.
- This format uses a type indicator that determines its record format and content and accommodates new requirements and functionality for future types.

The current typed format is used on regular files only when indirection is needed. Typed records are longer than the previous format and require less direct entries in the inode. Newly created files start out using the old format which allows for ten direct extents in the inode. The inode's block map is converted to the typed format when indirection is needed to offer the advantages of both formats.

Extent Attributes

VxFS allocates disk space to files in groups of one or more extents. VxFS also allows applications to control some aspects of the extent allocation. *Extent attributes* are the extent allocation policies associated with a file.

The `setext` and `getext` commands allow the administrator to set or view extent attributes associated with a file, as well as to preallocate space for a file. Refer to Chapter 2, "Extent Attributes," Chapter 5, "Application Interface," and the `setext`(1) and `getext`(1) manual pages for discussions on how to use extent attributes.

The `vxtunefs` command allows the administrator to set or view the default indirect data extent size. Refer to Chapter 4, "Performance and Tuning," and the `vxtunefs`(1M) manual page for discussions on how to use the indirect data extent size feature.

Fast File System Recovery

UFS relies on full structural verification by the `fsck` utility as the only means to recover from a system failure. For large disk configurations, this utility involves a time-consuming process of checking the entire structure, verifying that the file system is intact, and correcting any inconsistencies.

VxFS provides recovery only seconds after a system failure by utilizing a tracking feature called *intent logging*. This feature records pending changes to the file system structure in a circular *intent log*. During system failure recovery, the VxFS `fsck` utility performs an intent log replay, which scans the intent log and nullifies or completes file system operations that were active when the system failed. The file system can then be mounted without completing a full structural check of the entire file system. The intent log recovery feature is not readily apparent to the user or the system administrator except during a system failure.

Replaying the intent log may not completely recover the damaged file system structure if the disk suffers a hardware failure; such situations may require a complete system check using the `fsck` utility provided with VxFS.

Note The use of QuickLog does not affect fast file system recovery.

Online System Administration

A VxFS file system can be defragmented and resized while it remains online and accessible to users. The following sections contain detailed information about these features.

Defragmentation

Free resources are initially aligned and allocated to files in the most efficient order possible to provide optimal performance. On an active file system, the original order of free resources is lost over time as files are created, removed, and resized. The file system is spread further and further along the disk, leaving unused gaps or *fragments* between areas that are in use. This process is also known as *fragmentation* and leads to degraded performance because the file system has fewer options when assigning a file to an extent (a group of contiguous data blocks).

UFS uses the concept of *cylinder groups* to limit fragmentation. Cylinder groups are self-contained sections of a file system that indicate free inodes and data blocks. Allocation strategies in UFS attempt to place inodes and data blocks in close proximity. This reduces fragmentation but does not eliminate it.

VxFS provides the online administration utility `fsadm` to resolve the problem of fragmentation. The `fsadm` utility defragments a mounted file system by:

- Removing unused space from directories.
- Making all small files contiguous.
- Consolidating free blocks for file system use.

This utility can run on demand and should be scheduled regularly as a `cron` job.

Resizing

A file system is assigned a specific size as soon as it is created; the file system may become too small or too large as changes in file system usage take place over time.

UFS traditionally offers three solutions to address the lack of space in a small file system:

- Move some users to a different file system
- Move a subdirectory of the file system to a new file system
- Copy the entire file system to a larger file system

Most large file systems with too much space try to reclaim the unused space by off-loading the contents of the file system and rebuilding it to a preferable size. UFS requires unmounting the file system and blocking user access during the modification.

The VxFS utility `fsadm` can expand or shrink a file system without unmounting the file system or interrupting user productivity. However, to expand a file system, the underlying device on which it is mounted must be expandable.

VxVM facilitates expansion using virtual disks that can be increased in size while in use. The VxFS and VxVM packages complement each other to provide online expansion capability. Refer to the *VERITAS Volume Manager Administrator's Guide* for additional information about such capabilities.

Online Backup

VxFS provides a method of online backup of data using the *snapshot* feature. An image of a mounted file system instantly becomes an exact read-only copy of the file system at a certain point in time. The original file system is *snapped* while the copy is called the *snapshot*.

When changes are made to the snapped file system, the old data is first copied to the snapshot. When the snapshot is read, data that has not changed is read from the snapped file system. Changed data is read directly from the snapshot.

Backups require one of the following methods:

- Copying selected files from the snapshot file system (using `find` and `cpio`)
- Backing up the entire file system (using `fscat`)
- Initiating a full or incremental backup (using `vxdump`)

For detailed information about performing online backups, see Chapter 3.

Application Interface

VxFS conforms to the System V Interface Definition (SVID) requirements and supports user access through the Network File System (NFS). Applications that require performance features not available with other file systems can take advantage of VxFS enhancements that are introduced in this section and covered in detail in Chapter 5, "Application Interface."

Application Transparency

In most cases, any application designed to run on UFS should run transparently on VxFS.

Expanded Application Facilities

VxFS provides some facilities frequently associated with commercial applications that make it possible to:

- Preallocate space for a file
- Specify a fixed extent size for a file
- Bypass the system buffer cache for file I/O
- Specify the expected access pattern for a file

Since these facilities are provided using VxFS-specific ioctl system calls, most existing UNIX system applications do not use these facilities. The `cp`, `cpio`, and `mv` utilities use these facilities to preserve extent attributes and allocate space more efficiently. The current attributes of a file can be listed using the `getext` command or `ls` command. The facilities can also improve performance for custom applications. For portability reasons, these applications should check what file system type they are using before using these interfaces.

Extended mount Options

The VxFS file system supports extended `mount` options to specify:

- Enhanced data integrity modes
- Enhanced performance modes
- Temporary file system modes
- Improved synchronous writes

See Chapter 4, "Performance and Tuning," and the `mount_vxfs`(1M) manual page for details on the VxFS `mount` options.

Enhanced Data Integrity Modes

Note Performance trade-offs are associated with these `mount` options.

UFS is "buffered" in the sense that resources are allocated to files and data is written asynchronously to files. In general, the buffering schemes provide better performance without compromising data integrity.

If a system failure occurs during space allocation for a file, uninitialized data or data from another file may appear in the extended file after reboot. Data written shortly before the system failure may also be lost.

Using blkclear for Data Integrity

In environments where performance is more important than absolute data integrity, the preceding situation is not of great concern. However, VxFS supports environments that emphasize data integrity by providing the `mount -o blkclear` option that ensures uninitialized data does not appear in a file.

Using closesync for Data Integrity

VxFS provides the `mount -o mincache=closesync` option, which is useful in desktop environments with users who are likely to shut off the power on machines without halting them first. In `closesync` mode, only files that are written during the system crash or shutdown can lose data. Any changes to a file are flushed to disk when the file is closed.

Enhanced Performance Mode

UFS is asynchronous in the sense that structural changes to the file system are not immediately written to disk. File systems are designed this way to provide better performance. However, recent changes to the file system may be lost if a system failure occurs. More specifically, attribute changes to files and recently created files may disappear.

The default logging mode provided by VxFS (`mount -o log`) guarantees that all structural changes to the file system are logged to disk before the system call returns to the application. If a system failure occurs, `fsck` replays any recent changes to preserve all metadata. Recent file data may be lost unless a request was made to `sync` it to disk.

Using delaylog for Enhanced Performance

VxFS provides the `mount -o delaylog` option which increases performance by delaying the logging of some structural changes. However, recent changes may be lost during a system failure. This option provides at least the same level of data accuracy that traditional UNIX file systems provide for system failures, along with fast file system recovery.

Using qlog for Enhanced Performance

VxFS provides the `mount -o qlog=` option to activate QuickLog™ for a file system. QuickLog increases VxFS performance by exporting the file system log to a separate physical volume. This eliminates the disk seek time between the VxFS data and log areas on disk and increases the performance of synchronous log writes. See Chapter 10, “VERITAS QuickLog,” for details.

Temporary File System Modes

On most UNIX systems, temporary file system directories (such as `/tmp` and `/usr/tmp`) often hold files that do not need to be retained when the system reboots. The underlying file system does not need to maintain a high degree of structural integrity for these temporary directories.

Using tmplog For Temporary File Systems

VxFS provides a `mount -o tmplog` option which allows the user to achieve higher performance on temporary file systems by delaying the logging of most operations.

Improved Synchronous Writes

VxFS provides superior performance for synchronous write applications.

The default `datainlog` option to `mount` greatly improves the performance of small synchronous writes.

The `convosync=dsync` option to `mount` improves the performance of applications that require synchronous data writes but not synchronous inode time updates.

Note The use of the `convosync=dsync` option violates POSIX semantics.

Enhanced I/O Performance

VxFS provides enhanced I/O performance by applying an aggressive I/O clustering policy, integrating with VxVM, and allowing the system administrator to set application specific parameters on a per-file system basis.

Enhanced I/O Clustering

I/O clustering is a technique of grouping multiple I/O operations together for improved performance. VxFS I/O policies provide more aggressive clustering processes than other file systems and offer higher I/O throughput when using large files; the resulting performance is comparable to that provided by raw disk.

VxVM Integration

VxFS interfaces with VxVM to determine the I/O characteristics of the underlying volume and perform I/O accordingly. VxFS also uses this information when using `mkfs` to perform proper allocation unit alignments for efficient I/O operations from the kernel.

As part of VxFS/VxVM integration, VxVM exports a set of I/O parameters to achieve better I/O performance. This interface can enhance performance for different volume configurations such as RAID-5, striped, and mirrored volumes. Full stripe writes are important in a RAID-5 volume for strong I/O performance. VxFS uses these parameters to issue appropriate I/O requests to VxVM.

Application-Specific Parameters

System administrators can also set application specific parameters on a per-file system basis to improve I/O performance.

- Default Indirect Extent Size

  On disk layout Versions 1 and 2, this value can be set up to apply to all the indirect extents, provided a fixed extent size is not already set and the file does not already have indirect extents. The Version 4 disk layout uses typed extents which have variable sized indirects.

- Discovered Direct I/O

  All sizes above this value would be performed as direct I/O.

- Maximum Direct I/O Size

  This value defines the maximum size of a single direct I/O.

For a discussion on VxVM integration and performance benefits, refer to Chapter 4, "Performance and Tuning," Chapter 5, "Application Interface," and the `vxtunefs`(1M) and `tunefstab`(1M) manual pages.

Quotas

VxFS supports quotas, which allocate per-user quotas and limit the use of two principal resources: files and data blocks. The system administrator can assign quotas for each of these resources. Each quota consists of two limits for each resource:

- The *hard limit* represents an absolute limit on data blocks or files. The user may never exceed the hard limit under any circumstances.
- The *soft limit* is lower than the hard limit and may be exceeded for a limited amount of time. This allows users to temporarily exceed limits as long as they fall under those limits before the allotted time expires.

The system administrator is responsible for assigning hard and soft limits to users. See "Quota Limits" on page 76 for more information.

Access Control Lists

An Access Control List (ACL) stores a series of entries that identify specific users or groups and their access privileges for a directory or file. A file may have its own ACL or may share an ACL with other files. ACLs have the advantage of specifying detailed access permissions for multiple users and groups. Refer to the `getfacl`(1) and `setfacl`(1) manual pages for information on viewing and setting ACLs.

Support for Large Files

VxFS can now support files up to one terabyte in size because file system structures are no longer in fixed locations (see Appendix C, "Disk Layout"). See "largefiles | nolargefiles" on page 37 for information on how to create, mount, and manage file systems containing large files.

Note Some applications and utilities may not work on large files.

Cluster File Systems

Clustered file systems are an extension of VxFS that support concurrent direct media access from multiple systems. CFS employs a master/slave protocol. All cluster file systems can read file data directly from a shared disk. In addition, all systems can write "in-place" file data. Operations that require changes to file system metadata, such as allocation, creation, and deletion, can only be performed by the single primary file system node. To maintain file system consistency, secondary nodes must send messages to the primary, and the primary will perform the operations. Chapter 6, "Cluster File System," for details on cluster file systems.

Storage Checkpoints

To increase availability, recoverability, and performance, the VERITAS File System offers on-disk and online backup and restore capabilities that facilitate frequent and efficient backup strategies. Backup and restore applications can leverage the VERITAS *Storage Checkpoint*, a disk and I/O efficient copying technology for creating periodic *frozen images* of a file system. Storage Checkpoints present a view of a file system at a point in time, and subsequently identifies and maintains only changed file system blocks. Instead of using a disk-based mirroring method, Storage Checkpoints save disk space and significantly reduce I/O overhead by using the free space pool available to a file system. See Chapter 8, "Storage Checkpoints," for details on using the Storage Checkpoint feature.

Support for Databases

Databases are usually created on file systems to simplify backup, copying, and moving tasks and are slower compared to databases on raw disks.

Using the VERITAS Quick I/O for Databases feature with VxFS lets systems retain the benefits of having a database on a file system without sacrificing performance. VERITAS Quick I/O creates regular, preallocated files to use as character devices. Databases can be created on the character devices to achieve the same performance as databases created on raw disks.

Treating regular VxFS files as raw devices has the following advantages for databases:

- Commercial database servers such as Oracle Server can issue kernel supported asynchronous I/O calls on these pseudo devices but not on regular files.
- `read`() and `write`() system calls issued by the database server can avoid the acquisition and release of `read/write` locks inside the kernel that take place on regular files.
- VxFS can avoid double buffering of data already buffered by the database server. This ability frees up resources for other purposes and results in better performance.
- Since I/O to these devices bypasses the system buffer cache, VxFS saves on the cost of copying data between user space and kernel space when data is read from or written to a regular file. This process significantly reduces CPU time per I/O transaction compared to that of buffered I/O.

See Chapter 9, "Quick I/O for Databases," for details on VxFS database support.

VERITAS QuickLog

Without QuickLog™, the intent log information for VxFS is usually stored near the beginning of the file system volume and log data is written sequentially. From a volume perspective, writing to the log appears to be random because other disk operations (inode, data) are issued on the same volume, causing the disk head to seek between the log and file system data areas. QuickLog moves the VxFS log from the physical volume containing the file system onto a separate physical volume. This eliminates the seek time between the log and file system data and improves performance.

Extent Attributes 2

Introduction

The VERITAS File System (VxFS) allocates disk space to files in groups of one or more adjacent blocks called *extents*. VxFS defines an application interface that allows programs to control various aspects of the extent allocation for a given file (see Chapter 5, "Application Interface"). The extent allocation policies associated with a file are referred to as *extent attributes*.

The VxFS `getext` and `setext` commands let you view or manipulate file extent attributes. In addition, the `vxdump`, `vxrestore`, `mv_vxfs`, `cp_vxfs`, and `cpio_vxfs` commands preserve extent attributes when a file is backed up, moved, copied, or archived.

The following topics are covered in this chapter:

Attribute Specifics

The two basic extent attributes associated with a file are its *reservation* and its *fixed extent size*. You can preallocate space to the file by manipulating a file's reservation, or override the default allocation policy of the file system by setting a fixed extent size.

Other policies determine the way these attributes are expressed during the allocation process. You can specify that:

- The space reserved for a file must be contiguous
- No allocations are made for a file beyond the current reservation
- An unused reservation is released when the file is closed
- Space is allocated, but no reservation is assigned
- The file size is changed to immediately incorporate the allocated space

Some of the extent attributes are persistent and become part of the on-disk information about the file, while other attributes are temporary and are lost after the file is closed or the system is rebooted. The persistent attributes are similar to the file's permissions and are written in the inode for the file. When a file is copied, moved, or archived, only the persistent attributes of the source file are preserved in the new file (see "Other Controls" on page 20 for more information).

In general, the user will only set extent attributes for reservation. Many of the attributes are designed for applications that are tuned to a particular pattern of I/O or disk alignment (see the `mkfs_vxfs`(1M) manual page and Chapter 5, "Application Interface," for more information).

Reservation: Preallocating Space to a File

VxFS makes it possible to preallocate space to a file at the time of the request rather than when data is written into the file. This space cannot be allocated to other files in the file system. VxFS prevents any unexpected out-of-space condition on the file system by ensuring that a file's required space will be associated with the file before it is required.

Persistent reservation is not released when a file is truncated. The reservation must be cleared or the file must be removed to free reserved space.

Fixed Extent Size

The VxFS default allocation policy uses a variety of heuristics to determine how to make an allocation to a file when a write requires additional space. The policy attempts to balance the two goals of optimum I/O performance through large allocations and minimal file system fragmentation through allocation from space available in the file system that best fits the data.

Setting a fixed extent size overrides the default allocation policies for a file and always serves as a persistent attribute. Be careful to choose an extent size appropriate to the application when using fixed extents. An advantage of VxFS's extent based allocation policies is that they rarely use indirect blocks compared to block based file systems; VxFS eliminates many instances of disk access that stem from indirect references. However, a small extent size can eliminate this advantage.

Files with aggressive allocation sizes tend to be more contiguous and have better I/O characteristics. However, the overall performance of the file system degrades because the unused space fragments free space by breaking large extents into smaller pieces. By erring on the side of minimizing fragmentation for the file system, files may become so non-contiguous that their I/O characteristics would degrade.

Fixed extent sizes are particularly appropriate in the following situations:

- If a file is large and sparse and its write size is fixed, a fixed extent size that is a multiple of the write size can minimize space wasted by blocks that do not contain user data as a result of misalignment of write and extent sizes. (The default extent size for a sparse file is 8K.)
- If a file is large and contiguous, a large fixed extent size can minimize the number of extents in the file.

Custom applications may also use fixed extent sizes for specific reasons, such as the need to align extents to cylinder or striping boundaries on disk.

Other Controls

The auxiliary controls on extent attributes determine:

- Whether allocations are aligned
- Whether allocations are contiguous
- Whether the file can be written beyond its reservation
- Whether an unused reservation is released when the file is closed
- Whether the reservation is a persistent attribute of the file
- When the space reserved for a file will actually become part of the file

Alignment

Specific alignment restrictions coordinate a file's allocations with a particular I/O pattern or disk alignment (see the `mkfs_vxfs`(1M) manual page and Chapter 5, "Application Interface," for details). Alignment can only be specified if a fixed extent size has also been set. Setting alignment restrictions on allocations is best left to well designed applications.

Contiguity

A reservation request can specify that its allocation remain contiguous (all one extent). Maximum contiguity of a file optimizes its I/O characteristics.

Note Fixed extent sizes or alignment will cause the file system to return an error message reporting insufficient space if no suitably sized (or aligned) extent is available. This may happen even if the file system has plenty of free space and the fixed extent size is large.

Write Operations Beyond Reservation

A reservation request can specify that no allocations can take place after a write operation fills up the last available block in the reservation. This specification can be used in a similar way to `ulimit` to prevent a file's uncontrolled growth.

Reservation Trimming

A reservation request can specify that any unused reservation be released when the file is closed. The file is not completely closed until all processes open against the file have closed it.

Reservation Persistence

A reservation request can ensure the reservation does not become a persistent attribute of the file. Unused reservation is discarded when the file is closed.

Including Reservation in the File

A reservation request can make sure the size of the file is adjusted to include the reservation. Normally, the space of the reservation is not included in the file until an extending write operation requires it. A reservation that immediately changes the file size can generate large temporary files. Unlike a `ftruncate` operation that increases the size of a file, this type of reservation does not perform zeroing of the blocks included in the file and limits this facility to users with appropriate privileges. The data that appears in the file may have been previously contained in another file.

Commands Related to Extent Attributes

The VxFS commands for manipulating extent attributes are `setext` and `getext`; they allow the user to set up files with a given set of extent attributes or view any attributes that are already associated with a file. See the `getext`(1) and `setext`(1) manual pages for details on using these commands.

The VxFS-specific commands `vxdump`, `vxrestore`, `mv_vxfs`, `cp_vxfs`, and `cpio_vxfs` preserve extent attributes when backing up, restoring, moving, or copying files. Make sure to modify your PATH when using the VxFS versions of `mv`, `cp`, and `cpio`.

Most of these commands include a command line option (`-e`) for maintaining extent attributes on files. This option specifies dealing with a VxFS file that has extent attribute information including reserved space, a fixed extent size, and extent alignment. The extent attribute information may be lost if the destination file system does not support extent attributes, has a different block size than the source file system, or lacks free extents appropriate to satisfy the extent attribute requirements.

The `-e` option takes any of the following keywords as an argument:

`warn`	Issues a warning message if extent attribute information cannot be maintained (the default)
`force`	Fails the copy if extent attribute information cannot be maintained
`ignore`	Ignores extent attribute information entirely

The commands that move, copy, or archive files (`mv_vxfs`, `cp_vxfs` and `cpio_vxfs`) use the `-e` option with arguments of `ignore`, `warn`, or `force`.

For example, the `mv_vxfs` command could be used with the `-e` option to produce the following results:

- The `ignore` keyword loses any extent attributes for files.
- The `warn` keyword issues a warning if extent attributes for a file cannot be preserved. Such a situation may take place if the file is moved into a non-VxFS file system; the file would ultimately be moved while the extent attributes would be lost.
- The `force` keyword issues an error if attributes are lost and the file is not relocated.

The `ls` command has an `-e` option, which prints the extent attributes of the file.

Failure to Preserve Extent Attributes

Whenever a file is copied, moved, or archived using commands that preserve extent attributes, there is nevertheless the possibility of losing the attributes. Such a failure might occur for three reasons:

- The file system receiving a copied, moved, or restored file from an archive is not a VxFS type. Since other file system types do not support the extent attributes of the VxFS file system, the attributes of the source file are lost during the migration.
- The file system receiving a copied, moved, or restored file is a VxFS type but does not have enough free space to satisfy the extent attributes. For example, consider a 50K file and a reservation of 1 MB. If the target file system has 500K free, it could easily hold the file but fail to satisfy the reservation.
- The file system receiving a copied, moved, or restored file from an archive is a VxFS type but the different block sizes of the source and target file system make extent attributes impossible to maintain. For example, consider a source file system of block size 1024, a target file system of block size 4096, and a file that has a fixed extent size of 3 blocks (3072 bytes). This fixed extent size adapts to the source file system but cannot translate onto the target file system.

  The same source and target file systems in the preceding example with a file carrying a fixed extent size of 4 could preserve the attribute; a 4 block (4096 byte) extent on the source file system would translate into a 1 block extent on the target.

  On a system with mixed block sizes, a copy, move, or restoration operation may or may not succeed in preserving attributes. It is recommended that the same block size be used for all file systems on a given system.

Online Backup 3

Introduction

This chapter describes the online backup facility provided with the VERITAS File System (VxFS). The snapshot feature of VxFS can be used to create a snapshot image of a mounted file system, which becomes a duplicate read-only copy of the mounted file system.

The following topics are covered in this chapter:

Snapshot File Systems

The VxFS file system provides a mechanism for taking snapshot images of mounted file systems, which is useful for making backups. The *snapshot file system* is an exact image of the original file system, which is referred to as the *snapped file system*. The snapshot is a consistent view of the file system "snapped" at the point in time the snapshot is made. Selected files can be backed up from the snapshot (using standard utilities such as `cpio` or `cp`) or the entire file system image can be backed up (using the `vxdump` or `fscat` utilities).

The `mount` command is used to create a snapshot file system; there is no `mkfs` step involved. A snapshot file system is always read-only and exists only as long as it and the file system that has been snapped are mounted. A snapped file system cannot be unmounted until any corresponding snapshots are first unmounted. A snapshot file system ceases to exist when unmounted. While it is possible to have multiple snapshots of a file system made at different times, it is not possible to make a snapshot of a snapshot.

This chapter describes the creation of snapshot file systems and gives some examples of backing up all or part of a file system using the snapshot mechanism.

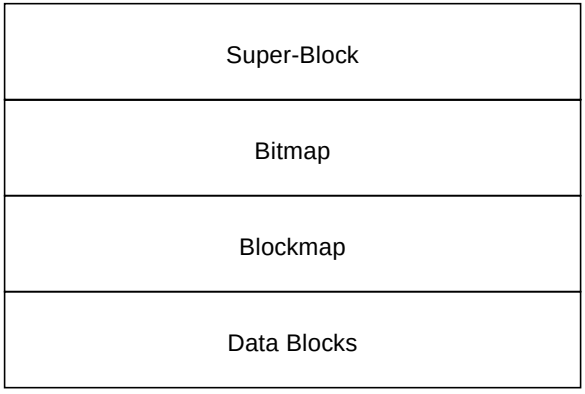
Snapshot File System Disk Structure

A snapshot file system consists of:

- A super-block
- A bitmap
- A blockmap
- Data blocks copied from the snapped file system

Figure 1 shows the disk structure of a snapshot file system.

Figure 1. The Snapshot Disk Structure



The super-block is similar to the super-block of a standard VxFS file system, however, the magic number is different and many of the fields are meaningless.

Immediately following the super-block is the bitmap. The bitmap contains one bit for every block on the snapped file system. Initially, all bitmap entries are zero. A set bit indicates that the appropriate block was copied from the snapped file system to the snapshot. In this case, the appropriate position in the blockmap will reference the copied block,

Following the bitmap is the blockmap. It contains one entry for each block on the snapped file system. Initially, all entries are zero. When a block is copied from the snapped file system to the snapshot, the appropriate entry in the blockmap is changed to contain the block number on the snapshot file system that holds the data from the snapped file system.

The data blocks used by the snapshot file system are located after the blockmap. These are filled by any data copied from the snapped file system, starting from the front of the data block area.

How a Snapshot File System Works

A snapshot file system is created by mounting an empty disk slice as a snapshot of a currently mounted file system. The bitmap, blockmap and super-block are initialized and then the currently mounted file system is frozen (see "Freeze and Thaw" on page 58, for a description of the VX_FREEZE ioctl). Once the file system to be snapped is frozen, the snapshot is enabled and mounted and the snapped file system is thawed. The snapshot appears as an exact image of the snapped file system at the time the snapshot was made.

Initially, the snapshot file system satisfies read requests by simply finding the data on the snapped file system and returning it to the requesting process. When an inode update or a write changes the data in block *n* of the snapped file system, the old data is first read and copied to the snapshot before the snapped file system is updated. The bitmap entry for block *n* is changed from 0 to 1 (indicating that the data for block *n* can be found on the snapped file system). The blockmap entry for block *n* is changed from 0 to the block number on the snapshot file system containing the old data.

A subsequent read request for block *n* on the snapshot file system will be satisfied by checking the bitmap entry for block *n* and reading the data from the indicated block on the snapshot file system, rather than from block *n* on the snapped file system. Subsequent writes to block *n* on the snapped file system do not result in additional copies to the snapshot file system, since the old data only needs to be saved once.

All updates to the snapped file system for inodes, directories, data in files, extent maps, etc., are handled in this fashion so that the snapshot can present a consistent view of all file system structures for the snapped file system for the time when the snapshot was created. As data blocks are changed on the snapped file system, the snapshot will gradually fill with data copied from the snapped file system.

The amount of disk space required for the snapshot depends on the rate of change of the snapped file system and the amount of time the snapshot is maintained. In the worst case, the snapped file system is completely full and every file is removed and rewritten. The snapshot file system would need enough blocks to hold a copy of every block on the snapped file system, plus additional blocks for the data structures that make up the snapshot file system. This is approximately 101 percent of the size of the snapped file system. Normally, most file systems do not undergo changes at this extreme rate. During periods of low activity, the snapshot should only require two to six percent of the blocks of the snapped file system. During periods of high activity, the snapshot might require 15 percent of the blocks of the snapped file system. These percentages tend to be lower for larger file systems and higher for smaller ones.

Note If a snapshot file system runs out of space for changed data blocks, it is disabled and all further access to it fails. This does not affect the snapped file system.

Using a Snapshot File System for Backup

After a snapshot file system is created, it can be used to perform a consistent backup of the snapped file system. Backup programs that function using the standard file system tree (such as `cpio`) can be used without modification on a snapshot file system, since the snapshot presents the same data as the snapped file system. Backup programs that access the disk structures of a VxFS file system (such as `vxdump`) require some modifications to deal with a snapshot file system. The VxFS utilities understand snapshot file systems and make suitable modifications in their behavior so that their operation on a snapshot file system is indistinguishable from that on a standard file system.

Other backup programs that normally read the raw disk image cannot work on snapshots without modification. These programs can use the `fscat` command to obtain a raw image of the entire file system identical to that which would have been obtained by a `dd` of the disk device containing the snapped file system at the exact moment the snapshot was created. The `snapread` ioctl takes arguments similar to those of the `read` system call and returns the same results as would have been obtained by performing a read on the disk device containing the snapped file system at the exact time the snapshot was created. In both cases, however, the snapshot file system provides a consistent image of the snapped file system with all activity complete; it is an instantaneous read of the entire file system. This is a marked contrast to the results that would be obtained by a `dd` or `read` of the disk device of an active file system.

If a complete backup of a snapshot file system is made through a utility such as `vxdump` and is later restored, it will be necessary to `fsck` the restored file system because the snapshot file system is only consistent and not clean. The file system may have some extended inode operations that must be completed, though there should be no other changes. Since the snapshot file system is not writable, it cannot be fully checked. However, the `fsck -n` command can be used to report any inconsistencies.

Creating a Snapshot File System

A snapshot file system is created by using the `-o snapof=` option of the `mount` command. The `-o snapsize=` option may also be required if the device being mounted does not identify the device size in its disk label, or if a size smaller than the entire device is desired. Use the following syntax to create a snapshot file system:

```
mount -F vxfs -o snapof=special,snapsize=snapshot_size \
snapshot_special snapshot_mount_point
```

The snapshot file system must be created large enough to hold any blocks on the snapped file system that may be written to while the snapshot file system exists. If a snapshot file system runs out of blocks to hold copied data, it will be disabled and all further access to the snapshot file system will fail.

During a period of low activity when the system is relatively inactive (for example, on nights and weekends), the snapshot only needs to contain two to six percent of the blocks of the snapped file system. During a period of higher activity, the snapshot of an "average" file system might require 15 percent of the blocks of the snapped file system, though most file systems do not experience this much turnover of data over an entire day. These percentages tend to be lower for larger file systems and higher for smaller ones. You should manage the blocks allocated to the snapshot based on such things as file system usage and duration of backups.

Note A snapshot file system ceases to exist when unmounted. If remounted, it will be a fresh snapshot of the snapped file system. A snapshot file system must be unmounted before the corresponding snapped file system can be unmounted. Neither `fuser` nor `mount` indicate that a snapped file system cannot be unmounted because a snapshot of it exists.

Caution Any existing data on the disk used for the snapshot is overwritten and lost.

Making a Backup

Here are some typical examples of making a backup of a 300,000 block file system named /home using a snapshot file system on /dev/dsk/c0t1d0s1 with a snapshot mount point of /backup/home:

- To back up files changed within the last week using cpio:

```
# mount -F vxfs -o snapof=/home,snapsize=100000 \
/dev/dsk/c0t1d0s1 /backup/home
# cd /backup
# find home -ctime -7 -depth -print | cpio -oc > /dev/rmt/c0s0
# umount /backup/home
```

- To do a full backup of /home, which exists on disk /dev/dsk/c0t0d0s7, and use dd to control blocking of output onto tape device using vxdump:

```
# vxdump f - /dev/rdsk/c0t0d0s7 | dd bs=128k > /dev/rmt/c0s0
```

- To do a level 3 backup of /dev/dsk/c0t0d0s7 and collect those files that have changed in the current directory:

```
# vxdump 3f - /dev/rdsk/c0t0d0s7 | vxrestore -xf -
```

- To do a full backup of a snapshot file system:

```
# mount -o snapof=/home,snapsize=100000 \
/dev/dsk/c0t1d0s1 /backup/home
# vxdump f - /dev/rdsk/c0t1d0s1 | dd bs=128k > /dev/rmt/c0s0
```

The vxdump utility ascertains whether /dev/rdsk/c0t1d0s1 is a snapshot mounted as /backup/home and do the appropriate work to get the snapshot data through the mount point.

Performance of Snapshot File Systems

Snapshot file systems maximize the performance of the snapshot at the expense of writes to the snapped file system. Reads from a snapshot file system typically perform at nearly the throughput rates of reads from a standard VxFS file system, allowing backups to proceed at the full speed of the standard file system.

The performance of reads from the snapped file system should not be affected. Writes to the snapped file system, however, typically average two to three times as long as without a snapshot, since the initial write to a data block now requires a read of the old data, a write of the data to the snapshot, and finally the write of the new data to the snapped file system. If multiple snapshots of the same snapped file system exist, writes will be even slower. Only the initial write to a block suffers this penalty, however, so operations like writes to the intent log or inode updates proceed at normal speed after the initial write.

Reads from the snapshot file system are impacted if the snapped file system is busy, since the snapshot reads are slowed by all of the disk I/O associated with the snapped file system.

The overall impact of the snapshot is dependent on the read to write ratio of an application and the mixing of the I/O operations. As an example, Oracle running an OLTP workload on a snapped file system was measured at about 15 to 20 percent slower than a file system that was not snapped.

Performance and Tuning 4

Introduction

For any file system, the ability to provide peak performance is important. Adjusting the available VERITAS File System (VxFS) options provides a way to optimize system performance. This chapter describes tools that an administrator can use to optimize VxFS. For information on optimizing an application for use with VxFS, see Chapter 5, "Application Interface."

The following topics are covered in this chapter:

- Choosing a Block Size
- Choosing an Intent Log Size
- Choosing Mount Options
  - log
  - delaylog
  - tmplog
  - nodatainlog
  - nodatainlog
  - blkclear
  - mincache
  - convosync
  - qlog
  - Combining mount Command Options
- Kernel Tunables
  - Internal Inode Table Size
  - VxVM Maximum I/O Size

Choosing a Block Size

You specify the block size when a file system is created; it cannot be changed later. With UFS, a file system defaults to a block size of 8K with a 1K fragment size. This means that space is allocated to small files (up to 8K) in 1K increments. Allocations for larger files are done in 8K increments except for the last block, which may be a fragment. Because many files are small, the fragment facility saves a large amount of space compared to allocating space 8K at a time.

The unit of allocation in VxFS is a block. There are no fragments because storage is allocated in extents that consist of one or more blocks. The smallest block size available is 1K, which is also the default block size for VxFS file systems created on the system.

Choose a block size based on the type of application being run. For example, if there are many small files, a 1K block size may save space. For large file systems, with relatively few files, a larger block size is more appropriate. The trade-off of specifying larger block sizes is a decrease in the amount of space used to hold the free extent bitmaps for each allocation unit, an increase in the maximum extent size, and a decrease in the number of extents used per file versus an increase in the amount of space wasted at the end of files that are not a multiple of the block size. Larger block sizes use less disk space in file system overhead, but consume more space for files that are not a multiple of the block size. The easiest way to judge which block sizes provide the greatest system efficiency is to try representative system loads against various sizes and pick the fastest. For most applications, it is best to use the default values.

Choosing an Intent Log Size

The intent log size is chosen when a file system is created and cannot be subsequently changed. The `mkfs` utility uses a default intent log size of 1024 blocks. The default size is sufficient for most workloads. If the system is used as an NFS server or for intensive synchronous write workloads, performance may be improved using a larger log size.

With larger intent log sizes, recovery time is proportionately longer and the file system may consume more system resources (such as memory) during normal operation.

There are several system performance benchmark suites for which VxFS performs better with larger log sizes. As with block sizes, the best way to pick the log size is to try representative system loads against various sizes and pick the fastest.

Note When using QuickLog, you choose the log at creation time and can easily change it at any time during use. For more information on log creation, log manipulation, and load balancing, see Chapter 10, “VERITAS QuickLog,”

Choosing Mount Options

In addition to the standard `mount` mode (`log` mode), VxFS provides `blkclear`, `delaylog`, `tmplog`, and `nodatainlog` modes of operation. Caching behavior can be altered with the `mincache` option, and the behavior of `O_SYNC` and `D_SYNC` (see the `fcntl`(2) manual page) writes can be altered with the `convosync` option.

The `delaylog` and `tmplog` modes can significantly improve performance. The improvement over `log` mode is typically about 15 to 20 percent with `delaylog`; with `tmplog`, the improvement is even higher. Performance improvement varies, depending on the operations being performed and the workload. Read / write intensive loads should show less improvement, while file system structure intensive loads (such as `mkdir`, `create`, and `rename`) may show over 100 percent improvement. The best way to select a mode is to test representative system loads against the logging modes and compare the performance results.

Most of the modes can be used in combination. For example, a desktop machine might use both the `blkclear` and `mincache=closesync` modes.

Additional information on `mount` options can be found in the `mount_vxfs`(1M) manual page.

log

The default logging mode is `log`. With `log` mode, VxFS guarantees that all structural changes to the file system have been logged on disk when the system call returns. If a system failure occurs, `fsck` replays recent changes so that they will not be lost.

delaylog

In `delaylog` mode, some system calls return before the intent log is written. This logging delay improves the performance of the system, but some changes are not guaranteed until a short time after the system call returns, when the intent log is written. If a system failure occurs, recent changes may be lost. This mode approximates traditional UNIX guarantees for correctness in case of system failures. Fast file system recovery works with this mode.

tmplog

In `tmplog` mode, intent logging is almost always delayed. This greatly improves performance, but recent changes may disappear if the system crashes. This mode is only recommended for temporary file systems. Fast file system recovery works with this mode.

nodatainlog

Use the `nodatainlog` mode on systems with disks that do not support bad block revectoring. Usually, a VxFS file system uses the intent log for synchronous writes. The inode update and the data are both logged in the transaction, so a synchronous write only requires one disk write instead of two. When the synchronous write returns to the application, the file system has told the application that the data is already written. If a disk error causes the data update to fail, then the file must be marked bad and the entire file is lost.

If a disk supports bad block revectoring, then a failure on the data update is unlikely, so logging synchronous writes should be allowed. If the disk does not support bad block revectoring, then a failure is more likely, so the `nodatainlog` mode should be used.

A `nodatainlog` mode file system is approximately 50 percent slower than a standard mode VxFS file system for synchronous writes. Other operations are not affected.

blkclear

The `blkclear` mode is used in increased data security environments. The `blkclear` mode guarantees that uninitialized storage never appears in files. The increased integrity is provided by clearing extents on disk when they are allocated within a file. Extending writes are not affected by this mode. A `blkclear` mode file system is approximately 10 percent slower than a standard mode VxFS file system, depending on the workload.

mincache

The mincache mode has five suboptions:

- mincache=closesync
- mincache=direct
- mincache=dsync
- mincache=unbuffered
- mincache=tmpcache

The mincache=closesync mode is useful in desktop environments where users are likely to shut off the power on the machine without halting it first. In this mode, any changes to the file are flushed to disk when the file is closed.

To improve performance, most file systems do not synchronously update data and inode changes to disk. If the system crashes, files that have been updated within the past minute are in danger of losing data. With the mincache=closesync mode, if the system crashes or is switched off, only files that are currently open can lose data. A mincache=closesync mode file system should be approximately 15 percent slower than a standard mode VxFS file system, depending on the workload.

The mincache=direct, mincache=unbuffered, and mincache=dsync modes are used in environments where applications are experiencing reliability problems caused by the kernel buffering of I/O and delayed flushing of non-synchronous I/O. The mincache=direct and mincache=unbuffered modes guarantee that all non-synchronous I/O requests to files will be handled as if the VX_DIRECT or VX_UNBUFFERED caching advisories had been specified. The mincache=dsync mode guarantees that all non-synchronous I/O requests to files will be handled as if the VX_DSYNC caching advisory had been specified. Refer to the vxfsio(7) manual page for explanations of VX_DIRECT, VX_UNBUFFERED, and VX_DSYNC. The mincache=direct, mincache=unbuffered, and mincache=dsync modes also flush file data on close as mincache=closesync does.

Since the mincache=direct, mincache=unbuffered, and mincache=dsync modes change non-synchronous I/O to synchronous I/O, there can be a substantial degradation in throughput for small to medium size files for most applications. Since the VX_DIRECT and VX_UNBUFFERED advisories do not allow any caching of data, applications that would normally benefit from caching for reads will usually experience less degradation with the mincache=dsync mode. mincache=direct and mincache=unbuffered require significantly less CPU time than buffered I/O.

If performance is more important than data integrity, the mincache=tmpcache mode may be used. The mincache=tmpcache mode disables special delayed extending write handling, trading off less integrity for better performance. Unlike the other mincache modes, tmpcache does not flush the file to disk when it is closed. When this option is used, garbage may appear in a file that was being extended when a crash occurred.

convosync

Note Use of the `convosync=dsync` option violates POSIX guarantees for synchronous I/O.

The "convert osync" (`convosync`) mode has five suboptions: `convosync=closesync`, `convosync=direct`, `convosync=dsync`, `convosync=unbuffered`, and `convosync=delay`.

The `convosync=closesync` mode converts synchronous and data synchronous writes to non-synchronous writes and flushes the changes to the file to disk when the file is closed.

The `convosync=delay` mode causes synchronous and data synchronous writes to be delayed rather than to take effect immediately. No special action is performed when closing a file. This option effectively cancels any data integrity guarantees normally provided by opening a file with `O_SYNC`. See the `open`(2), `fcntl`(2), and `vxfsio`(7) manual pages for more information on `O_SYNC`.

Caution Extreme care should be taken when using the `convosync=closesync` or `convosync=delay` mode because they actually change synchronous I/O into non-synchronous I/O. This may cause applications that use synchronous I/O for data reliability to fail if the system crashes and synchronously written data is lost.

The `convosync=direct` and `convosync=unbuffered` mode convert synchronous and data synchronous reads and writes to direct reads and writes.

The `convosync=dsync` mode converts synchronous writes to data synchronous writes.

As with `closesync`, the `direct`, `unbuffered`, and `dsync` modes flush changes to the file to disk when it is closed. These modes can be used to speed up applications that use synchronous I/O. Many applications that are concerned with data integrity specify the `O_SYNC fcntl` in order to write the file data synchronously. However, this has the undesirable side effect of updating inode times and therefore slowing down performance. The `convosync=dsync`, `convosync=unbuffered`, and `convosync=direct` modes alleviate this problem by allowing applications to take advantage of synchronous writes without modifying inode times as well.

Note Before using `convosync=dsync`, `convosync=unbuffered`, or `convosync=direct`, make sure that all applications that use the file system do not require synchronous inode time updates for `O_SYNC` writes.

qlog

qlog can be used in conjunction with the name of a QuickLog device. For example, to set the QuickLog device `vxlog1` to log the file system, use `qlog=vxlog1`. If `qlog=` is specified with no QuickLog device, the QuickLog driver chooses an appropriate log device automatically.

For more information, see Chapter 10, “VERITAS QuickLog,”

largefiles | nolargefiles

VxFS supports files up to two terabytes in size.

Note Be careful when enabling large file capability. Applications and utilities such as backup may experience problems if they are not aware of large files.

Creating a File System with Large Files

You can create a file system with large file capability by entering the following command:

```
# mkfs -F vxfs -o largefiles special_device size
```

Specifying `largefiles` sets the `largefiles` flag, which allows the file system to hold files up to two terabytes in size. Conversely, the default `nolargefiles` option clears the flag and prevents large files from being created:

```
# mkfs -F vxfs -o nolargefiles special_device size
```

Note The `largefiles` flag is persistent and stored on disk.

Mounting a File System with Large Files

If a mount succeeds and `nolargefiles` is specified, the file system cannot contain or create any large files. If a mount succeeds and `largefiles` is specified, the file system may contain and create large files.

The `mount` command fails if the specified `largefiles|nolargefiles` option does not match the on-disk flag.

The `mount` command defaults to match the current setting of the on-disk flag if specified without the `largefiles` or `nolargefiles` option, so it's best not to specify either option. After a file system is mounted, you can use the `fsadm` utility to change mount options.

Managing a File System with Large Files

You can determine the current status of the largefiles flag with either of the following commands:

```
# mkfs -F vxfs -m special special_device
```

```
# fsadm -F vxfs mount_point | special_device
```

You can switch capabilities on a mounted file system with the `fsadm` command:

```
# fsadm -F vxfs -o [no]largefiles mount_point
```

You can also switch capabilities on an unmounted file system:

```
# fsadm -F vxfs -o [no]largefiles special_device
```

You cannot switch a file system to `nolargefiles` if it holds large files.

See the `mount_vxfs`(1M), `fsadm_vxfs`(1M), and `mkfs_vxfs`(1M) manual pages.

Combining mount Command Options

Although `mount` options can be combined arbitrarily, some combinations do not make sense. The following examples provide some common and reasonable `mount` option combinations.

Example 1 - Desktop File System

```
# mount -F vxfs -o log,mincache=closesync /dev/dsk/c1t3d0s1 \
/mnt
```

This guarantees that when a file is closed, its data is synchronized to disk and cannot be lost. Thus, once an application is exited and its files are closed, no data will be lost even if the system is immediately turned off.

Example 2 - Temporary File System or Restoring from Backup

```
# mount -F vxfs -o tmplog,convosync=delay,mincache=tmpcache \
/dev/dsk/c1t3d0s1 /mnt
```

This combination might be used for a temporary file system where performance is more important than absolute data integrity. Any `O_SYNC` writes are performed as delayed writes and delayed extending writes are not handled specially (which could result in a file that contains garbage if the system crashes at the wrong time). Any file written 30 seconds or so before a crash may contain garbage or be missing if this `mount` combination is in effect. However, such a file system will do significantly less disk writes than a `log` file system, and should have significantly better performance, depending on the application.

Example 3 - Data Synchronous Writes

```
# mount -F vxfs -o log,convosync=dsync /dev/dsk/c1t3d0s1 /mnt
```

This combination would be used to improve the performance of applications that perform `O_SYNC` writes, but only require data synchronous write semantics. Their performance can be significantly improved if the file system is mounted using `convosync=dsync` without any loss of data integrity.

Kernel Tunables

This section describes the kernel tunables in VxFS.

Internal Inode Table Size

Each file system caches inodes are cached in an *inode table*. Every file system type has a tunable to determine the number of entries in its inode table. For VxFS, the tunable is `vxfs_ninode`.

The VxFS file system type uses the value of `vxfs_ninode` in `/etc/system` as the number of entries in the VxFS inode table. By default, the file system uses a value of `vxfs_ninode`, which is computed based on system memory size. To increase the value, make the following change in `/etc/system` and reboot:

```
set vxfs:vxfs_ninode = new_value
```

It may be necessary to tune the *dnlc* (directory name lookup cache) size to keep the value within an acceptable range relative to `vxfs_ninode`. It must be within 80% of `vxfs_ninode` to avoid spurious ENFILE errors or excessive CPU consumption, but must be more than 50% of `vxfs_ninode` to maintain good performance. The variable *ncsize* determines the size of dnlc. The default value of *ncsize* is based on the kernel variable *maxusers*. It is computed at system boot time. This value can be changed by making an entry in the `/etc/system` file:

```
set ncsize = new_value
```

The new *ncsize* is effective after you reboot the system.

VxVM Maximum I/O Size

If you are using VxFS in conjunction with the VERITAS Volume Manager (VxVM), VxVM by default breaks up I/O requests larger than 256K. When using striping, to optimize performance, the file system issues I/O requests that are up to a full stripe in size. If the stripe size is larger than 256K, those requests are broken up.

To avoid undesirable I/O breakup, you can increase the maximum I/O size by changing the value of the `vol_maxio` parameter in the `/etc/system` file.

vol_maxio

The `vol_maxio` parameter controls the maximum size of logical I/O operations that can be performed without breaking up a request. Logical I/O requests larger than this value are broken up and performed synchronously. Physical I/Os are broken up based on the capabilities of the disk device and are unaffected by changes to the `vol_maxio` logical request limit.

Raising the `vol_maxio` limit can cause problems if the size of an I/O requires more memory or kernel mapping space than exists. The recommended maximum for `vol_maxio` is 20% of the smaller of physical memory or kernel virtual memory. It is not advisable to go over this limit. Within this limit, you can generally obtain the best results by setting `vol_maxio` to the size of your largest stripe. This applies to both RAID-0 striping and RAID-5 striping.

To increase the value of `vol_maxio`, add an entry to `/etc/system` (after the entry `forceload:drv/vxio`) and reboot for the change to take effect. For example, the following line sets the maximum I/O size to 16 MB:

```
set vxio:vol_maxio=32768
```

This parameter is in 512-byte sectors and is stored as a 16-bit number, so it cannot be larger than 65535.

See the *VERITAS Volume Manager Administrator's Guide* for more information on avoiding I/O breakup by setting the maximum I/O tunable parameter.

Monitoring Free Space

In general, VxFS works best if the percentage of free space in the file system does not get below 10 percent. This is because file systems with 10 percent or more free space have less fragmentation and better extent allocation. Regular use of the `df` command to monitor free space is desirable. Full file systems may have an adverse effect on file system performance. Full file systems should therefore have some files removed, or should be expanded (see the `fsadm_vxfs`(1M) manual page for a description of online file system expansion).

Monitoring Fragmentation

Fragmentation reduces performance and availability. Regular use of `fsadm`'s fragmentation reporting and reorganization facilities is therefore advisable.

The easiest way to ensure that fragmentation does not become a problem is to schedule regular defragmentation runs from `cron`.

Defragmentation scheduling should range from weekly (for frequently used file systems) to monthly (for infrequently used file systems). Extent fragmentation should be monitored with `fsadm` or the `df -o s` commands. There are three factors which can be used to determine the degree of fragmentation:

- Percentage of free space in extents of less than eight blocks in length
- Percentage of free space in extents of less than 64 blocks in length
- Percentage of free space in extents of length 64 blocks or greater

An unfragmented file system will have the following characteristics:

- Less than 1 percent of free space in extents of less than eight blocks in length
- Less than 5 percent of free space in extents of less than 64 blocks in length
- More than 5 percent of the total file system size available as free extents in lengths of 64 or more blocks

A badly fragmented file system will have one or more of the following characteristics:

- Greater than 5 percent of free space in extents of less than 8 blocks in length
- More than 50 percent of free space in extents of less than 64 blocks in length
- Less than 5 percent of the total file system size available as free extents in lengths of 64 or more blocks

The optimal period for scheduling of extent reorganization runs can be determined by choosing a reasonable interval, scheduling `fsadm` runs at the initial interval, and running the extent fragmentation report feature of `fsadm` before and after the reorganization.

The "before" result is the degree of fragmentation prior to the reorganization. If the degree of fragmentation is approaching the figures for bad fragmentation, then the interval between `fsadm` runs should be reduced. If the degree of fragmentation is low, the interval between `fsadm` runs can be increased.

The “after” result is an indication of how well the reorganizer is performing. The degree of fragmentation should be close to the characteristics of an unfragmented file system. The file system may be a candidate for expansion. (Full file systems tend to fragment and are difficult to defragment.) It is also possible that the reorganization is not being performed at a time during which the file system in question is relatively idle.

Directory reorganization is not nearly as critical as extent reorganization, but regular directory reorganization will improve performance. It is advisable to schedule directory reorganization for file systems when the extent reorganization is scheduled. The following is a sample script that is run periodically at 3:00 A.M. from `cron` for a number of file systems:

```
outfile=/usr/spool/fsadm/out.‘/bin/date +’%m%d’‘
for i in /home /home2 /project /db
do
   /bin/echo "Reorganizing $i"
   /bin/timex fsadm -F vxfs -e -E -s $i
   /bin/timex fsadm -F vxfs -s -d -D $i
done > $outfile 2>&1
```

I/O Tuning

Note The tunables and the techniques described in this section are for tuning on a per file system basis and should be used judiciously based on the underlying device properties and characteristics of the applications that use the file system.

Performance of a file system can be enhanced by a suitable choice of I/O sizes and proper alignment of the I/O requests based on the requirements of the underlying special device. VxFS provides tools to tune the file systems.

Tuning VxFS I/O Parameters

VxFS provides a set of tunable I/O parameters that control some of its behavior. These I/O parameters are useful to help the file system adjust to striped or RAID-5 volumes that could yield performance far superior to a single disk. Typically, data streaming applications that access large files see the largest benefit from tuning the file system.

If VxFS is being used with the VERITAS Volume Manager, the file system queries VxVM to determine the geometry of the underlying volume and automatically sets the I/O parameters. VxVM is queried by `mkfs` when the file system is created to automatically align the file system to the volume geometry. The `mount` command also queries VxVM when the file system is mounted and downloads the I/O parameters.

If the default parameters are not acceptable or the file system is being used without VxVM, then the `/etc/vx/tunefstab` file can be used to set values for I/O parameters. The `mount` command reads the `/etc/vx/tunefstab` file and downloads any parameters specified for a file system. The `tunefstab` file overrides any values obtained from VxVM. While the file system is mounted, any I/O parameters can be changed using the `vxtunefs` command which can have tunables specified on the command line or can read them from the `/etc/vx/tunefstab` file. For more details, see the `vxtunefs`(1M) and `tunefstab`(4) manual pages. The `vxtunefs` command can be used to print the current values of the I/O parameters.

If the default alignment from `mkfs` is not acceptable, the `-o align=n` option can be used to override alignment information obtained from VxVM.

Tunable VxFS I/O Parameters

`read_pref_io`	The preferred read request size. The file system uses this in conjunction with the `read_nstream` value to determine how much data to read ahead. The default value is 64K.
`write_pref_io`	The preferred write request size. The file system uses this in conjunction with the `write_nstream` value to determine how to do flush behind on writes. The default value is 64K.
`read_nstream`	The number of parallel read requests of size `read_pref_io` to have outstanding at one time. The file system uses the product of `read_nstream` multiplied by `read_pref_io` to determine its read ahead size. The default value for `read_nstream` is 1.
`write_nstream`	The number of parallel write requests of size `write_pref_io` to have outstanding at one time. The file system uses the product of `write_nstream` multiplied by `write_pref_io` to determine when to do flush behind on writes. The default value for `write_nstream` is 1.
`default_indir_size`	On VxFS, files can have up to ten direct extents of variable size stored in the inode. Once these extents are used up, the file must use indirect extents which are a fixed size that is set when the file first uses indirect extents. These indirect extents are 8K by default. The file system does not use larger indirect extents because it must fail a write and return `ENOSPC` if there are no extents available that are the indirect extent size. For file systems with a lot of large files, the 8K indirect extent size is too small. The files that get into indirect extents use a lot of smaller extents instead of a few larger ones. By using this parameter, the default indirect extent size can be increased so large that files in indirects use fewer larger extents. The tunable `default_indir_size` should be used carefully. If it is set too large, then writes will fail when they are unable to allocate extents of the indirect extent size to a file. In general, the fewer and the larger the files on a file system, the larger the `default_indir_size` can be set. This parameter should generally be set to some multiple of the `read_pref_io` parameter. `default_indir_size` is not applicable on Version 4 disk layouts.

`discovered_direct_iosz`	Any file I/O requests larger than the `discovered_direct_iosz` are handled as discovered direct I/O. A discovered direct I/O is unbuffered similar to direct I/O, but it does not require a synchronous commit of the inode when the file is extended or blocks are allocated. For larger I/O requests, the CPU time for copying the data into the page cache and the cost of using memory to buffer the I/O data becomes more expensive than the cost of doing the disk I/O. For these I/O requests, using discovered direct I/O is more efficient than regular I/O. The default value of this parameter is 256K.
`initial_extent_size`	Changes the default initial extent size. VxFS determines, based on the first write to a new file, the size of the first extent to be allocated to the file. Normally the first extent is the smallest power of 2 that is larger than the size of the first write. If that power of 2 is less than 8K, the first extent allocated is 8K. After the initial extent, the file system increases the size of subsequent extents (see `max_seqio_extent_size`) with each allocation. Since most applications write to files using a buffer size of 8K or less, the increasing extents start doubling from a small initial extent. `initial_extent_size` can change the default initial extent size to be larger, so the doubling policy will start from a much larger initial size and the file system will not allocate a set of small extents at the start of file. Use this parameter only on file systems that will have a very large average file size. On these file systems it will result in fewer extents per file and less fragmentation. `initial_extent_size` is measured in file system blocks.
`max_direct_iosz`	The maximum size of a direct I/O request that will be issued by the file system. If a larger I/O request comes in, then it is broken up into `max_direct_iosz` chunks. This parameter defines how much memory an I/O request can lock at once, so it should not be set to more than 20 percent of memory.
`max_diskq`	Limits the maximum disk queue generated by a single file. When the file system is flushing data for a file and the number of pages being flushed exceeds `max_diskq`, processes will block until the amount of data being flushed decreases. Although this doesn't limit the actual disk queue, it prevents flushing processes from making the system unresponsive. The default value is 1 MB.

`max_seqio_extent_size`	Increases or decreases the maximum size of an extent. When the file system is following its default allocation policy for sequential writes to a file, it allocates an initial extent which is large enough for the first write to the file. When additional extents are allocated, they are progressively larger (the algorithm tries to double the size of the file with each new extent) so each extent can hold several writes worth of data. This is done to reduce the total number of extents in anticipation of continued sequential writes. When the file stops being written, any unused space is freed for other files to use. Normally this allocation stops increasing the size of extents at 2048 blocks which prevents one file from holding too much unused space. `max_seqio_extent_size` is measured in file system blocks.
`qio_cache_enable`	Enables or disables caching on Quick I/O files. The default behavior is to disable caching. To enable caching, set `qio_cache_enable` to 1. On systems with large memories, the database cannot always use all of the memory as a cache. By enabling file system caching as a second level cache, performance may be improved. If the database is performing sequential scans of tables, the scans may run faster by enabling file system caching so the file system will perform aggressive read-ahead on the files.

`write_throttle`

The `write_throttle` parameter is useful in special situations where a computer system has a combination of a lot of memory and slow storage devices. In this configuration, sync operations (such as `fsync()`) may take long enough to complete that a system appears to hang. This behavior occurs because the file system is creating *dirty pages* (in-memory updates) faster than they can be asynchronously flushed to disk without slowing system performance.

Lowering the value of `write_throttle` limits the number of dirty pages per file that a file system will generate before flushing the pages to disk. After the number of dirty pages for a file reaches the `write_throttle` threshold, the file system starts flushing pages to disk even if free memory is still available. The default value of `write_throttle` typically generates a lot of dirty pages, but maintains fast user writes. Depending on the speed of the storage device, if you lower `write_throttle`, user write performance may suffer, but the number of dirty pages is limited, so sync operations will complete much faster.

Because lowering `write_throttle` may in some cases delay write requests (for example, lowering `write_throttle` may increase the file disk queue to the `max_diskq` value, delaying user writes until the disk queue decreases), it is advisable not to change the value of `write_throttle` unless your system has a combination of large physical memory and slow storage devices.

If the file system is being used with VxVM, it is advisable to let the VxFS I/O parameters get set to default values based on the volume geometry.

If the file system is being used with a hardware disk array or volume manager other than VxVM, try to align the parameters to match the geometry of the logical disk. With striping or RAID-5, it is common to set `read_pref_io` to the stripe unit size and `read_nstream` to the number of columns in the stripe. For striping arrays, use the same values for `write_pref_io` and `write_nstream`, but for RAID-5 arrays, set `write_pref_io` to the full stripe size and `write_nstream` to 1.

For an application to do efficient disk I/O, it should issue read requests that are equal to the product of `read_nstream` multiplied by `read_pref_io`. Generally, any multiple or factor of `read_nstream` multiplied by `read_pref_io` should be a good size for performance. For writing, the same rule of thumb applies to the `write_pref_io` and `write_nstream` parameters. When tuning a file system, the best thing to do is try out the tuning parameters under a real life workload.

If an application is doing sequential I/O to large files, it should try to issue requests larger than the `discovered_direct_iosz`. This causes the I/O requests to be performed as discovered direct I/O requests, which are unbuffered like direct I/O but do not require synchronous inode updates when extending the file. If the file is larger than can fit in the cache, then using unbuffered I/O avoids throwing useful data out of the cache and it avoids a lot of CPU overhead.

Application Interface 5

Introduction

The VERITAS File System (VxFS) provides enhancements that can be used by applications that require certain performance features. This chapter describes cache advisories and provides information about fixed extent sizes and reservation of space for a file.

This chapter describes how the application writer can optimize applications for use with the VxFS. To optimize VxFS for use with applications, see Chapter 4, “Performance and Tuning.”

The following topics are covered in this chapter:

- Cache Advisories
  - Direct I/O
  - Unbuffered I/O
  - Discovered Direct I/O
  - Data Synchronous I/O
  - Other Advisories
- Extent Information
  - Space Reservation
  - Fixed Extent Sizes
  - Freeze and Thaw
- Get I/O Parameters ioctl

Cache Advisories

VxFS allows an application to set cache advisories for use when accessing files. These advisories are in memory only and they do not persist across reboots. Some advisories are currently maintained on a per-file, not a per-file-descriptor, basis. This means that only one set of advisories can be in effect for all accesses to the file. If two conflicting applications set different advisories, both use the last advisories that were set.

All advisories are set using the `VX_SETCACHE` ioctl command. The current set of advisories can be obtained with the `VX_GETCACHE` ioctl command. For details on the use of these ioctl commands, see the `vxfsio`(7) manual page.

Direct I/O

Direct I/O is an unbuffered form of I/O. If the `VX_DIRECT` advisory is set, the user is requesting direct data transfer between the disk and the user-supplied buffer for reads and writes. This bypasses the kernel buffering of data, and reduces the CPU overhead associated with I/O by eliminating the data copy between the kernel buffer and the user's buffer. This also avoids taking up space in the buffer cache that might be better used for something else. The direct I/O feature can provide significant performance gains for some applications.

For an I/O operation to be performed as direct I/O, it must meet certain alignment criteria. The alignment constraints are usually determined by the disk driver, the disk controller, and the system memory management hardware and software. The file offset must be aligned on a sector boundary.

If a request fails to meet the alignment constraints for direct I/O, the request is performed as data synchronous I/O. If the file is currently being accessed by using memory mapped I/O, any direct I/O accesses are done as data synchronous I/O.

Since direct I/O maintains the same data integrity as synchronous I/O, it can be used in many applications that currently use synchronous I/O. If a direct I/O request does not allocate storage or extend the file, the inode is not immediately written.

The CPU cost of direct I/O is about the same as a raw disk transfer. For sequential I/O to very large files, using direct I/O with large transfer sizes can provide the same speed as buffered I/O with much less CPU overhead.

If the file is being extended or storage is being allocated, direct I/O must write the inode change before returning to the application. This eliminates some of the performance advantages of direct I/O.

The direct I/O and `VX_DIRECT` advisories are maintained on a per-file-descriptor basis.

Unbuffered I/O

If the VX_UNBUFFERED advisory is set, I/O behavior is the same as direct I/O with the VX_DIRECT advisory set, so the alignment constraints that apply to direct I/O also apply to unbuffered. For I/O with unbuffered I/O, however, if the file is being extended, or storage is being allocated to the file, inode changes are not updated synchronously before the write returns to the user. The VX_UNBUFFERED advisory is maintained on a per-file-descriptor basis.

Discovered Direct I/O

Discovered Direct I/O is not a cache advisory that the user can set using the VX_SETCACHE ioctl. When the file system gets an I/O request larger than the discovered_direct_iosz, it tries to use direct I/O on the request. For large I/O sizes, Discovered Direct I/O can perform much better than buffered I/O.

Discovered Direct I/O behavior is similar to direct I/O and has the same alignment constraints, except writes that allocate storage or extend the file size do not require writing the inode changes before returning to the application.

For information on how to set the discovered_direct_iosz, see "I/O Tuning" on page 43.

Data Synchronous I/O

If the VX_DSYNC advisory is set, the user is requesting data synchronous I/O. In synchronous I/O, the data is written, and the inode is written with updated times and (if necessary) an increased file size. In data synchronous I/O, the data is transferred to disk synchronously before the write returns to the user. If the file is not extended by the write, the times are updated in memory, and the call returns to the user. If the file is extended by the operation, the inode is written before the write returns.

Like direct I/O, the data synchronous I/O feature can provide significant application performance gains. Since data synchronous I/O maintains the same data integrity as synchronous I/O, it can be used in many applications that currently use synchronous I/O. If the data synchronous I/O does not allocate storage or extend the file, the inode is not immediately written. The data synchronous I/O does not have any alignment constraints, so applications that find it difficult to meet the alignment constraints of direct I/O should use data synchronous I/O.

If the file is being extended or storage is allocated, data synchronous I/O must write the inode change before returning to the application. This case eliminates the performance advantage of data synchronous I/O.

The direct I/O and VX_DSYNC advisories are maintained on a per-file-descriptor basis.

Other Advisories

The VX_SEQ advisory indicates that the file is being accessed sequentially. When the file is being read, the maximum read-ahead is always performed. When the file is written, instead of trying to determine whether the I/O is sequential or random by examining the write offset, sequential I/O is assumed. The pages for the write are not immediately flushed. Instead, pages are flushed some distance behind the current write point.

The VX_RANDOM advisory indicates that the file is being accessed randomly. For reads, this disables read-ahead. For writes, this disables the flush-behind. The data is flushed by the pager, at a rate based on memory contention.

The VX_NOREUSE advisory is used as a modifier. If both VX_RANDOM and VX_NOREUSE are set, pages are immediately freed and put on the quick reuse free list as soon as the data has been used. If VX_NOREUSE is set when doing sequential I/O, pages are also put on the quick reuse free list when they are flushed. The VX_NOREUSE may slow down access to the file, but it can reduce the cached data held by the system. This can allow more data to be cached for other files and may speed up those accesses.

Extent Information

The VX_SETEXT ioctl command allows an application to reserve space for a file, and set fixed extent sizes and file allocation flags. The current state of much of this information can be obtained by applications by using the VX_GETEXT ioctl (the `getext` command provides access to this functionality). For details, see the `getext`(1), `setext`(1), and `vxfsio`(7) manual pages.

Each invocation of the VX_SETEXT ioctl affects all the elements in the `vx_ext` structure. When using VX_SETEXT, always use the following procedure:

1. Use VX_GETEXT to read the current settings.
2. Modify the values to be changed.
3. Call VX_SETEXT to set the values.

Note Follow this procedure carefully. Otherwise, a fixed extent size could be cleared when the reservation is changed.

Space Reservation

Storage can be reserved for a file at any time. When a VX_SETEXT ioctl is issued, the reservation value is set in the inode on disk. If the file size is less than the reservation amount, the kernel allocates space to the file from the current file size up to the reservation amount. When the file is truncated, space below the reserved amount is not freed. The VX_TRIM, VX_NOEXTEND, VX_CHGSIZE, VX_NORESERVE and VX_CONTIGUOUS flags can be used to modify reservation requests.

Note VX_NOEXTEND is the only one of these flags that is persistent; the other flags may have persistent effects, but they are not returned by the VX_GETEXT ioctl.

If the VX_TRIM flag is set, when the last close occurs on the inode, the reservation is trimmed to match the file size and the VX_TRIM flag is cleared. Any unused space is freed. This can be useful if an application needs enough space for a file, but it is not known how large the file will become. Enough space can be reserved to hold the largest expected file, and when the file has been written and closed, any extra space will be released.

If the VX_NOEXTEND flag is set, an attempt to write beyond the current reservation, which requires the allocation of new space for the file, fails instead. To allocate new space to the file, the space reservation must be increased. This can be used like ulimit to prevent a file from using too much space.

If the VX_CONTIGUOUS flag is set, any space allocated to satisfy the current reservation request is allocated in one extent. If there is not one extent large enough to satisfy the request, the request fails. For example, if a file is created and a 1 MB contiguous reservation is requested, the file size is set to zero and the reservation to 1 MB. The file will have one extent that is 1 MB long. If another reservation request is made for a 3 MB contiguous reservation, the new request will find that the first 1 MB is already allocated and allocate a 2 MB extent to satisfy the request. If there are no 2 MB extents available, the request fails. (Extents are, by definition, contiguous.)

Note Because VX_CONTIGUOUS is not a persistent flag, space will not be allocated contiguously after doing a file system restore.

If the VX_NORESERVE flag is set, the reservation value in the inode is not changed. This flag is used by applications to do temporary reservation. Any space past the end of the file is given up when the file is closed. For example, if the cp command is copying a file that is 1 MB long, it can request a 1 MB reservation with the VX_NORESERVE flag set. The space is allocated, but the reservation in the file is left at 0. If the program aborts for any reason or the system crashes, the unused space past the end of the file is released. When the program finishes, there is no cleanup because the reservation was never recorded on disk.

If the VX_CHGSIZE flag is set, the file size is increased to match the reservation amount. This flag can be used to create files with uninitialized data. Because this allows uninitialized data in files, it is restricted to users with appropriate privileges.

It is possible to use these flags in combination. For example, using VX_CHGSIZE and VX_NORESERVE changes the file size but does not set any reservation. When the file is truncated, the space is freed. If the VX_NORESERVE flag had not been used, the reservation would have been set on disk along with the file size.

Space reservation is used to make sure applications do not fail because the file system is out of space. An application can preallocate space for all the files it needs before starting to do any work. By allocating space in advance, the file is optimally allocated for performance, and file accesses are not slowed down by the need to allocate storage. This allocation of resources can be important in applications that require a guaranteed response time.

With very large files, use of space reservation can avoid the need to use indirect extents. It can also improve performance and reduce fragmentation by guaranteeing that the file consists of large contiguous extents. Sometimes when critical file systems run out of space, cron jobs, mail, or printer requests fail. These failures are harder to track if the logs kept by the application cannot be written due to a lack of space on the file system.

By reserving space for key log files, the logs will not fail when the system runs out of space. Process accounting files can also have space reserved so accounting records will not be lost if the file system runs out of space. In addition, by using the VX_NOEXTEND flag for log files, the maximum size of these files can be limited. This can prevent a runaway failure in one component of the system from filling the file system with error messages and causing other failures. If the VX_NOEXTEND flag is used for log files, the logs should be cleaned up before they reach the size limit in order to avoid losing information.

Fixed Extent Sizes

VxFS uses the I/O size of write requests, and a default policy, when allocating space to a file. For some applications, this may not work out well. These applications can set a fixed extent size, so that all new extents allocated to the file are of the fixed extent size.

By using a fixed extent size, an application can reduce allocations and guarantee good extent sizes for a file. An application can reserve most of the space a file needs, and then set a relatively large fixed extent size. If the file grows beyond the reservation, any new extents are allocated in the fixed extent size.

Another use of a fixed extent size occurs with sparse files. The file system usually does I/O in page size multiples. When allocating to a sparse file, the file system allocates pages as the smallest default unit. If the application always does sub-page I/O, it can request a fixed extent size to match its I/O size and avoid wasting extra space.

When setting a fixed extent size, an application should not select too large a size. When all extents of the required size have been used, attempts to allocate new extents fail: this failure can happen even though there are blocks free in smaller extents.

Fixed extent sizes can be modified by the `VX_ALIGN` flag. If the `VX_ALIGN` flag is set, then any future extents allocated to the file are aligned on a fixed extent size boundary relative to the start of the allocation unit. This can be used to align extents to disk striping boundaries or physical disk boundaries.

The `VX_ALIGN` flag is persistent and is returned by the `VX_GETEXT` ioctl.

Freeze and Thaw

The VX_FREEZE ioctl command is used to freeze a file system. Freezing a file system temporarily blocks all I/O operations to a file system and then performs a sync on the file system. When the VX_FREEZE ioctl is issued, all access to the file system is blocked at the system call level. Current operations are completed and the file system is synchronized to disk. Freezing provides a stable, consistent file system.

When the file system is frozen, any attempt to use the frozen file system, except for a VX_THAW ioctl command, is blocked until a process executes the VX_THAW ioctl command or the time-out on the freeze expires.

Note While a file system is frozen by the VX_FREEZE ioctl, all logging performed by the QuickLog device is suspended. Logging of the file system is resumed once the VX_THAW ioctl is issued. For more information on the effects of these ioctls on QuickLog, see Chapter 10, "VERITAS QuickLog."

Get I/O Parameters ioctl

VxFS provides the VX_GET_IOPARAMETERS ioctl to get the recommended I/O sizes to use on a file system. This ioctl can be used by the application to make decisions about the I/O sizes issued to VxFS for a file or file device. For more details on this ioctl, refer to the vxfsio(7) manual page. For a discussion on various I/O parameters, refer to Chapter 4, "Performance and Tuning," and the vxtunefs(1M) manual page.

Cluster File System 6

Introduction

The VERITAS Cluster File System (CFS) is a shared file system that enables multiple hosts to mount and perform file operations concurrently on the same file. CFS is a feature of the VERITAS File System (VxFS), but requires an integrated set of VERITAS products to function.

To configure a cluster and to provide failover support, CFS requires the VERITAS Cluster Server (VCS). VCS supplies two major components integral to CFS. The Low Latency Transport (LLT) package provides node-to-node communications and monitors network communications. The Group Membership and Atomic Broadcast (GAB) package provides cluster state, configuration, and membership service, and monitors the heartbeat links between systems to ensure that they are active.

CFS also requires the VERITAS Volume Manager (VxVM) to create the cluster volumes (CVM) necessary for mounting cluster file systems.

Note To install and administer cluster file systems, you should have a working knowledge of VCS and VxVM. For more information on these products, refer to the VERITAS Cluster Server and VERITAS Volume Manager documentation. The user guides for these products are available in the `/opt` directory after you install the CFS packages.

Topics in this chapter include:

VCS Overview

The VERITAS Cluster Server provides the communication, configuration, and membership services required to create a cluster. VCS is the first component installed and configured to set up a cluster file system (see the *VERITAS Cluster File System Installation and Configuration Guide*).

VCS Communication

GAB and LLT are VCS-specific protocols implemented directly on an Ethernet data link or on a Fibre Channel fabric. Both GAB and LLT run over redundant data links that connect all the servers in a cluster. VCS requires redundant cluster communication links to minimize the possibility of cluster failure due to the failure of a single communication link.

GAB

GAB is a broadcast protocol which guarantees that messages broadcast throughout a cluster are received in the same order by all nodes in the cluster and acknowledged. The main function of GAB is to provide a membership service, both for the cluster as a whole, and for groups of applications running it. The GAB membership service provides orderly startup and shutdown of a cluster.

The file `/etc/gabtab` is used to configure GAB. Configuration is done with the `gabconfig` command. For example, the `–n` option of the command specifies the number of nodes in the cluster. GAB is configured automatically when you run the VCS installation script, but you may have to reconfigure GAB when you add a node to a cluster. See the `gabconfig`(1m) manual page for additional information.

LLT

LLT provides kernel-to-kernel communications and monitors network communications. The LLT files `/etc/llthosts` and `/etc/lltab` can be configured to set system IDs within a cluster, set cluster IDs for multiple clusters, and tune network parameters such as heartbeat frequency. LLT is implemented so that events such as state changes are reflected quickly, which in turn enables fast responses.

As with GAB, LLT is configured automatically when you run the VCS installation script. The file /etc/`lltab` contains information derived from what you input during installation. You may also have to reconfigure LLT when you add a node to a cluster. See the `lltab`(4) manual page for details on how you can modify the LLT configuration.

CVM Overview

The cluster functionality (CVM) of the VERITAS Volume Manager allows multiple hosts to concurrently access and manage a given set of logical devices under VxVM control. A VxVM cluster is a set of hosts sharing a set of devices; each host is a node in the cluster. The nodes are connected across a network. If one node fails, other nodes can still access the devices. The VxVM cluster feature presents the same logical view of the device configurations, including changes, on all nodes.

You configure CVM shared storage after VCS sets up a cluster configuration (see the *VERITAS Cluster File System Installation and Configuration Guide*). You must initialize at least one, non-shared disk into `rootdg` for use by CFS.

CFS Overview

A file system cluster consists of one primary (or master), and up to three secondaries (or slaves). The primary-secondary terminology applies to one file system, not to a specific node (or hardware platform). So it is possible, and in fact recommended, to have the same CFS node be primary for one shared volume, while at the same time it is secondary for another shared volume.

For CVM, a single node is the master for all shared disk groups and shared volumes.

Cluster/Shared Mount

A VxFS file system that is mounted with the `mount -o cluster` option is called a cluster or *shared* mount. A file system mounted in shared mode must be on a VxVM shared volume in a cluster (CVM) environment. All nodes in a cluster must be mounted with identical options.

A local mount cannot be remounted in shared mode and a shared mount cannot be remounted in local mode. Do not mount a shared device in local mode because the same device can be mounted from another node, exposing data to risk of corruption.

CFS Primary and CFS Secondary

The primary node handles the metadata intent logging for the cluster. The first node of a cluster to mount is called the primary node. Other nodes are called secondary nodes. If a primary node fails, an internal election process determines which of the secondaries becomes the primary.

Because all nodes must be mounted with identical options, a secondary writable node is not allowed if the primary node is mounted as read-only.

Use the following command to determine primaryship:

```
# fsclustadm –v showprimary mount_point
```

Use the following command to give primaryship to a node,

```
# fsclustadm –v setprimary mount_point
```

CFS and CVM Agents

Agents are VCS processes that manage predefined resource types. CFS and CVM require agents to interact with VCS. Agents bring resources online, take resources offline, monitor resources, and report any state changes to VCS. VCS bundled agents are part of VCS and are installed when VCS is installed. The CFS and CVM agents are add-on resources to VCS specifically for the VERITAS File System and VERITAS Volume Manager (see the *VERITAS Cluster File System Installation and Configuration Guide* for detailed information on CFS and CVM agents).

When to Use CFS

You should use CFS for any application that requires the sharing of files, such as for home directories and boot server files, Web pages, and for cluster-ready applications. CFS is also applicable when you want highly available standby data, in predominantly read-only environments where you just need to access data, or when you do not want to rely on NFS for file sharing.

CFS Administration

This section describes some of the major aspects of administering cluster file systems and differences from the single-host administration of VxFS.

CFS Commands

The `mount` and `fsclustadm` commands are the most important for configuring cluster file systems.

mount

The `mount` command with the `-o cluster` option is what enables you to create shared file systems. See the `mount_vxfs`(1M) manual page for information on allowable mount options.

fsclustadm

The `fsclustadm` command reports various attributes of a cluster file system. Using `fsclustadm` you can show and set the primary node in a cluster, translate node IDs to host names and vice versa, list all nodes that currently have a cluster mount of the specified file system mount point, and determine whether a mount is a local or cluster mount. The `fsclustadm` command operates from any node in a cluster. See the `fsclustadm`(1M) manual page for information on usage.

fsadm

The `fsadm` must be run from the primary node; it fails if invoked on secondaries. See the `fsadm`(1M) manual page for information on allowable mount options.

Running Commands Safely in a Cluster Environment

Any UNIX command that can write to a raw device must be used carefully in a shared environment to prevent data from being corrupted. For shared VxVM volumes, CFS provides protection by reserving the volumes in a cluster to prevent VxFS commands, such as `fsck` and `mkfs`, from inadvertently damaging a mounted file system from another node in a cluster. However, commands such as `dd` execute without any reservation, and can damage a file system mounted from another node. Before running this kind of command on a file system, be sure the file system is not mounted on a cluster. You can run the `mount` command to see if a file system is a shared or local mount.

Time Synchronization for Cluster File Systems

CFS requires that the system clocks on all nodes are synchronized using some external component such as the Network Time Protocol (NTP) daemon. If the nodes are not in sync, timestamps for creation (`ctime`) and modification (`mtime`) may not be consistent with the sequence in which operations actually happened.

Growing a Cluster File System

There is a master node for CVM as well as a primary for CFS. When growing a file system, you grow the volume from the CVM master, and then grow the file system from the CFS primary. Growing a file system only works if both the CVM master and CFS primary are on the system you are running this from.

To determine the primary file system in a cluster, enter:

```
# fsclustadm showprimary
```

To determine the master CVM node, enter:

```
# vxdctl -c mode
```

The vfstab File

In the `/etc/vfstab` file, do not specify any cluster file systems to mount-at-boot because mounts initiated from `vfstab` occur before cluster configuration begins. For cluster mounts, use the VCS configuration file to determine which file systems to enable following a reboot.

Distributing the Load on a Cluster

Distributing the workload in a cluster provides performance and failover advantages. Because each cluster mounted file system can have a different node as its primary, CFS lets you easily distribute load in a cluster.

For example, if you have eight file systems and four nodes, designating two file systems per node as the primary would be beneficial. Primaryship is determined by which node first mounts the file system. You can also use the `fsclustadm` to designate a CFS primary.

Using GUIs

The VERITAS Volume Manager Storage Administrator (VMSA) version included in the first release of CFS is not cluster aware, so you can use VMSA only for VxFS functions such as making and mounting local file systems. In subsequent releases of CFS, VMSA will have cluster functionality included. Be sure to read the *VERITAS File System Release Notes* for this update information.

Use the VCS Cluster Manager GUI to configure and monitor CFS. The VCS GUI provides log files for debugging LLT and GAB events.

Will It Work on CFS?

To find out if a feature operates on CFS, refer to the summary in "VxFS Functionality on Cluster File Systems" on page 67. Also, every VxFS online manual page has a CFS Issues section that states whether the command functions on a cluster-mounted file system.

CFS Troubleshooting

If there is a device failure or controller failure to a device, the file system may become disabled cluster-wide. To address the problem, unmount all secondary mounts, unmount the primary, then run a full `fsck`. When the file system check completes, mount all nodes again.

Possible Mount Failures

The `mount -o cluster` command can fail for the following reasons:

- The file system is not using disk layout Version 4
- The mount options do not match the options of already mounted nodes.
- A shared CVM volume was not specified.
- The device is still mounted as a local file system somewhere on the cluster. Unmount the device.
- The `fsck` or `mkfs` command is being run on the same volume from another node, or the volume is mounted in non-cluster mode from another node.
- The `vxfsckd` daemon is not running. This will typically happen only if the `CFSfsckd` agent was not started correctly.

Possible Unmount Failures

The `umount` command can fail for the following reasons:

- When unmounting shared file systems, you must mount the secondaries before unmounting the primary. This is true even for forced unmounts (`umount -o force`). Unmount from the secondaries first, or move the primaryship to another node using `fsclustadm setprimary` command.
- A reference is being held by an NFS server. Unshare the mount point and try the unmount again.

High Availability Issues

Network Partitioning

Network partitioning (or *split brain*) is a condition where a network failure can be misinterpreted as a failure of one or more nodes in a cluster. If one system in the cluster assumes wrongly that another system failed, it may restart applications already running on the other system, thereby corrupting data. VCS tries to prevent this by having redundant heartbeat links. Do not remove the communication links while shared storage is still connected.

Low Memory

Under very heavy loads, the heartbeat communication links may run out of kernel memory. If this occurs, a node will be halted to avoid any chance of network partitioning. Reduce the load on the node if this happens frequently.

A similar situation may occur if the LLT values in the `/etc/lltab` files on all cluster nodes are not correct or identical.

Quick Node Join

It is advisable to bring up one node at a time. If nodes are started simultaneously, CVM online agents may fail. If this happens, bring the affected node online manually.

Quick Node Exit

It is also advisable to take one node offline at a time. Always use VCS to shut down resources to ensure that it is done in the correct order. Never use the `halt` or `init` commands to take a node offline.

VxFS Functionality on Cluster File Systems

This section lists major VxFS features and functionality and states whether they are currently supported on cluster file systems. Some of the items listed will be supported in future releases of CFS. See the *VERITAS File System Release Notes* for more information.

Functionality that is described as not supported may not be expressly prevented from operating on cluster file systems, but because CFS is not looking for these attributes, the actual behavior is indeterminate. It is not advisable to use unsupported functionality on CFS, or to alternate mounting file systems with these options as local and cluster mounts.

Not Supported

QuickLog

QuickLog is not supported on cluster file systems.

Storage Checkpoints

Storage Checkpoints are not supported on cluster file systems.

Snapshots

Snapshots are not supported on cluster file systems.

Quotas

Quotas are not supported on cluster file systems. You cannot use local mounted file systems that were quota-enabled as part of a cluster.

ACLs

Access control lists are not supported on cluster file systems. Do not cluster mount file systems that contains ACLs.

DMAPI

Data Management Application Programming Interface, typically used for hierarchical storage management, is not supported on cluster file systems.

Cache Advisories

Cache advisories are set with the mount command on individual file systems, but are not propagated to other nodes of a cluster.

Commands That Depend on File Access Times

File access times may appear different across nodes because the `atime` file attribute is not closely synchronized in a cluster file system, so utilities that depend on checking access times may not work correctly.

Supported

Quick I/O

The Quick I/O for Databases features is supported on CFS, but is licensable only through VERITAS Database Editions products. Cached Quick I/O is not supported.

Disk Layout Version 4

CFS supports only the Version 4 disk layout. Cluster mounted file systems cannot be upgraded, but a local mounted file system can be upgraded, unmounted, and mounted again as part of a cluster. Use the `fstyp -v` *special_device* command to ascertain the disk layout version of a VxFS file system.

Freeze and Thaw

Sync operations, which require freezing and thawing file systems, are done on a cluster-wide basis.

Locking

Record locking and advisory file locking are supported on CFS. For the F_GETLK command, if there is a process holding a conflicting lock, the `l_pid` field returns the process ID of the process holding the conflicting lock, and `l_sysid` is set to the `nodeid` of the node on which the process holding the conflicting lock is running. The nodeid-to-node name translation can be done by examining the `/etc/llthosts` file or with the `fsclustadm` command.

Mandatory locking, and deadlock detection supported by traditional `fcntl` locks, are not supported on CFS. See the `fscntl`(2) manual page for more information on record and file locking.

Memory Mapping

Shared memory mapping established by the `mmap`() function is supported on CFS. See the `mmap`(2) manual page for more information on mapping memory pages.

NFS Mounts

You can mount cluster file systems to NFS.

Configuration Information

The *VERTAS Cluster File System Installation and Configuration Guide* provides extensive background information on CFS.

- Hardware overview and setup—describes a typical cluster configuration over a *Fibre Channel fabric* and provides a list of recommended platforms, arrays, switches, and components.
- Software components—lists the VERITAS software packages required to operate CFS.
- Installing VCS—describes how to install VCS using the VCS interactive installation utility, then how to verify that the VCS components, LLT, GAB, and the Cluster Manager GUI, were installed correctly and are operating.
- Cluster package installation—describes how to install CFS using the `cfsinstall` script, and how to install the CFS license key and software patches.
- Configuring and initializing VxVM cluster functionality—describes how to set up shared storage for CFS.
- Cluster file system agents and their installation—defines the agents required to create VCS and CFS interoperability, and describes the CFS agent installation procedure.
- Adding a node to an existing cluster—describes how to add a new node to a previously configured cluster.
- Potential problems installing VCS and CFS—analyzes some typical problems you can encounter during the installation process and doing basic CFS operations.
- Creating a shared disk group, creating a cluster file system, and mounting a cluster file system in shared mode—provides an example of the each of these procedures.
- Deinstalling the CFS and VCS packages—how to deinstall all VERITAS packages using scripts or the `pkgrm` command.

VERITAS Cluster File System Technical Overview

The VERITAS Cluster File System allows clustered servers to mount and use a file system simultaneously as if all applications using the file system were running on the same server. The VERITAS Volume Manager cluster functionality (CVM) makes logical volumes and raw device applications accessible throughout a cluster.

VERITAS Cluster File System Architecture

Master/Slave File System Design

The VERITAS Cluster File System uses a master/slave, or primary/secondary architecture to manage file system metadata on shared disk storage. The first server to mount each cluster file system becomes its primary; all other nodes in the cluster become secondaries. Applications access the user data in files directly from the server on which they are running. Any CFS file system's metadata, however, is only updated by its CFS primary node (the first node to mount the file system). The CFS primary node is responsible for making all metadata updates and for maintaining the file system's metadata update intent log. Other servers can update file system metadata, for example, to allocate new files or delete old ones, by communicating their requests to the primary, which performs the actual updates and responds to the requesting server. This guarantees consistency of file system metadata and the intent log used to recover from system failures.

CFS Failover

If the server on which the CFS primary is running fails, the remaining cluster nodes elect a new primary. The new primary reads the file system intent log and completes any metadata updates that were in process at the time of the failure. Application I/O from other nodes is suspended during the change over, which typically takes a few seconds. When the file system is again consistent, it is mounted automatically by the new primary, and application processing resumes.

Because nodes using a cluster file system in secondary mode do not update file system metadata directly, failure of a secondary node does not require any metadata repair. CFS recovery from secondary node failure is therefore faster than recovery from primary node failure.

CFS and the Group Lock Manager

CFS uses the VERITAS Group Lock Manager (GLM) to reproduce UNIX single-host file system semantics in clusters. This is most important in write behavior. UNIX file systems make writes appear to be atomic. This means that when an application writes a stream of data to a file, any subsequent application that reads from the same area of the file will retrieve the new data, even if it has been cached by the file system and not yet written to disk. Applications can never retrieve stale data, or partial results from a previous write.

To reproduce single-host write semantics, system caches must be kept coherent and each must instantly reflect any updates to cached data, no matter from which cluster node they originate. GLM locks a file so that no other node in the cluster can update it simultaneously, or read it before the update is complete.

Cluster File System Features, Benefits, and Applications

CFS Features

The VERITAS Cluster File System is based on the VERITAS File System (VxFS). Most VxFS features (see “VxFS Features” on page 2) are available on cluster file systems, including:

- Extent based space management that maps files up to a terabyte in size.
- Fast recovery from system crashes using the intent log to track recent file system metadata updates.
- Online administration that allows file systems to be extended and defragmented while they are in use.
- The Quick I/O feature that allows database files to bypass kernel locking by treating files as raw devices. (Licensable only through VERITAS Edition products.)

CFS Benefits

CFS simplifies or eliminates system administration tasks that result from hardware limitations:

- The single system image administrative model of CFS simplifies administration by making all file system management operations, except resizing and reorganization (defragmentation), independent of the location from which they are invoked.
- You can create and manage terabyte-sized volumes, so partitioning file systems to fit within disk limitations is usually not necessary.
- CFS can support file systems with up to a terabyte of capacity, so only extremely large data farms must be partitioned because of file system addressing limitations.
- Because all servers in a cluster have access to CFS cluster-shareable file systems, keeping data consistent across multiple servers is automatic. All cluster nodes have access to the same data, and all data is accessible by all servers using single server file system semantics.
- Because all files can be accessed by all servers, applications can be allocated to servers to balance load or meet other operational requirements. Similarly, failover becomes more flexible because it is not constrained by data accessibility.
- Because each CFS file system can be the primary on any cluster node, the file system recovery portion of failover time in an *n*-node cluster can be reduced by a factor of *n* by distributing the primaryship of file systems uniformly across cluster nodes.
- Enterprise RAID subsystems can be used more effectively because all of their capacity can be mounted by all servers, and allocated by using administrative operations instead of hardware reconfigurations.
- Larger volumes with wider striping improve application I/O load balancing. Not only is the I/O load of each server spread across storage resources, but with CFS shared file systems, the loads of all servers are balanced against each other.
- Extending clusters by adding servers is easier because each new server's storage configuration does not need to be set up—new servers simply adopt the cluster-wide volume and file system configuration.
- The Quick I/O feature that makes file-based databases perform as well as raw partition-based databases is available to applications running in a cluster.

CFS Applications

Almost all applications can benefit from CFS. Applications that are not “cluster-aware” can operate on and access data from anywhere in a cluster. If multiple cluster unaware applications running on different servers are accessing data in a cluster file system, overall system I/O performance improves due to the load balancing effect of having one cluster file system on a separate underlying volume. This is automatic; no tuning or other administrative action is required.

Many applications consist of multiple concurrent threads of execution that could run on different servers if they had a way to coordinate their data accesses. CFS provides this coordination. Such applications can be made cluster-aware allowing their instances to cooperate to balance client and data access load, and thereby scale beyond the capacity of any single server. In such applications, CFS provides shared data access, enabling application-level load balancing across cluster nodes.

- For single-host applications that must be continuously available, CFS can reduce application failover time because it provides an already-running file system environment in which an application can restart after a server failure.
- For parallel applications, such as distributed database management systems and Web servers, CFS provides shared data to all application instances concurrently. CFS also allows these applications to grow by the addition of servers, and improves their availability by enabling them to redistribute load in the event of server failure simply by reassigning network addresses.
- For workflow applications, such as video production, in which very large files are passed from station to station, the CFS eliminates time consuming and error prone data copying by making files available at all stations.
- For backup, the CFS can reduce the impact on operations by running on a separate server, accessing data in cluster-shareable file systems.

The following are examples of applications and how they might work with the CFS:

File Servers

Two or more servers connected in a cluster configuration (that is, connected to the same clients and the same storage) serve separate file systems. If one of the servers fails, the other recognizes the failure, recovers, assumes the primaryship, and begins responding to clients using the failed server’s IP addresses.

Web Servers

Web servers are particularly suitable to shared clustering because their application is typically read-only. Moreover, with a client load balancing front end, a Web server cluster’s capacity can be expanded by adding a server and another copy of the site. A CFS-based cluster greatly simplifies scaling and administration for this type of application.

Quotas 7

Introduction

The VERITAS File System (VxFS) supports user quotas. The quota system limits the use of two principal resources of a file system: files and data blocks. For each of these resources, you can assign quotas to individual users to limit their usage.

The following topics are covered in this chapter:

Quota Limits

You can set limits for individual users to file and data block usage on a file system. You can set two kinds of limits for each of the two resources:

- The *hard limit* is an absolute limit that cannot be exceeded under any circumstances.
- The *soft limit*, which must be lower than the hard limit, can be exceeded, but only for a limited time. The time limit can be configured on a per-file system basis only. The VxFS default limit is seven days.

A typical use of soft limits is when a user must run an application that could generate large temporary files. In this case, you can allow the user to exceed the quota limit for a limited time. No allocations are allowed after the expiration of the time limit. Use the `vxedquota` to set limits (see "Using Quotas" on page 78 for an example).

Although file and data block limits can be set individually for each user, the time limits apply to the file system as a whole. The quota limit information is associated with user IDs and is stored in a quotas file (see "Quotas File on VxFS" below).

The quota soft limit can be exceeded when VxFS preallocates space to a file. See "Attribute Specifics" on page 18 for information on extent allocation policies.

Quotas File on VxFS

A *quotas* file (named `quotas`) must exist in the root directory of a file system for any of the quota commands to work. The `quotas` file in the root directory is referred to as the *external quotas* file. VxFS also maintains an *internal quotas* file for its own use.

The quota administration commands read and write to the external `quotas` file to obtain or change usage limits. VxFS uses the internal `quotas` file to maintain counts of data blocks and inodes used by each user. When quotas are turned on, the quota limits are copied from the external `quotas` file into the internal `quotas` file. While quotas are on, all the changes in the usage information as well as changes to quotas are registered in the internal `quotas` file. When quotas are turned off, the contents of the internal `quotas` file are flushed into the external `quotas` file so that all data is in sync between the two files.]

Quota Commands

Note Most of the quota commands in VxFS (as with UFS) are similar to BSD quota commands. However, the `quotacheck` command is an exception—VxFS does not support an equivalent command. This is discussed in more detail in "Quota Checking With VxFS."

In general, quota administration for VxFS is performed using commands similar to UFS quota commands. On Solaris, the available quota commands are UFS specific (that is, these commands work only on UFS file systems). For this reason, VxFS supports a similar set of commands that work only for VxFS file systems.

VxFS supports the following quota-related commands:

- `vxedquota`—used to edit quota limits for users. The limit changes made by `vxedquota` are reflected both in the internal `quotas` file and the external `quotas` file.
- `vxrepquota`—provides a summary of quotas and disk usage
- `vxquot`—provides file ownership and usage summaries
- `vxquota`—used to view quota limits and usage
- `vxquotaon`—used to turn quotas on for a mounted VxFS file system
- `vxquotaoff`—used to turn quotas off for a mounted VxFS file system

Besides these commands, the VxFS `mount` command supports a special mount option (`-o quota`), which can be used to turn on quotas at mount time.

For additional information on the quota commands, see the corresponding manual pages.

Quota Checking With VxFS

The standard practice with most quota implementations is to mount all file systems and then run a quota check on each one. The quota check reads all the inodes on disk and calculates the usage for each user. This can be time consuming, and because the file system is mounted, the usage can change while quotacheck is running.

VxFS does not support a `quotacheck` command. With VxFS, quota checking is performed automatically (if necessary) at the time quotas are turned on. A quota check is necessary if the file system has changed with respect to the usage information as recorded in the internal quotas file. This happens only if the file system was written with quotas turned off, or if there was structural damage to the file system that required a full file system check (see `fsck_vxfs`(1M)).

A quota check generally reads information for each inode off of the disk and rebuilds the internal quotas file. It is possible that while quotas were not on, quota limits were changed by the system administrator. These changes are stored in the external quotas file. As part of enabling quotas processing, quota limits are read from the external quotas file into the internal quotas file.

Using Quotas

To use the quota functionality on a file system, quotas must be turned on. You can turn them on at mount time or after a file system is mounted.

Note Before turning on quotas, the root directory of the file system must contain a file named `quotas` owned by `root`.

To turn on quotas for a VxFS file system, enter:

```
# vxquotaon /mount_point
```

Quotas can also be turned on for a file system at mount time by specifying the `-o quota` option to the `mount` command:

```
# mount -F vxfs -o quota special /mount_point
```

User quotas can be set up using the `vxedquota` command. You must have superuser privileges to edit quotas:

```
# vxedquota username
```

`vxedquota` creates a temporary file for the given user; this file contains on-disk quotas for each mounted file system that has a quotas file. It is not necessary that quotas be turned on for `vxedquota` to work. However, the quota limits will be applicable only after quotas are turned on for a given file system.

The soft and hard limits can be modified or assigned desired values. For any user, usage can never exceed the hard limit.

Time limits can be modified using the command:

```
# vxedquota -t
```

Modified time limits apply to the entire file system and cannot be set selectively for each user.

The `vxquota` command can be used to view a user's disk quotas and usage on VxFS file systems:

```
# vxquota -v username
```

This displays the user's quotas and disk usage on all mounted VxFS file systems where the `quotas` file exists.

To turn off quotas for a mounted file system, enter:

```
# vxquotaoff /mount_point
```


Storage Checkpoints 8

Storage Checkpoints are a feature of the VERITAS File System (VxFS) that serves as the enabling technology for other VERITAS products: *Block-Level Incremental Backups* and *Storage Rollback*, which are used for backing up databases. VxFS now has an administrative model that allows Storage Checkpoints to be treated like regular file systems so that you can take direct advantage of this technology.

Storage Checkpoints are only available through VERITAS Database Edition products. You can use Storage Checkpoints only on file systems with the Version 4 disk layout; if you are using an older disk layout, you must upgrade your layout to Version 4 to use this feature. It is not advisable to use Storage Checkpoints in environments that use a large number of files and directories. This feature is best suited for file systems containing database files.

The following topics are covered in this chapter:

What is a Storage Checkpoint?

The VERITAS File System provides a unique Storage Checkpoint facility which quickly creates a persistent image of a file system at an exact point in time. Storage Checkpoints significantly reduce I/O overhead by identifying and maintaining only the file system blocks that have changed since the last Storage Checkpoint or backup via a *copy-on-write* technique (see "How a Storage Checkpoint Works" on page 83). Unlike a disk-based mirroring technology that requires a separate storage space, this VERITAS technology minimizes the use of disk space by creating a Storage Checkpoint within the same free space available to the file system.

Storage Checkpoints are data objects which are managed and controlled by the file system; as a result, Storage Checkpoints are persistent across system reboots and crashes. You can create, remove, and rename Storage Checkpoints because they are data objects with associated names (see "Storage Checkpoint Administration" on page 87). After you create a Storage Checkpoint of a mounted file system, you can also continue to create, remove, and update files on the file system without affecting the logical image of the Storage Checkpoint. This technology preserves not only the name space (directory hierarchy) of the file system, but also the user data as it existed at the moment the Storage Checkpoint was taken.

Unlike the VERITAS File System snapshot feature, a Storage Checkpoint:

- Allows write operations to the Storage Checkpoint itself
- Persists after a system reboot or failure
- Shares the same pool of free space as the file system
- Maintains a relationship with other Storage Checkpoints by identifying changed blocks since the last Storage Checkpoint
- Reduces I/O operations and required storage space when dealing with multiple Storage Checkpoints because the latest Storage Checkpoint is the only one that accumulates "before" images.

Various backup and replication solutions can take advantage of Storage Checkpoints. The ability of Storage Checkpoints to track the file system blocks that have changed since the last Storage Checkpoint facilitates backup and replication applications which only need to retrieve the changed data. Storage Checkpoints significantly minimize data movement and may promote higher availability and data integrity by increasing the frequency of backup and replication solutions.

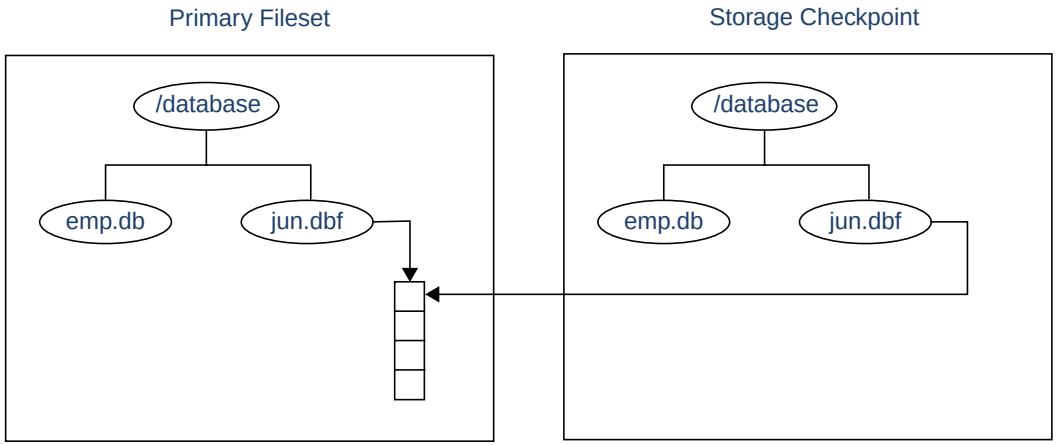
How a Storage Checkpoint Works

The Storage Checkpoint facility freezes the mounted file system (known as the *primary fileset*), initializes the Storage Checkpoint, and thaws the file system. Specifically, the file system is first brought to a stable state where all of its data is written to disk. The *freezing* process that follows momentarily blocks all I/O operations to the file system. A Storage Checkpoint is then created without any actual data; the Storage Checkpoint instead points to the *block map* (described below) of the primary fileset. The *thawing* process that follows restarts I/O operations to the file system.

You can create a Storage Checkpoint on a single file system or a list of file systems. A multiple file system Storage Checkpoint simultaneously freezes the file systems, creates a Storage Checkpoint on all file systems, and thaws the file systems. As a result, the Storage Checkpoints for multiple file systems have the same creation timestamp. The Storage Checkpoint facility guarantees that multiple file system Storage Checkpoints are created on all or none of the specified file systems, unless a system crash takes place while the operation is in progress. The calling application is responsible for cleaning up Storage Checkpoints after a system crash.

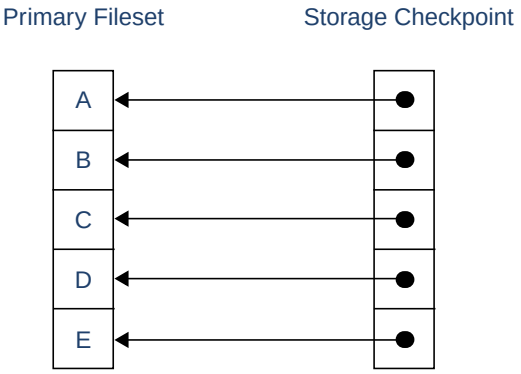
As mentioned above, a Storage Checkpoint of the primary fileset initially contains a pointer to the file system block map rather than any actual data. The block map points to the data on the primary fileset. Figure 2 shows the file system `/database` and its Storage Checkpoint. The Storage Checkpoint is logically identical to the primary fileset when the Storage Checkpoint is created, but does not contain any actual data blocks.

Figure 2. Primary Fileset and Its Storage Checkpoint



In Figure 3, each block of the file system is represented by a square. Similar to Figure 2, this figure shows a Storage Checkpoint containing pointers to the primary fileset at the time the Storage Checkpoint is taken.

Figure 3. Initializing a Storage Checkpoint

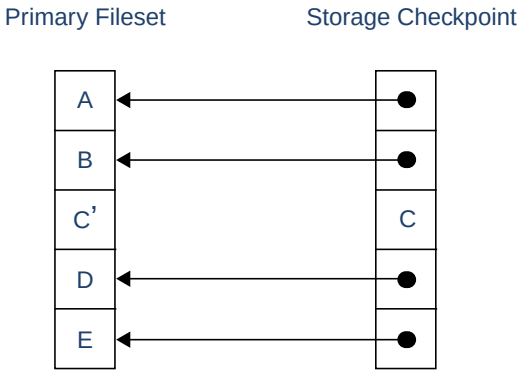


The Storage Checkpoint presents the exact image of the file system by finding the data from the primary fileset. As the primary fileset is updated, the original data is copied to the Storage Checkpoint before the new data is written. When a write operation changes a specific data block in the primary fileset, the old data is first read and copied to the Storage Checkpoint before the primary fileset is updated. Subsequent writes to the specified data block on the primary fileset do not result in additional updates to the Storage Checkpoint because the old data needs to be saved only once. As blocks in the primary fileset continue to change, the Storage Checkpoint accumulates the original data blocks.

In Figure 4, the third block originally containing C is updated. Before the block is updated with new data, the original data is copied to the Storage Checkpoint. This is called the *copy-on-write* technique, which allows the Storage Checkpoint to preserve the image of the primary fileset when the Storage Checkpoint is taken.

Every update or write operation does not necessarily result in the process of copying data to the Storage Checkpoint. In this example, subsequent updates to this block, now containing C', are not copied to the Storage Checkpoint because the original image of the block containing C is already saved.

Figure 4. Updates to the Primary Fileset



98

Types of Storage Checkpoints

You can create the following types of Storage Checkpoints:

- Data Storage Checkpoints
- Nodata Storage Checkpoints
- Removable Storage Checkpoints
- Non-mountable Storage Checkpoints

Data Storage Checkpoints

A data Storage Checkpoint is a complete image of the file system at the time the Storage Checkpoint is created. This type of Storage Checkpoint contains the file system metadata *and* file data blocks. You can mount, access, and write to a data Storage Checkpoint just as you would to a file system. Data Storage Checkpoints are useful for backup applications which require a consistent and stable image of an active file system. Data Storage Checkpoints introduce some overhead to the system and to the application performing the write operation. For best results, you can limit the life of data Storage Checkpoints to minimize the impact on system resources.

Nodata Storage Checkpoints

A nodata Storage Checkpoint only contains file system metadata; this type of Storage Checkpoint does not contain any file data blocks. As the original file system changes, the nodata Storage Checkpoint records the location of every changed block. Nodata Storage Checkpoints use minimal system resources and have little impact on the performance of the file system because the data itself does not have to be copied.

Removable Storage Checkpoints

A removable Storage Checkpoint can "self-destruct" under certain conditions when the file system runs out of space (see "Space Management Considerations" on page 98 for more information). After encountering certain ENOSPC conditions, the kernel removes Storage Checkpoints to free up space for the application to continue running on the file system. In almost all situations, you should create Storage Checkpoints with the removable attribute.

Non-mountable Storage Checkpoints

A non-mountable Storage Checkpoint cannot be mounted. You can use this type of Storage Checkpoint as a security feature which prevents other applications from accessing the Storage Checkpoint and modifying it.

Storage Checkpoint Administration

Storage Checkpoint administrative operations require the `fsckptadm` utility (see `fsckptadm`(1M)). You can use the `fsckptadm` utility to create and remove Storage Checkpoints, change attributes, and ascertain statistical data. Every Storage Checkpoint has an associated name which allows you to manage Storage Checkpoints; this name is limited to 127 characters and cannot contain a colon (`:`).

Creating a Storage Checkpoint

You can create a Storage Checkpoint using the `fsckptadm` utility. In these examples, `/mnt0` is a mounted VxFS file system with a Version 4 disk layout.

This example shows the creation of a nodata Storage Checkpoint (see "Space Management Considerations" on page 98) named `thu_7pm` on `/mnt0` and lists all Storage Checkpoints of the `/mnt0` file system:

```
#  fsckptadm -n create thu_7pm /mnt0
#  fsckptadm list /mnt0
/mnt0
thu_7pm:
   ctime     = Thu Jun 1 19:02:17 2000
   mtime     = Thu Jun 1 19:02:17 2000
   flags     = nodata
```

This example shows the creation of a removable Storage Checkpoint named `thu_8pm` on `/mnt0` and list all Storage Checkpoints of the `/mnt0` file system:

```
#  fsckptadm -r create thu_8pm /mnt0
#  fsckptadm list /mnt0
/mnt0
thu_8pm:
   ctime     = Thu Jun 1 20:01:19 2000
   mtime     = Thu Jun 1 20:01:19 2000
   flags     = removable
thu_7pm:
   ctime     = Thu Jun 1 19:02:17 2000
   mtime     = Thu Jun 1 19:02:17 2000
   flags     = nodata
```


Removing a Storage Checkpoint

You can delete a Storage Checkpoint by specifying the `remove` keyword of the `fsckptadm` command. Specifically, you can use either the *synchronous* or *asynchronous* method of removing a Storage Checkpoint; the asynchronous method is the default method. The synchronous method entirely removes the Storage Checkpoint and returns all of the blocks to the file system before completing the `fsckptadm` operation. The asynchronous method simply marks the Storage Checkpoint for removal and causes `fsckptadm` to return immediately. At a later time, an independent kernel thread completes the removal operation and releases the space used by the Storage Checkpoint.

In this example, `/mnt0` is a mounted VxFS file system with a Version 4 disk layout. This example shows the asynchronous removal of the Storage Checkpoint named `thu_8pm` and synchronous removal of the Storage Checkpoint named `thu_7pm`. This example also lists all the Storage Checkpoints remaining on the `/mnt0` file system after the specified Storage Checkpoint removal:

```
#  fsckptadm remove thu_8pm /mnt0
#  fsckptadm list /mnt0
/mnt0
thu_7pm:
   ctime     = Thu Jun 1 19:02:17 2000
   mtime     = Thu Jun 1 19:02:17 2000
   flags     = nodata
# fsckptadm -s remove thu_7pm /mnt0
# fsckptadm list /mnt0
/mnt0
```

Accessing a Storage Checkpoint

You can mount Storage Checkpoints using the `mount` command (see `mount_vxfs`(1M)) with the mount option `–o ckpt=ckpt_name`. Observe the following rules when mounting Storage Checkpoints:

- Storage Checkpoints are mounted as read-only Storage Checkpoints by default. If you need to write to a Storage Checkpoint, mount it using the `-o rw` option.
- If a Storage Checkpoint is originally mounted as a read-only Storage Checkpoint, you can remount it as a writable Storage Checkpoint using the `-o remount` option.
- To mount a Storage Checkpoint of a file system, first mount the file system itself.
- To unmount a file system, first unmount all of its Storage Checkpoints.

Caution If you create a Storage Checkpoint for backup purposes, do not mount it as a writable Storage Checkpoint. You will lose the point-in-time image if you accidently write to the Storage Checkpoint.

A Storage Checkpoint is mounted on a special *pseudo device*. This pseudo device does not exist in the system name space; the device is internally created by the system and used while the Storage Checkpoint is mounted. The pseudo device is removed after you unmount the Storage Checkpoint. A pseudo device name is formed by appending the Storage Checkpoint name to the file system device name using the colon character (`:`) as the separator.

For example, if a Storage Checkpoint named `may_23` belongs to the file system residing on the special device `/dev/vx/dsk/fsvol/vol1`, the Storage Checkpoint pseudo device name is:

```
/dev/vx/dsk/fsvol/vol1:may_23
```

To mount the Storage Checkpoint named `may_23` as a read-only (default) Storage Checkpoint on directory `/fsvol_may_23`, type:

```
# mount -F vxfs -o ckpt=may_23 /dev/vx/dsk/fsvol/vol1:may_23 \
 /fsvol_may_23
```

The `/fsvol` file system must already be mounted before the Storage Checkpoint can be mounted. To remount the Storage Checkpoint named `may_23` as a writable Storage Checkpoint, type:

```
# mount -F vxfs -o ckpt=may_23,remount,rw \
/dev/vx/dsk/fsvol/vol1:may_23 /fsvol_may_23
```


To automatically mount this Storage Checkpoint when the system starts up, put the following entries in the `/etc/vfstab` file:

#device to mount	device to fsck	mount point	FS type	fsck pass	mount at boot	mount options
/dev/vx/dsk/fsvol/vol1	/dev/vx/rdsk/fsvol/vol1	/fsvol	vxfs	1	yes	—
/dev/vx/dsk/fsvol/vol1:may_23	—	/fsvol_may_23	vxfs	0	yes	`ckpt=may_23`

To mount a Storage Checkpoint of a cluster file system, you must also use the `-o cluster` option:

```
# mount -F vxfs -o cluster,ckpt=may_23 \
/dev/vx/dsk/fsvol/vol1:may_23 /fsvol_may_23
```

You can only mount a Storage Checkpoint clusterwide if the file system that the Storage Checkpoint belongs to is also mounted clusterwide. Similarly, you can only mount a Storage Checkpoint locally if the file system that the Storage Checkpoint belongs to is mounted locally.

You can unmount Storage Checkpoints using the `umount` command (see `umount`(1M)). Storage Checkpoints can be unmounted by the mount point or pseudo device name:

```
# umount /fsvol_may_23
# umount /dev/vx/dsk/fsvol/vol1:may_23
```

Note You do not need to run the `fsck` utility on a Storage Checkpoint pseudo device because this utility runs on the actual file system.

Converting a Data Storage Checkpoint to a Nodata Storage Checkpoint

A nodata Storage Checkpoint does not contain actual file data. Instead, this type of Storage Checkpoint contains a collection of markers indicating the location of all the changed blocks since the Storage Checkpoint was created (see “Types of Storage Checkpoints” on page 86 for more information).

You can use either the *synchronous* or *asynchronous* method of convert a data Storage Checkpoint to a nodata Storage Checkpoint; the asynchronous method is the default method. In a synchronous conversion, `fsckptadm` waits for all files to undergo the conversion process to “nodata” status before completing the operation. In an asynchronous conversion, `fsckptadm` returns immediately and marks the Storage Checkpoint as a nodata Storage Checkpoint even though the Storage Checkpoint’s data blocks are not immediately returned to the pool of free blocks in the file system. The Storage Checkpoint deallocates all of its file data blocks in the background and eventually returns them to the pool of free blocks in the file system.

If all of the older Storage Checkpoints in a file system are nodata Storage Checkpoints, use the synchronous method to convert a data Storage Checkpoint to a nodata Storage Checkpoint. If an older data Storage Checkpoint exists in the file system, use the asynchronous method to mark the Storage Checkpoint you want to convert for a delayed conversion. In this case, the actual conversion will continue to be delayed until the Storage Checkpoint becomes the oldest Storage Checkpoint in the file system, or all of the older Storage Checkpoints have been converted to nodata Storage Checkpoints.

Note You cannot convert a nodata Storage Checkpoint to a data Storage Checkpoint because a nodata Storage Checkpoint only keeps track of the location of block changes and does not save the content of file data blocks.

Difference Between a Data and a Nodata Storage Checkpoint

The following example shows the difference between data Storage Checkpoints and nodata Storage Checkpoints:

1. Create a file system and mount it on `/mnt0`:

```
#  mkfs -F vxfs /dev/vx/rdsk/test0
version 4 layout
11845780 sectors, 5922890 blocks of size 1024, log size 1024
blocks unlimited inodes, largefiles not supported
5922890 data blocks, 5920314 free data blocks
181 allocation units of 32768 blocks, 32768 data blocks
last allocation unit has 24650 data blocks
#  mount -F vxfs /dev/vx/dsk/test0 /mnt0
```

2. Create a small file with a known content. Create a Storage Checkpoint and mount it on `/mnt0@5_30pm`:

```
#  echo "hello, world" > /mnt0/file
#  fsckptadm create ckpt@5_30pm /mnt0
#  mkdir /mnt0@5_30pm
#  mount -F vxfs -o ckpt=ckpt@5_30pm \
   /dev/vx/dsk/test0:ckpt@5_30pm /mnt0@5_30pm
```

3. Examine the content of the original file and the Storage Checkpoint file:

```
#  cat /mnt0/file
hello, world
#  cat /mnt0@5_30pm/file
hello, world
```

4. Change the content of the original file:

```
#  echo "goodbye!" > /mnt0/file
```

5. Examine the content of the original file and the Storage Checkpoint file. The original file contains the latest data while the Storage Checkpoint file still contains the data at the time of the Storage Checkpoint creation:

```
#  cat /mnt0/file
goodbye!
#  cat /mnt0@5_30pm/file
hello, world
```

6. Unmount the Storage Checkpoint, convert the Storage Checkpoint to a nodata Storage Checkpoint, and mount the Storage Checkpoint again.

```
# umount /mnt0@5_30pm
# ./fsckptadm -s set nodata ckpt@5_30pm /mnt0
# mount -F vxfs -o ckpt=ckpt@5_30pm \
   /dev/vx/dsk/test0:ckpt@5_30pm /mnt0@5_30pm
```

7. Examine the content of both files. The original file must contain the latest data:

```
#  cat /mnt0/file
goodbye!
```

You can traverse and read the directories of the nodata Storage Checkpoint; however, the files contain no data but only markers to indicate which block of the file has been changed since the Storage Checkpoint was created:

```
#  ls -l /mnt0@5_30pm/file
-rw-r--r--   1 root     other 9 Jul 13 17:13 /mnt0@5_30pm/file
#  cat /mnt0@5_30pm/file
cat: input error on /mnt0@5_30pm/file: I/O error
```


Conversion with Multiple Storage Checkpoints

The following example highlights the conversion of data Storage Checkpoints to nodata Storage Checkpoints, particularly when dealing with older Storage Checkpoints on the same file system:

1. Create a file system and mount it on `/mnt0`:

```
#  mkfs -F vxfs /dev/vx/rdsk/test0
version 4 layout
4194304 sectors, 2097152 blocks of size 1024,
log size 1024 blocks
unlimited inodes, largefiles not supported
2097152 data blocks, 2095536 free data blocks
64 allocation units of 32768 blocks, 32768 data blocks
#  mount -F vxfs /dev/vx/dsk/test0 /mnt0
```

2. Create four data Storage Checkpoints on this file system, note the order of creation, and list them:

```
#  fsckptadm create oldest /mnt0
#  fsckptadm create older /mnt0
#  fsckptadm create old /mnt0
#  fsckptadm create latest /mnt0
#  fsckptadm list /mnt0
/mnt0
latest:
   ctime                =Mon Oct 16 11:56:55 2000
   mtime                =Mon Oct 16 11:56:55 2000
   flags                =none
old:
   ctime                =Mon Oct 16 11:56:51 2000
   mtime                =Mon Oct 16 11:56:51 2000
   flags                =none
older:
   ctime                =Mon Oct 16 11:56:46 2000
   mtime                =Mon Oct 16 11:56:46 2000
   flags                =none
oldest:
   ctime                =Mon Oct 16 11:56:41 2000
   mtime                =Mon Oct 16 11:56:41 2000
   flags                =none
```

3. Try to convert synchronously the “latest” Storage Checkpoint to a nodata Storage Checkpoint. The attempt will fail because the Storage Checkpoints older than the “latest” Storage Checkpoint are data Storage Checkpoints, namely the Storage Checkpoint “old”:

```
# fsckptadm -s set nodata latest /mnt0
vxfs fsckptadm: checkpoint set failed on latest,
File exists (17)
```

4. You can instead convert the “latest” Storage Checkpoint to a nodata Storage Checkpoint in a delayed or asynchronous manner. If you list the Storage Checkpoints, you will see that the “latest” Storage Checkpoint is marked for conversion in the future:

```
# fsckptadm set nodata latest /mnt0
# fsckptadm list /mnt0
/mnt0
latest:
   ctime                =Mon Oct 16 11:56:55 2000
   mtime                =Mon Oct 16 11:56:55 2000
   flags                =nodata, delayed
old:
   ctime                =Mon Oct 16 11:56:51 2000
   mtime                =Mon Oct 16 11:56:51 2000
   flags                =none
older:
   ctime                =Mon Oct 16 11:56:46 2000
   mtime                =Mon Oct 16 11:56:46 2000
   flags                =none
oldest:
   ctime                =Mon Oct 16 11:56:41 2000
   mtime                =Mon Oct 16 11:56:41 2000
   flags                =none
```


5. You can combine the two previous steps and create the “latest” Storage Checkpoint as a nodata Storage Checkpoint. The creation process will detect the presence of the older data Storage Checkpoints and create the “latest” Storage Checkpoint as a delayed nodata Storage Checkpoint. First remove the “latest” Storage Checkpoint:

```
# fsckptadm remove latest /mnt0
# fsckptadm list /mnt0
/mnt0
old:
   ctime                =Mon Oct 16 11:56:51 2000
   mtime                =Mon Oct 16 11:56:51 2000
   flags                =none
older:
   ctime                =Mon Oct 16 11:56:46 2000
   mtime                =Mon Oct 16 11:56:46 2000
   flags                =none
oldest:
   ctime                =Mon Oct 16 11:56:41 2000
   mtime                =Mon Oct 16 11:56:41 2000
   flags                =none
```

Then recreate it as a nodata Storage Checkpoint:

```
# fsckptadm -n create latest /mnt0
# fsckptadm list /mnt0
/mnt0
latest:
   ctime                =  Mon Oct 16 12:06:42 2000
   mtime                =  Mon Oct 16 12:06:42 2000
   flags                =  nodata, delayed
old:
   ctime                =  Mon Oct 16 11:56:51 2000
   mtime                =  Mon Oct 16 11:56:51 2000
   flags                =  none
older:
   ctime                =  Mon Oct 16 11:56:46 2000
   mtime                =  Mon Oct 16 11:56:46 2000
   flags                =  none
oldest:
   ctime                =  Mon Oct 16 11:56:41 2000
   mtime                =  Mon Oct 16 11:56:41 2000
   flags                =  none
```

6. You can synchronously convert the “oldest” Storage Checkpoint to a nodata Storage Checkpoint because it is the oldest Storage Checkpoint in the file system:

```
# fsckptadm -s set nodata oldest /mnt0
# fsckptadm list /mnt0
/mnt0
latest:
   ctime                =Mon Oct 16 12:06:42 2000
   mtime                =Mon Oct 16 12:06:42 2000
   flags                =nodata, delayed
old:
   ctime                =Mon Oct 16 11:56:51 2000
   mtime                =Mon Oct 16 11:56:51 2000
   flags                =none
older:
   ctime                =Mon Oct 16 11:56:46 2000
   mtime                =Mon Oct 16 11:56:46 2000
   flags                =none
oldest:
   ctime                =Mon Oct 16 11:56:41 2000
   mtime                =Mon Oct 16 11:56:41 2000
   flags                =nodata
```

7. Remove the “older” and “old” Storage Checkpoints. After you remove the “old” Storage Checkpoint, the “latest” Storage Checkpoint is automatically converted to a nodata Storage Checkpoint because the only remaining older Storage Checkpoint (“oldest”) is already a nodata Storage Checkpoint:

```
# fsckptadm remove older /mnt0
# fsckptadm remove old /mnt0
# fsckptadm list /mnt0
/mnt0
latest:
   ctime                =Mon Oct 16 12:06:42 2000
   mtime                =Mon Oct 16 12:06:42 2000
   flags                =nodata
oldest:
   ctime                =Mon Oct 16 11:56:41 2000
   mtime                =Mon Oct 16 11:56:41 2000
   flags                =nodata
```


Space Management Considerations

Several operations, such as the removal of a file or the overwriting of an existing file, can fail when a file system containing Storage Checkpoints runs out of space. Traditionally, such failures do not stem from the amount of space available on the file system; however, these operations on a file system containing Storage Checkpoints may cause a data block copy which, in turn, may require extent allocation. If the system cannot allocate sufficient space, the operation will fail.

Database applications usually preallocate storage for their files and may not expect a write operation to fail. If a file system runs out of space, the kernel automatically removes Storage Checkpoints and attempts to complete the write operation after sufficient space becomes available. The kernel removes Storage Checkpoints to prevent commands, such as `rm` (see `rm`(1)), from failing under an ENOSPC condition.

The kernel will follow these policies when automatically removing Storage Checkpoints:

1. Remove as few Storage Checkpoints as possible to complete the operation.
2. Never select a non-removable Storage Checkpoint.
3. Select a nodata Storage Checkpoint only when data Storage Checkpoints no longer exist.
4. Remove the oldest Storage Checkpoint first.

Quick I/O for Databases 9

Introduction

VERITAS Quick I/O for Databases (referred to as Quick I/O) lets applications access preallocated VxFS files as raw character devices. This provides the administrative benefits of running databases on file systems without the performance degradation usually associated with databases created on file systems.

Quick I/O is a separately licensable feature available from VERITAS as part of VERITAS Editions products.

Topics covered in this chapter:

Quick I/O Functionality and Performance

Many database administrators (DBAs) create databases on file systems because it makes common administrative tasks (such as moving, copying, and backup) much simpler. However, putting databases on file systems significantly reduces database performance. By using VERITAS Quick I/O, you can retain the advantages of having databases on file systems without performance degradation.

Quick I/O uses a special naming convention to allow database applications to access regular files as raw character devices. This provides higher database performance in the following ways:

- Supporting kernel asynchronous I/O
- Supporting direct I/O
- Avoiding kernel write locks
- Avoiding double buffering

Supporting Kernel Asynchronous I/O

Some operating systems provide kernel support for asynchronous I/O on raw devices, but not on regular files. As a result, even if the database server is capable of using asynchronous I/O, it cannot issue asynchronous I/O requests when the database is built on a file system. Lack of asynchronous I/O significantly degrades performance. Quick I/O lets the database server take advantage of kernel supported asynchronous I/O on file system files accessed via the Quick I/O interface by providing a character device node that is treated by the OS as a raw device.

Supporting Direct I/O

I/O on files using `read`() and `write`() system calls typically results in data being copied twice: once between user and kernel space, and later between kernel space and disk. In contrast, I/O on raw devices is direct. That is, data is copied directly between user space and disk, saving one level of copying. As with I/O on raw devices, Quick I/O avoids the extra copying.

Avoiding Kernel Write Locks

When database I/O is performed via the `write`() system call, each system call acquires and releases a write lock inside the kernel. This lock prevents simultaneous write operations on the same file. Because database systems usually implement their own locks for managing concurrent access to files, write locks unnecessarily serialize I/O operations. Quick I/O bypasses file system locking and lets the database server control data access.

Avoiding Double Buffering

Most database servers implement their own buffer cache and do not need the system buffer cache. So the memory used by the system buffer cache is wasted, and results in data being cached twice: first in the database cache and then in the system buffer cache. By using direct I/O, Quick I/O does not waste memory on double buffering. This frees up memory that can then be used by the database server buffer cache, leading to increased performance.

Using VxFS Files as Raw Character Devices

When VxFS with Quick I/O is installed, there are two ways of accessing a file:

- The VxFS interface treats the file as a regular VxFS file
- The Quick I/O interface treats the same file as if it were a raw character device, having performance similar to a raw device

This allows a database server to use the Quick I/O interface while a backup server uses the VxFS interface.

Quick I/O Naming Convention

To treat a file as a raw character device, Quick I/O requires a file name extension to create an alias for a regular VxFS file. Quick I/O recognizes the alias when you add the following suffix to a file name:

```
::cdev:vxfs:
```

Whenever an application opens an existing VxFS file with the suffix `::cdev:vxfs` (the *cdev* portion is an acronym for *character device*), Quick I/O treats the file as if it were a raw device. For example, if the file *xxx* is a regular VxFS file, then an application can access *xxx* as a raw character device by opening it with the name:

xxx`::cdev:vxfs:`

Note When Quick I/O is enabled, you cannot create a regular VxFS file with a name that uses the `::cdev:vxfs:` extension. If an application tries to create a regular file named *xxx*`::cdev:vxfs:`, the create fails. If Quick I/O is not available, it is possible to create a regular file with the `::cdev:vxfs:` extension, but this could cause problems if Quick I/O is later enabled. It is advisable to reserve the extension only for Quick I/O files.

Use Restrictions

There are restrictions to using regular VxFS files as Quick I/O files.

1. The name *xxx*`::cdev:vxfs:` is recognized as a special name by VxFS only when:
   a. the `qio` module is loaded
   b. Quick I/O has a valid license
   c. the regular file *xxx* is physically present on the VxFS file system
   d. there is no regular file named *xxx*`::cdev:vxfs:` on the system
2. If the file *xxx* is being used for memory mapped I/O, it cannot be accessed as a Quick I/O file.
3. An I/O fails if the file *xxx* has a logical hole and the I/O is done to that hole on *xxx*`::cdev:vxfs:`.
4. The size of the file cannot be extended by writes through the Quick I/O interface.

Creating a Quick I/O File Using qiomkfile

The best way to make regular files accessible to the Quick I/O interface and preallocate space for them is to use the `qiomkfile` command. Unlike the VxFS `setext` command, which requires superuser privileges, any user who has read/write permissions can run `qiomkfile` to create the files. The `qiomkfile` command has five options:

`-a`	Creates a symbolic link with an absolute path name for a specified file. The default is to create a symbolic link with a relative path name.
`-e`	(For Oracle database files to allow tablespace resizing.) Extends the file size *by* the specified amount.
`-h`	(For Oracle database files.) Creates a file with additional space allocated for the Oracle header.
`-r`	(For Oracle database files to allow tablespace resizing.) Increases the file *to* the specified size.
`-s`	Preallocates space for a file.

You can specify file size in terms of bytes (the default), or in kilobytes, megabytes, gigabytes, or sectors (512 bytes) by adding a `k`, `K`, `m`, `M`, `g`, `G`, `s` or `S` suffix. If the size of file including the header is not a multiple of the file system block size, it is rounded to a multiple of the file system block size before preallocation.

`qiomkfile` creates two files: a regular file with preallocated, contiguous space; and a symbolic link pointing to the Quick I/O name extension. For example, to create a 100 MB file named `dbfile` in `/database`, enter:

```
$ qiomkfile -s 100m /database/dbfile
```

In this example, the first file created is a regular file named `/database/.dbfile` (which has the real space allocated).

The second file is a symbolic link named `/database/dbfile`. This is a relative link to `/database/.dbfile` via the Quick I/O interface, that is, to `.dbfile::cdev:vxfs:`. This allows `.dbfile` to be accessed by any database or application as a raw character device. To check the results, enter:

```
$ ls -al
-rw-r--r--   1 oracle   dba 104857600 Oct 22 15:03 .dbfile
lrwxrwxrwx   1 oracle   dba 19 Oct 22 15:03 dbfile -> \
                        .dbfile::cdev:vxfs:
```

or:

```
$ ls -lL
crw-r----- 1    oracle   dba 43,0 Oct 22 15:04 dbfile
-rw-r--r-- 1    oracle   dba 10485760 Oct 22 15:04 .dbfile
```

If you specify the `-a` option, an absolute path name (see "Using Absolute or Relative Path Names" on page 104) is used so `/database/dbfile` points to `/database/.dbfile::cdev:vxfs:`. To check the results, enter:

```
$ ls -al
-rw-r--r--   1 oracle   dba 104857600 Oct 22 15:05 .dbfile
lrwxrwxrwx   1 oracle   dba 31 Oct 22 15:05 dbfile ->
                       /database/.dbfile::cdev:vxfs:
```

See the `qiomkfile`(1) manual page for more information.

Accessing Regular VxFS Files Through Symbolic Links

Another way to use Quick I/O is to create a symbolic link for each file in your database and use the symbolic link to access the regular files as Quick I/O files.

The following commands create a 100 MB Quick I/O file named `dbfile` on the VxFS file system `/database`. The `dd` command preallocates the file space:

```
$ cd /database
$ dd if=/dev/zero of=/database/.dbfile bs=128k count=800
$ ln -s .dbfile::cdev:vxfs: /database/dbfile
```

Any database or application can then access the file `dbfile` as a raw character device. See the *VERITAS Database Edition for Oracle Database Administrator's Guide* for more information.

Using Absolute or Relative Path Names

It is usually better to use relative path names instead of absolute path names when creating symbolic links to access regular files as Quick I/O files. Using relative path names prevents copies of the symbolic link from referring to the original file. This is important if you are backing up or moving database files with a command that preserves the symbolic link. However, some applications, such as SAP, require absolute path names.

If you create a symbolic link using a relative path name, both the symbolic link and the file are under the same parent directory. If you want to relocate the file, both the file and the symbolic link must be moved.

It is also possible to use the absolute path name when creating a symbolic link. If the database file is relocated to another directory, however, you must change the symbolic link to use the new absolute path. You can put all the symbolic links in a directory separate from the data directories. For example, you may create a directory named `/database` and put in all the symbolic links, with the symbolic links pointing to absolute path names.

Preallocating Files Using the setext Command

You can use the VxFS `setext` command to preallocate file space, however, the `setext` command requires superuser privileges. You may need to use the `chown` and `chgrp` commands to change the owner and group permissions on the file after it is created. The following example shows how to use `setext` to create a 100 MB database file for an Oracle database:

```
# cd /database
# touch /database/dbfile
# setext -r 102400 -f noreserve -f chgsize /database/.dbfile
# ln -s .dbfile::cdev:vxfs: /database/dbfile
# chown oracle /database/dbfile
# chgrp dba /database/dbfile
```

See the `setext`(1) manual page for more information.

Using Quick I/O with Oracle Databases

The following example shows how a file can be used by an Oracle database to create a tablespace. This command would be run by the Oracle DBA (typically user ID `oracle`):

```
$ qiomkfile -h -s 100m /database/dbfile
$ svrmgrl
SVRMGR> connect internal
SVRMGR> create tablespace ts1
SVRMGR> datafile '/database/dbfile' size 100M;
SVRMGR> exit;
```

The following example shows how the file can be used by an Oracle database to create a tablespace. Oracle requires additional space for one Oracle header size. So in this example, although 100 MB was allocated to `/database/dbfile`, the Oracle database can use only up to 100 MB minus the Oracle parameter `db_block_size`.

```
$ svrmgrl
SVRMGR> connect internal
SVRMGR> create tablespace ts1
SVRMGR> datafile '/database/dbfile' size 99M;
SVRMGR> exit;
```


Using Quick I/O with Sybase Databases

Quick I/O works similarly on Sybase database devices.

To create a new database device, preallocate space on the file system by using the `qiomkfile` command, then use the Sybase `buildmaster` command for a master device, or the Transact SQL `disk init` command for a database device. `qiomkfile` creates two files: a regular file using preallocated, contiguous space, and a symbolic link pointing to the `::cdev:vxfs:` name extension. For example, to create a 100 megabyte master device `masterdev` on the file system `/sybmaster`, enter:

```
$ cd /sybmaster
$ qiomkfile -s 100m masterdev
```

You can use this master device while running the `sybsetup` program or `sybinit` script. If you are creating the master device directly, type:

```
$ buildmaster -d masterdev -s 51200
```

To add a new 500 megabyte database device `datadev` to the file system `/sybdata` on your dataserver, enter:

```
$ cd /sybdata
$ qiomkfile -s 500m datadev
...
$ isql -U sa -P sa_password -S dataserver_name
1> disk init
2> name = "logical_name",
3> physname = "/sybdata/datadev",
4> vdevno = "device_number",
5> size = 256000
6> go
```

Enabling and Disabling Quick I/O

If the Quick I/O feature is licensed and installed, Quick I/O is enabled by default when a file system is mounted. Alternatively, the VxFS `mount -o qio` command enables Quick I/O. The `mount -o noqio` command disables Quick I/O.

If Quick I/O is not installed or licensed, a file system mounts by default without Quick I/O and no error message is displayed. However, if you specify the `-o qio` option, the `mount` command prints the following error message and terminates without mounting the file system.

```
VxFDD: You don't have a license to run this program
vxfs mount: Quick I/O not available
```

Cached Quick I/O For Databases

On 32-bit operating systems, databases can use a maximum of only 4 GB of memory because of the 32-bit address limitation. The Cached Quick I/O feature improves database performance on machines with sufficient memory by also using the file system cache to store data.

For read operations through the Quick I/O interface, data is cached in the system page cache, so subsequent reads of the same data can access this cached copy and avoid doing disk I/O. To maintain the correct data in its buffer for write operations, Cached Quick I/O keeps the page cache in sync with the data written to disk.

On 64-bit systems, where limited memory is not a critical problem, using the file system cache still provides performance benefits by using this *read-ahead* functionality. Because of the read-ahead functionality, sequential table scans will benefit the most from using Cached Quick I/O by significantly reducing the query response time.

To use this feature, set the `qio_cache_enable` system parameter with the `vxtunefs` utility, and use the `qioadmin` command to turn the per-file cache advisory on or off. See the `vxtunefs`(1M) and `qioadmin`(1) man pages for more information.

Enabling Cached Quick I/O

Caching for Quick I/O files can be enabled online when the database is running. You enable caching in two steps:

1. Set the `qio_cache_enable` parameter of `vxtunefs` to enable caching on a file system.

2. Enable the Cached Quick I/O feature for specific files using the `qioadmin` command.

Note Quick I/O must be enabled on the file system for Cached Quick I/O to operate.

Enabling Cached Quick I/O for File Systems

Caching is initially disabled on a file system. You enable Cached Quick I/O for a file system by setting the `qio_cache_enable` option of the `vxtunefs` command after the file system is mounted. For example, to enable Cached Quick I/O for the file system `/database01`, enter:

```
# vxtunefs -s -o qio_cache_enable=1 /database01
```

where `/database01` is a VxFS file system containing the Quick I/O files.

Note This command enables caching for all the Quick I/O files on this file system.

You can make this setting persistent across mounts by adding a file system entry in the file `/etc/vx/tunefstab`. For example:

```
/dev/vx/dsk/datadg/database01 qio_cache_enable=1
/dev/vx/dsk/datadg/database02 qio_cache_enable=1
```

For information on how to add tuning parameters, see the `tunefstab`(4) manual page.

Enabling Cached Quick I/O for Individual Files

There are several ways to enable caching for a Quick I/O file. Use the following syntax to enable caching on an individual file:

```
$ qioadmin -S filename=on mount_point
```

To enable caching for the Quick I/O file `/database01/names.dbf`, type:

```
$ qioadmin -S names.dbf=ON /database01
```

To disable the caching for that file, enter:

```
$ qioadmin -S names.dbf=OFF /database01
```

To make the setting persistent across mounts, create a *qiotab* file, `/etc/vx/qioadmin`, to list files and their caching advisories. Based on the following example, the file `/database/sell.dbf` will have caching turned on whenever the file system `/database` is mounted:

```
device=/dev/vx/dsk/datadg/database01
dates.dbf,off
names.dbf,off
sell.dbf,on
```

Note The cache advisories operate only if Cached Quick I/O is enabled for the file system. If the `qio_cache_enable` flag is zero, Cached Quick I/O is `OFF` for all the files in that file system even if the individual file cache advisory for a file is `ON`.

To check on the current cache advisory settings for a file, enter:

```
$ qioadmin -P names.dbf /database01
    names.dbf,OFF
```

To check the setting of the `qio_cache_enable` flag for a file system, enter:

```
$ vxtunefs -p /database01
    qio_cache_enable = 1
```

For more information on the format of the `/etc/vx/qioadmin` file and the command syntax, see the `qioadmin`(1) manual page.

Note Check the setting of the flag `qio_cache_enable` using the `vxtunefs` command, and the individual cache advisories for each file, to verify caching.

Tuning Cached Quick I/O

Not all database files can take advantage of caching. Performance may even degrade in some instances (due to double buffering, for example). Determining which files and applications can benefit from Cached Quick I/O requires that you first collect and analyze the caching statistics.

See the `qiostat`(1) man page for information on gathering statistics, and the *VERITAS Database Edition for Oracle Database Administrator's Guide* for a description of the Cached Quick I/O tuning methodology.

Quick I/O Statistics

Quick I/O provides the `qiostat` utility to collect database I/O statistics generated over a period of time. `qiostat` reports statistics such as the number of read and write operations, the number of blocks read or written, and the average time spent on read and write operations during an interval. See the `qiostat`(1) manual page for more information.

Quick I/O Summary

To increase database performance on a VxFS file system using Quick I/O:

1. Make sure that the Quick I/O module is loaded.

   ```
   # modinfo | grep fdd
   ```

2. You can add the following line to the file `/etc/system` to load Quick I/O whenever the system reboots.

   ```
   forceload: drv/fdd
   ```

3. Create a regular VxFS file and preallocate it to the required size. The size of this preallocation depends on the size requirement of the database server.

4. Create and access the database using the file name *xxx*`::cdev:vxfs:.`

For information on how to configure VxFS and set up file devices for use with new and existing Oracle databases, see the *VERITAS Database Edition for Oracle Database Administrator's Guide*.

VERITAS QuickLog 10

Introduction

VERITAS QuickLog™ is an optionally licensable feature that enhances file system performance. Although QuickLog can improve file system performance, VxFS does not require QuickLog to operate effectively.

VERITAS QuickLog is part of the `VRTSvxfs` package, but is available only with VERITAS Editions products.

Topics in this chapter include:

Command Name Changes in QuickLog

The table below lists the new names of the QuickLog commands and files along with their original names in the VERITAS Accelerator *for NFS* product.

Old Name	New Name	Command Description
vxld_adm	qlogadm	Low level IOCTL utility for the QuickLog driver.
vxld_cfgrec	qlogrec	Recovers the QuickLog configuration file during failover recovery.
vxld_mntlog vxld_mntall	qlogattach	Attaches a previously formatted QuickLog volume to a QuickLog device.
vxld_logck	qlogck	Recovers QuickLog devices during the boot process.
vxld_umntlog vxld_umntall	qlogdetach	Detaches a QuickLog volume from a QuickLog device.
vxld_umntfs	qlogdisable	Disables QuickLog logging on a VxFS file system and remounts it.
vxld_mntfs	qlogenable	Enables QuickLog logging on a VxFS file system and remounts it.
vxld_mklog	qlogmk	Creates and attaches a QuickLog volume to a QuickLog device.
vxld_print	qlogprint	Displays records from the QuickLog configuration.
vxld_rmall vxld_rmlog	qlogrm	Removes a QuickLog volume from the configuration file.
vxld_stat	qlogstat	Prints statistics for running QuickLog devices, QuickLog volumes, and VxFS file systems.
vxld_trace	qlogtrace	Prints QuickLog tracing.
vxld	qlog	VERITAS QuickLog device driver.
/etc/vxld/config	/etc/qlog/config	QuickLog configuration file.
vxld_cleaner		Removed.
	qlogdb	QuickLog debugging tool (new in this release).

VERITAS QuickLog Overview

The VxFS intent log is stored near the beginning of the volume on which the file system resides (The word *volume* here describes either a VERITAS Volume Manager (VxVM) volume or a raw disk partition). VxFS log writes are sequential, meaning that each log record is written to disk where the previous log record finished. The performance of the log writes is limited because the file system is doing other operations (inode updates, reading and writing data) that require reads and writes from other areas of the disk. The disk head is constantly seeking between the log and data areas of VxFS, reducing the benefits associated with sequential writes to disk.

QuickLog improves file system performance by eliminating the time that a disk spends seeking between the log and data areas of VxFS. This is accomplished by exporting the file system intent log to a separate physical volume called a QuickLog device. A QuickLog device should not reside on a physical disk that shares space with other file systems, since the performance improvement that QuickLog provides depends on the disk head always being in position to write the next log record.

QuickLog is transparent to the end user and requires a minimum of intervention or training to operate.

Figure 5 on page 114 shows a logical view of QuickLog and how it interfaces with the operating system.

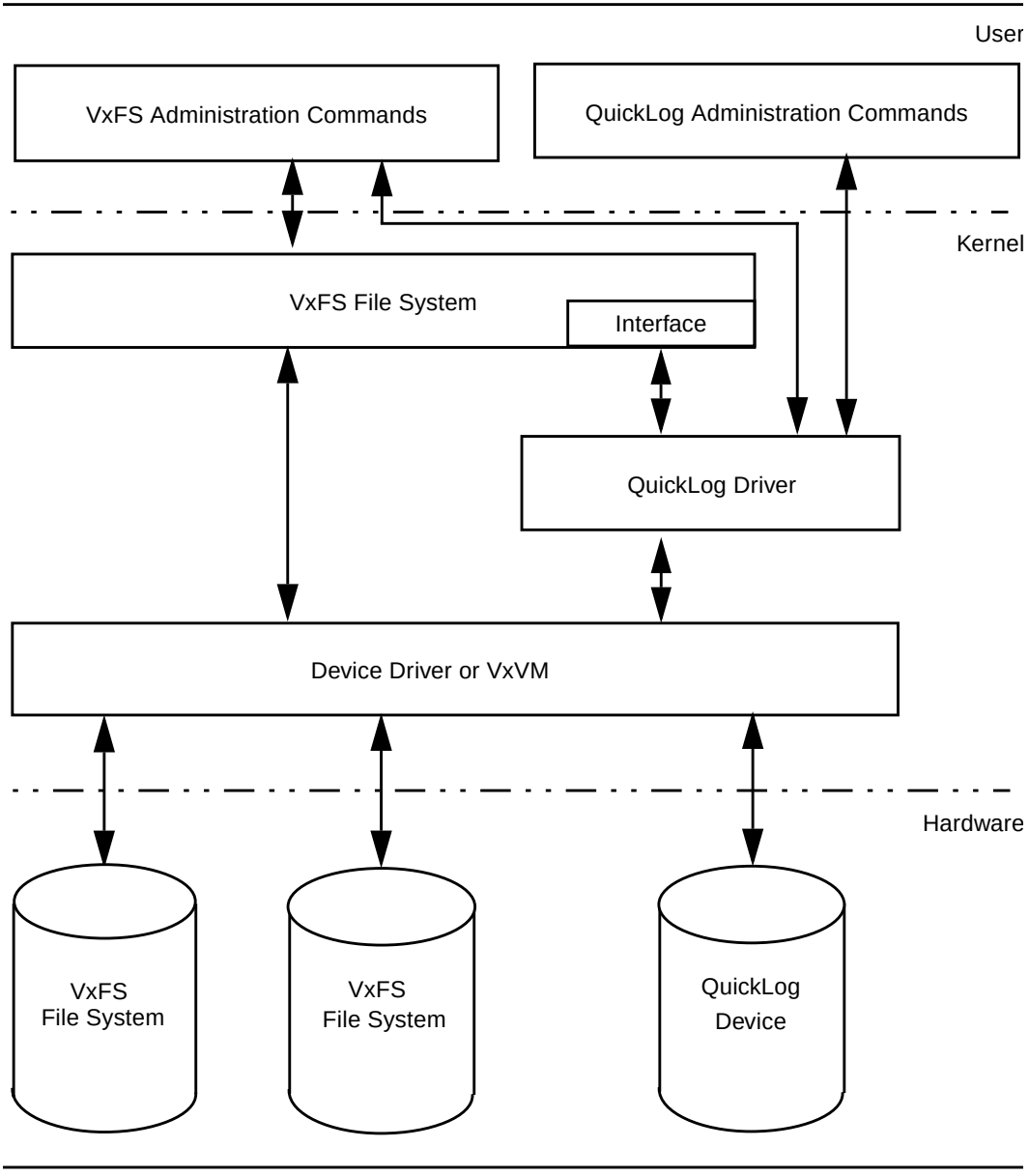
QuickLog Setup

VERITAS QuickLog supports:

- Up to 31 QuickLog devices
- Up to 32 VxFS file systems per QuickLog device
- From one to four QuickLog volumes per QuickLog device (see “QuickLog Load Balancing” on page 117 for details)
- Communication between QuickLog and VxFS through an integrated interface

QuickLog cannot be enabled for the root file system.

Figure 5. QuickLog Logical View



Creating a QuickLog Device

The creation of a QuickLog device requires the following two steps:

1. Create a VxVM volume using the command `vxassist`:

   # **`vxassist -g` *`diskgroup`* `make` *`qlog_volume size vxvm_disk`***

or a raw disk partition using the `format` command.

If the QuickLog volume is a VxVM volume, it must reside in the same disk group as the file system to be logged. Each QuickLog volume should reside on a separate physical disk. Specify *vxvm_disk* during the creation of the VxVM volume to be used by the QuickLog device. The VxVM disk should not be shared or used by any other volumes.

To determine the appropriate size of your QuickLog device, figure out how many file systems you plan to log for this device (1-31). Multiply this number by 16 (16MB is the optimal VxFS log size) to get the total size of the log area for your QuickLog device.The QuickLog device should be approximately 150% of the log area to allow space for QuickLog maps and superblocks. QuickLog devices should be a minimum of 32 MB.

To determine the minimum size of your QuickLog device, figure out how many file systems you plan to log using this device (1-31). Multiply this number by 16 (16 MB is the optimal VxFS log size) to get the total size of the log area for your QuickLog device. This device should be approximately 50% larger than this QuickLog log area and a minimum of 32 MB. For example, to estimate the size needed for 4 file systems on a single QuickLog device:

(4 x 16 MB) x 1.5 = 96 MB

Note For optimal performance, it is advisable to use 2 gigabyte logs if you are logging more than one file system.

2. Build a QuickLog volume using the command `qlogmk`:

   # **`qlogmk -g` *`diskgroup`* `vxlog`[`x`] *`qlog_volume`***

One to four QuickLog volumes must be attached once you have determined the size of your QuickLog device. These volumes provide the static storage for the QuickLog device, including the VxFS log records, QuickLog superblocks and QuickLog maps.

The size of the QuickLog device can be spread out across the one to four QuickLog volumes to be attached (see “QuickLog Load Balancing” on page 117 for details).

The command `qlogmk` both writes out the QuickLog volume layout to the volume *qlog_volume* and attaches the QuickLog volume to the specified QuickLog device. Acceptable QuickLog device names are `vxlog1` through `vxlog31`.

Removing a QuickLog Device

The removal of a QuickLog device involves the `qlogrm` and `vxedit` commands:

```
# qlogrm -g diskgroup qlog_volume
```

`qlogrm` detaches a QuickLog volume from its QuickLog device. If the QuickLog volume is the only volume attached to the QuickLog device, all file systems that are logging to the QuickLog device must have logging by QuickLog disabled prior to using `qlogrm` (see "Disabling a QuickLog Device" on page 117 for details).

Use `vxedit` to remove the VxVM volume:

```
# vxedit -g diskgroup -rf rm qlog_volume
```

VxFS Administration Using QuickLog

Enabling a QuickLog Device

There are two methods to enable logging of a VxFS file system by QuickLog: the QuickLog utility `qlogenable` and a VxFS special mount option.

The `-o qlog=` option to the mount command is provided by VxFS to enable logging by QuickLog. This can be used in conjunction with the `-o remount` mount option to enable QuickLog or change QuickLog devices for active file systems.

From the command line, remount the VxFS file system using `qlogenable`:

```
# qlogenable [qlog_device] /mountpoint
```

or by using the VxFS `-o remount` option:

```
# mount -F vxfs -o remount,qlog=[qlog_device] special /mountpoint
```

The use of either method is transparent to users and does not stop or unmount mounted file systems. When no QuickLog device name is specified, QuickLog automatically assigns one of the idle or least loaded QuickLog devices in the same disk group as that of the file system.

To ensure that QuickLog is enabled for a specific VxFS file system after every system reboot, add "`qlog=`" to the mount option field in the file `/etc/vfstab` for that file system entry, as shown in the following example:

```
# device     device    mountFS fsck mount mount
# to mountto fsck point typepass at boot options
#
/dev/vx/dsk/vol1 /dev/vx/rdsk/vol1 /vol1 vxfs 1 no qlog=
```

If no QuickLog device name is selected after the `qlog=` argument, QuickLog automatically assigns an idle or least loaded QuickLog device.

Disabling a QuickLog Device

To disable logging by QuickLog without unmounting a VxFS file system, use the `qlogdisable` command:

```
# qlogdisable /mountpoint
```

Make sure to disable QuickLog devices for all mounted and logged VxFS file systems and detach all QuickLog volumes before unloading the QuickLog driver (see `qlogdetach`(1M)).

QuickLog Administration and Troubleshooting

This section discusses QuickLog functionality important to a system administrator responsible for implementing and tuning QuickLog.

QuickLog Load Balancing

QuickLog can perform load balancing when two or more physical volumes are attached to a QuickLog device. QuickLog supports from one to four QuickLog volumes attached to each of the 31 QuickLog devices.

QuickLog monitors the average response time for each volume attached to a QuickLog device. If some volume(s) are responding faster than others, QuickLog diverts more of the log writes to those volumes, decreasing the overall response time for the device.

You can add a QuickLog volume to a particular QuickLog device with no more than three QuickLog volumes attached to grow the device's capacity. Similarly, you can remove a QuickLog volume from a QuickLog device with at least one other QuickLog volume attached to shrink the device. Growing or shrinking a QuickLog device does not interrupt file systems logged by QuickLog.

To shrink a QuickLog device that has more than one attached QuickLog volume, detach a QuickLog volume from the QuickLog device by using `qlogdetach`:

```
# qlogdetach vxlog[1-31] qlog_volume
```

Alternatively, if you want to remove the QuickLog volume that you are detaching from the QuickLog device your are shrinking, use `qlogrm`:

```
# qlogrm qlog_volume
```


Before the QuickLog volume is detached, `qlogdetach` flushes all valid log blocks back to the corresponding VxFS logs. The remaining attached QuickLog volumes take up the load released by the removed volume.

To grow a QuickLog device that has three or fewer attached QuickLog volumes, create and attach a QuickLog volume to the QuickLog device by using `qlogmk`:

```
# qlogmk -g diskgroup vxlog[1-31] qlog_volume
```

If the QuickLog volume that you want to attach already exists, attach the volume by using `qlogattach`:

```
# qlogattach vxlog[1-31] qlog_volume
```

The newly attached QuickLog volume begins receiving VxFS log writes being sent to the QuickLog device, easing the load on the existing QuickLog device volumes.

QuickLog Statistics

QuickLog maintains statistics about the QuickLog devices, QuickLog volumes and the VxFS file systems logged by QuickLog. The statistics include:

- The number of read and write I/O operations per second
- The average number of read and write I/O operations per second
- The number of bytes per second for read and write I/O operations
- The average number of bytes per second for read and write I/O operations
- The average service time for read and write I/O operations

See the `qlogstat`(1M) online manual page for details.

QuickLog Recovery

During the boot sequence, the QuickLog start up script `/etc/rcS.d/`S88qlog-startup searches the QuickLog configuration file `/etc/qlog/config`. For each QuickLog device in this file that is in the ATTACHED state, the script tries to replay the log data and metadata that has not been committed to the VxFS file systems before the crash or reboot occurred. This log replay is similar to that of the VxFS `fsck` command (see `fsck_vxfs`(1M) for details). If the log replay is successful, VxFS does not need to perform a full file system consistency check when running `fsck`. (See the `qlogck`(1M) man page for more information).

If an error occurs on one of the QuickLog volumes, the QuickLog device to which this volume is attached is disabled and a full file system consistency check is done on all VxFS file systems that were enabled on this device.

If an error occurs on only one of the file systems logged on a QuickLog device, a full file system consistency check is run only on that file system.

The start up script calls `qlogattach`, which reattaches all recovered QuickLog volumes. The QuickLog volumes must be reattached before you can remount VxFS file systems to log with QuickLog.

Note All operations are done automatically during system start up; no manual intervention is required.

VERITAS File System Quick Start Reference A

Introduction

This appendix provides instructions and examples on performing the following VERITAS File System (VxFS) operations:

- Creating a File System
- Mounting a File System
- Unmounting a File System
- Displaying Information on Mounted File Systems
- Identifying File System Types
- Resizing a File System
- Backing Up and Restoring a File System
- Using Quotas

Creating a File System

The `mkfs` command creates a VxFS file system by writing to a special character device file. The special character device is a raw disk device or a VERITAS Volume Manager (VxVM) volume. `mkfs` builds a file system with a root directory and a `lost+found` directory.

Before running `mkfs`, you must initialize the target device:

- If you have added a new kind of disk controller, which requires a new driver, you must run `drvconfig.`
- If you have added a new disk, you must run `disks`, then run `format` to create a disk slice.

  The `drvconfig`(1M), `disks`(1M), and `format`(1M) manual pages explain how to perform these tasks. If you are using a logical device (such as a VxVM volume), see the VxVM documentation for instructions on device initialization.

How to Create a File System

To create a file system, use the `mkfs` command:

```
mkfs [-F vxfs] [generic_options] [-o specific_options] special [size]
```

`vxfs`	The file system type.
generic_options	Options common to most other file system types.
specific_options	Options specific to VxFS.
`-o N`	Displays the geometry of the file system and does not write to the device.
`-o largefiles`	Allows user to create files larger than two gigabytes.
special	The character (raw) device or VERITAS Volume Manager volume.
size	The size of the new file system (in sectors).

See the following manual pages for more information about the `mkfs` command and its available options:

- `mkfs`(1M)
- `mkfs_vxfs`(1M)

Example

To create a VxFS file system 12288 sectors in size on `/dev/rdsk/c0t6d0s2`, enter:

```
# mkfs -F vxfs /dev/rdsk/c0t6d0s2 12288
```

Information similar to the following displays:

```
version 4 layout

12288 sectors, 6144 blocks of size 1024, log size 512 blocks
unlimited inodes, 5597 data blocks, 5492 free data blocks
1 allocation units of 32778 blocks, 32768 data blocks
last allocation unit has 5597 data blocks
first allocation unit starts at block 537
overhead per allocation unit is 10 blocks
initial allocation overhead is 105 blocks
```

At this point, you can mount the newly created file system.

Mounting a File System

You can mount a VxFS file system by using the `mount` command. When you enter the `mount` command, the generic `mount` command parses the arguments and the `-F` *FSType* option executes the `mount` command specific to that file system type. The `mount` command first searches the `/etc/fs/FSType` directory, then the `/usr/lib/fs/FSType` directory. If the `-F` option is not supplied, the command searches the file `/etc/vfstab` for a file system and an *FSType* matching the special file or mount point provided. If no file system type is specified, `mount` uses the default file system.

How to Mount a File System

After you create a VxFS file system, you can use the `mount` command to mount the file system:

```
mount [-F vxfs] [generic_options] [-r] [-o specific_options] \
special mount_point
```

`vxfs`	File system type.
generic_options	Options common to most other file system types.
specific_options	Options specific to VxFS.
`-o ckpt=`*ckpt_name*	Mounts a VERITAS Storage Checkpoint.
`-o cluster`	Mounts a file system in shared mode. Available only with the VxFS cluster file system feature.
special	Block special device.
mount_point	Directory on which to mount the file system.
`-r`	Mounts the file system as read-only.

Mount Options

The `mount` command has numerous options to tailor a file system for various functions and environments. Some *specific_options* are listed below.

- Security feature

  If security is important, use `blkclear` to ensure that deleted files are completely erased before the space is reused.

- Support for large files

  If you specify the `largefiles` option, you can create files larger than two gigabytes on the file system.

- Support for cluster file systems

  If you specify the `cluster` option, the file system is mounted in shared mode. Cluster file systems depend on several other VERITAS products that must be correctly configured before a complete clustering environment is enabled.

- Using Storage Checkpoints

  The `-o ckpt=`*checkpoint_name* option mounts a Storage Checkpoint of a mounted file system that was previously created by the `fsckptadm` command.

- Using databases

  If you are using databases with VxFS and if you have installed a license key for the VERITAS Quick I/O for Databases feature, the `mount` command enables Quick I/O by default (the same as specifying the `qio` option). The `noqio` option disables Quick I/O. If you do not have Quick I/O, `mount` ignores the `qio` option. Alternatively, you can increase database performance using the mount option `convosync=direct`, which utilizes direct I/O. See "Quick I/O for Databases" on page 99 for more information.

- News file systems

  If you are using `cnews`, use `delaylog` (or `tmplog`),`mincache=closesync` because `cnews` does an `fsync()` on each news file before marking it received. The `fsync()` is performed synchronously as required, but other options are delayed.

- VERITAS QuickLog

  If you are using QuickLog, you can improve I/O performance by moving logging to a separate disk device by using `qlog=`[*dev*]. See "VERITAS QuickLog" on page 111 for more information.

- Temporary file systems

  For a temporary file system such as `/tmp`, where performance is more important than data integrity, use `tmplog,mincache=tmpcache`.

See "Choosing Mount Options" on page 33 and the following manual pages for more information about the `mount` command and its available options:

- `fsckptadm`(1M)
- `mount`(1M)
- `mount_vxfs`(1M)
- `vfstab`(4)

Example

To mount the file system `/dev/dsk/c0t6d0s2` on the `/ext` directory with read / write access and delayed logging, enter:

```
# mount -F vxfs -o delaylog /dev/dsk/c0t6d0s2 /ext
```

How to Edit the vfstab File

You can edit the `/etc/vfstab` file to automatically mount a file system at boot time. You must specify:

- the special block device name to `mount`
- the special character device name used by `fsck`
- the mount point
- the `mount` options
- the file system type (`vxfs`)
- which `fsck` pass looks at the file system
- whether to mount the file system at boot time

Each entry must be on a single line. See the `vfstab`(4) manual page for more information about the `/etc/vfstab` file format.

Here is a typical `vfstab` file with the new file system on the last line:

# device # to mount #	device to fsck	mount point	FS type	fsck pass	mount at boot	mount options
# /dev/dsk/c1d0s2	/dev/rdsk/c1d0s2	/usr	ufs	1	yes	—
/proc	—	/proc	proc	—	no	—
fd	—	/dev/fd	fd	—	no	—
swap	—	/tmp	tmpfs	—	yes	—
/dev/dsk/c0t3d0s0	/dev/rdsk/c0t3d0s0	/	ufs	1	no	—
/dev/dsk/c0t3d0s1	—	—	swap	—	no	—
/dev/dsk/c0t6d0s2	/dev/rdsk/c0t6d0s2	/ext	vxfs	1	yes	—

Unmounting a File System

Use the umount command to unmount a currently mounted file system.

How to Unmount a File System

To unmount a file system, use the following syntax:

```
umount special | mount_point
```

Specify the file system to be unmounted as a *mount_point* or *special* (the device on which the file system resides). See the umount(1M) manual page for more information about this command and its available options.

Example

To unmount the file system /dev/dsk/c0t6d0s2, enter:

```
# umount /dev/dsk/c0t6d0s2
```

To unmount all file systems not required by the system, enter:

```
# umount -a
```

This unmounts all file systems except /, /usr, /usr/kvm, /var, /proc, /dev/fd, and /tmp.

Displaying Information on Mounted File Systems

You can use the `mount` command to display a list of currently mounted file systems.

How to Display File System Information

To view the status of mounted file systems, use the syntax:

```
mount -v
```

This shows the file system type and `mount` options for all mounted file systems. The `-v` option specifies verbose mode.

See the following manual pages for more information about the `mount` command and its available options:

- `mount`(1M)
- `mount_vxfs`(1M)

Example

When invoked without options, the `mount` command displays file system information similar to the following:

```
# mount
/ on /dev/root read/write/setuid on Thu May 26 16:58:24 2000
/proc on /proc read/write on Thu May 26 16:58:25 2000
/dev/fd on /dev/fd read/write on Thu May 26 16:58:26 2000
/tmp on /tmp read/write on Thu May 26 16:59:33 2000
/var/tmp on /var/tmp read/write on Thu May 26 16:59:34 2000
```


Identifying File System Types

Use the `fstyp` command to determine the file system type for a specified file system. This is useful when a file system was created elsewhere and you want to know its type.

How to Identify a File System

To determine the status of mounted file systems, use the syntax:

```
fstyp -v special
```

special	The character (or raw) device.
`-v`	Specifies verbose mode.

See the following manual pages for more information about the `fstyp` command and its available options:

- `fstyp`(1M)
- `fstyp_vxfs`(1M)

Example

To find out what kind of file system is on the device `/dev/rdsk/c0t6d0s2`, enter:

```
# fstyp -v /dev/rdsk/c0t6d0s2
```

The output indicates that the file system type is `vxfs`, and displays file system information similar to the following:

```
vxfs
magic a501fcf5  version 2  ctime Tue Oct 23 18:29:39 1999
logstart 17  logend 1040
bsize  1024 size  1048576 dsize  1047255  ninode 0  nau 8
defiextsize 64  ilbsize 0  immedlen 96  ndaddr 10
aufirst 1049  emap 2  imap 0  iextop 0  istart 0
bstart 34  femap 1051  fimap 0  fiextop 0  fistart 0  fbstart 1083
nindir 2048  aulen 131106  auimlen 0  auemlen 32
auilen 0  aupad 0  aublocks 131072  maxtier 17
inopb 4  inopau 0  ndiripau 0  iaddrlen 8   bshift 10
inoshift 2  bmask fffffc00  boffmask 3ff  checksum d7938aa1
oltext1 9  oltext2 1041  oltsize 8  checksum2 52a
free 382614  ifree 0
efree  676 413 426 466 612 462 226 112 85 35 14 3 6 5 4 4 0 0
```

Resizing a File System

You can extend or shrink mounted VxFS file systems using the `fsadm` command. See the following manual pages for more information about resizing file systems:

- `format`(1M)
- `fsadm_vxfs`(1M)

How to Extend a File System Using fsadm

If a VxFS file system is not large enough, you can increase its size. The size of the file system is specified in units of 512-byte blocks (or sectors).

To extend a VxFS file system, use the syntax:

```
/usr/lib/fs/vxfs/fsadm [-b newsize] [-r rawdev] mount_point
```

`vxfs`	The file system type.
newsize	The size (in sectors) to which the file system will increase.
mount_point	The file system's mount point.
`-r` *rawdev*	Specifies the path name of the raw device if there is no entry in `/etc/vfstab` and `fsadm` cannot determine the raw device.

Note The device must have enough space to contain the larger file system. See the `format`(1M) manual page or the *VERITAS Volume Manager Administrator's Guide* for more information.

Example

To extend the VxFS file system mounted on `/ext` to 22528 sectors, enter:

```
# fsadm -b 22528 /ext
```


How to Shrink a File System

You can decrease the size of the file system using `fsadm`, even while the file system is mounted.

Note In cases where data is allocated towards the end of the file system, shrinking may not be possible.

To decrease the size of a VxFS file system, use the syntax:

```
fsadm [-b newsize] [-r rawdev] mount_point
```

`vxfs`	The file system type.
newsize	The size (in sectors) to which the file system will shrink.
mount_point	The file system's mount point.
`-r` *rawdev*	Specifies the path name of the raw device if there is no entry in `/etc/vfstab` and `fsadm` cannot determine the raw device.

Example

To shrink a VxFS file system mounted at `/ext` to 20480 sectors, enter:

```
# fsadm -b 20480 /ext
```

Note After this operation, there is unused space at the end of the device. You can now resize the device. Be careful not to make the device smaller than the new size of the file system.

How to Reorganize a File System

You can reorganize (or compact) a fragmented file system using `fsadm`, even while the file system is mounted. This may help shrink a file system that could not previously be decreased. To reorganize a VxFS file system, use the syntax:

```
fsadm [-e] [-d] [-E] [-D] [-r rawdev] mount_point
```

`-d`	Reorders directory entries to put subdirectory entries first, then all other entries in decreasing order of time of last access. Also compacts directories to remove free space.
`-D`	Reports on directory fragmentation.
`-e`	Minimizes file system fragmentation. Files are reorganized to have the minimum number of extents.
`-E`	Reports on extent fragmentation.
mount_point	The file system's mount point.
`-r` *rawdev*	Specifies the path name of the raw device if there is no entry in `/etc/vfstab` and `fsadm` cannot determine the raw device.

Example

To reorganize the VxFS file system mounted at `/ext`, enter:

```
# fsadm -EeDd /ext
```

Backing Up and Restoring a File System

To back up a VxFS file system, you first create a read-only snapshot file system, then back up the snapshot. This procedure lets you keep the main file system on line. The snapshot is a copy of the *snapped* file system that is frozen at the moment the snapshot is created.

See "Online Backup" on page 23 and the following manual pages for more information about the `mount`, `vxdump`, and `vxrestore` commands and their available options:

- `mount`(1M)
- `mount_vxfs`(1M)
- `vxdump`(1M)
- `vxrestore`(1M)

How to Create and Mount a Snapshot File System

The first step in backing up a VxFS file system is to create and mount a snapshot file system. To create and mount a snapshot of a VxFS file system, use the syntax:

```
mount [-F vxfs] -o snapof=source,[snapsize=size] \
destination snap_mount_point
```

source	The special device name or mount point of the file system to copy.
destination	The name of the special device on which to create the snapshot.
size	The size of the snapshot file system in sectors.
snap_mount_point	Location where to mount the snapshot; *snap_mount_point* must exist before you enter this command.

Example

To create a snapshot file system of the file system at `/dev/dsk/c0t6d0s2` on `/dev/dsk/c0t5d0s2` and mount it at `/snapmount`, enter:

```
# mount -F vxfs -o snapof=/dev/dsk/c0t6d0s2, \
  snapsize=32768 /dev/dsk/c0t5d0s2 /snapmount
```

You can now back up the file system, as described in the following section.

How to Back Up a File System

After creating a snapshot file system as described in the previous section, you can use `vxdump` to back it up. To back up a VxFS snapshot file system, use the syntax:

```
vxdump [-c] [-f backupdev] snap_mount_point
```

`-c`	Specifies using a cartridge tape device.
backupdev	The device on which to back up the file system.
snap_mount_point	The snapshot file system's mount point.

Example

To back up the VxFS snapshot file system mounted at `/snapmount` to the tape drive with device name `/dev/rmt/00m`, enter:

```
# vxdump -cf /dev/rmt/00m /snapmount
```

How to Restore a File System

After backing up the file system, you can restore it using the `vxrestore` command. First, create and mount an empty file system. To restore a VxFS snapshot file system, use the syntax:

```
vxrestore [-v] [-x] [mount_point]
```

`-v`	Specifies verbose mode.
`-x`	Extracts the named files from the tape.
mount_point	The restored file system's mount point.

Example

To restore a VxFS snapshot file system using `/restore` as a mount point, enter:

```
# vxrestore -vx /restore
```


Using Quotas

You can use quotas to allocate per-user quotas on VxFS file systems.

See the following manual pages for more information about the `vxquota`, `vxquotaon`, `vxquotaoff`, and `vxedquota` commands and their available options:

- `vxquota`(1M)
- `vxquotaon`(1M)
- `vxquotaoff`(1M)
- `vxedquota`(1M)
- “Quotas” on page 75

How to Turn On Quotas

You can enable quotas at mount time or after a file system is mounted. The root directory of the file system must contain a file named `quotas` that is owned by `root`.

To turn on quotas for a mounted file system, use the syntax:

```
vxquotaon mount_point
```

To mount a file system and turn on quotas at the same time, use the syntax:

```
mount -F vxfs -o quota special mount_point
```

If the root directory does not contain a `quotas` file, the `mount` command succeeds, but `quotas` are not turned on.

Example

To create a quotas file (if it does not already exist) and turn on quotas for a VxFS file system mounted at `/mnt`, enter:

```
# touch /mnt/quotas
# vxquotaon /mnt
```

To turn on quotas for a file system at mount time, enter:

```
# mount -F vxfs -o quota /dev/dsk/c0t5d0s2 /mnt
```

How to Set Up User Quotas

You can set user quotas with the `vxedquota` command if you have superuser privileges. User quotas can have a *soft limit* and / or *hard limit*. You can modify the limits or assign them specific values. Users are allowed to exceed the soft limit, but only for a specified time. Disk usage can never exceed the hard limit. The default time limit for exceeding the soft limit is seven days on VxFS file systems.

`vxedquota` creates a temporary file for a specified user. This file contains on-disk quotas for each mounted VxFS file system that has a `quotas` file. The temporary file has one or more lines similar to:

```
fs /mnt blocks (soft = 0, hard = 0) inodes (soft=0, hard=0)
fs /mnt1 blocks (soft = 100, hard = 200) inodes (soft=10, hard=20)
```

Quotas do not need to be turned on for `vxedquota` to work. However, the quota limits apply only after quotas are turned on for a given file system.

`vxedquota` has an option to modify time limits. Modified time limits apply to the entire file system; you cannot set time limits for an individual user.

To invoke the quota editor, use the syntax:

```
vxedquota username
```

To modify the time limit, use the syntax:

```
vxedquota -t
```


How to View Quotas

The superuser or individual user can view disk quotas and usage on VxFS file systems using the `vxquota` command. To view quotas for a specific user, use the syntax:

```
vxquota -v username
```

This command displays the user's quotas and disk usage on all mounted VxFS file systems where the `quotas` file exists. You will see all established quotas regardless of whether or not the quotas are actually turned on.

How to Turn Off Quotas

You can turn off quotas for a mounted file system using the `vxquotaoff` command. To turn off quotas for a file system, use the syntax:

```
vxquotaoff mount_point
```

Example

To turn off quotas for a VxFS file system mounted at `/mnt`, enter:

```
# vxquotaoff /mnt
```

Kernel Messages B

Introduction

This appendix contains a listing of diagnostic or error messages generated by the VERITAS File System (VxFS) kernel. Each message has a description and a suggestion on how to handle or correct the underlying problem.

The following topics are covered in this chapter:

File System Response to Problems

When the file system encounters problems, it responds in one of three ways:

- Marks an inode bad
- Disables transactions
- Disables the file system

Marking an Inode Bad

Inodes can be marked bad if an inode update or a directory-block update fails. In these types of failures, the file system does not know what information is on the disk, and considers all the information that it finds to be invalid. After an inode is marked bad, the kernel still permits access to the file name, but any attempt to access the data in the file or change the inode fails.

Disabling Transactions

If the file system detects an error while writing the intent log, it disables transactions. After transactions are disabled, the files in the file system can still be read or written, but no block or inode frees or allocations, structural changes, directory entry changes, or other changes to metadata are allowed.

Disabling a File System

If an error occurs that compromises the integrity of the file system, VxFS disables itself. If the intent log fails or an inode-list error occurs, the super-block is ordinarily updated (setting the `VX_FULLFSCK` flag) so that the next `fsck` does a full structural check. If this super-block update fails, any further changes to the file system can cause inconsistencies that are undetectable by the intent log replay. To avoid this situation, the file system disables itself.

Recovering a Disabled File System

When the file system is disabled, no data can be written to the disk. Although some minor file system operations still work, most simply return `EIO`. The only thing that can be done when the file system is disabled is to do a `umount` and run a full `fsck`.

Although a log replay may produce a clean file system, do a full structural check to be safe. To do a full structural check, enter:

```
# fsck -F vxfs -o full -y /dev/rdsk/c1t0d0s1
```

The file system usually becomes disabled because of disk errors. Disk failures that disable a file system should be fixed as quickly as possible (see `fsck_vxfs`(1M)).

Kernel Messages

This section lists the VxFS kernel error messages in numerical order. The *Description* subsection for each message describes the problem, the *Action* sub-section suggests possible solutions.

Global Message IDs

Each time a VxFS kernel message is displayed on the system console, it is displayed along with a monatomically increasing message ID, shown in the `msgcnt` field. This ID guarantees that the sequence of events is known to help analyze file system problems.

Each message is also written to an internal kernel buffer that you can view in the file `/var/adm/messages`.

In some cases, additional data is written to the kernel buffer. For example, if an inode is marked bad, the contents of the bad inode are written. When an error message is displayed on the console, you can use the unique message ID to find the message in `/var/adm/messages` and obtain the additional information.

Message Number	Message and Definition
001	NOTICE: msgcnt x: vxfs: mesg 001: vx_nospace - *mount_point* file system full (*n* block extent) The file system is out of space. Often, there is plenty of space and one runaway process used up all the remaining free space. In other cases, the available free space becomes fragmented and unusable for some files. ◆ Action Monitor the free space in the file system and prevent it from becoming full. If a runaway process has used up all the space, stop that process, find the files created by the process, and remove them. If the file system is out of space, remove files, defragment, or expand the file system. To remove files, use the `find` command to locate the files that are to be removed. To get the most space with the least amount of work, remove large files or file trees that are no longer needed. To defragment or expand the file system, use `fsadm` (see the `fsadm`(1M) manual page).
002	WARNING: msgcnt x: vxfs: mesg 002: vx_snap_strategy - *mount_point* file system write attempt to read-only file system WARNING: msgcnt x: vxfs: mesg 002: vx_snap_copyblk - *mount_point* file system write attempt to read-only file system ◆ Description The kernel tried to write to a read-only file system. This is an unlikely problem, but if it occurs, the file system is disabled. ◆ Action The file system was not written, so no action is required. Report this as a bug to your customer support organization.

Message Number	Message and Definition
003, 004, 005	WARNING: msgcnt x: vxfs: mesg 003: vx_mapbad - *mount_point* file system free extent bitmap in au aun marked bad WARNING: msgcnt x: vxfs: mesg 004: vx_mapbad - *mount_point* file system free inode bitmap in au aun marked bad WARNING: msgcnt x: vxfs: mesg 005: vx_mapbad - *mount_point* file system inode extended operation bitmap in au aun marked bad ◆ Description If there is an I/O failure while writing a bitmap, the map is marked bad. The kernel considers the maps to be invalid, so does not do any more resource allocation from maps. This situation can cause the file system to report `out of space` or `out of inode` error messages even though `df` may report an adequate amount of free space. This error may also occur due to bitmap inconsistencies. If a bitmap fails a consistency check, or blocks are freed that are already free in the bitmap, the file system has been corrupted. This may have occurred because a user or process wrote directly to the device or used `fsdb` to change the file system. The VX_FULLFSCK flag is set. If the map that failed was a free extent bitmap, and the VX_FULLFSCK flag can't be set, then the file system is disabled. ◆ Action Check the console log for I/O errors. If the problem is a disk failure, replace the disk. If the problem is not related to an I/O failure, find out how the disk became corrupted. If no user or process was writing to the device, report the problem to your customer support organization. Unmount the file system and use `fsck` to run a full structural check.
006, 007	WARNING: msgcnt *x*: vxfs: mesg 006: vx_sumupd - *mount_point* file system summary update in au *aun* failed WARNING: msgcnt *x*: vxfs: mesg 007: vx_sumupd - *mount_point* file system summary update in inode au *iaun* failed ◆ Description An I/O error occurred while writing the allocation unit or inode allocation unit bitmap summary to disk. This sets the VX_FULLFSCK flag on the file system. If the VX_FULLFSCK flag can't be set, the file system is disabled. ◆ Action Check the console log for I/O errors. If the problem was caused by a disk failure, replace the disk before the file system is mounted for write access, and use `fsck` to run a full structural check.

Message Number	Message and Definition
008, 009	WARNING: msgcnt *x*: vxfs: mesg 008: vx_direrr - *mount_point* file system inode *inumber* block *blkno* error *errno* WARNING: msgcnt *x*: vxfs: mesg 009: vx_direrr - *mount_point* file system inode *inumber* immediate directory error *errno* ◆ Description A directory operation failed in an unexpected manner. The mount point, inode, and block number identify the failing directory. If the inode is an immediate directory, the directory entries are stored in the inode, so no block number is reported. If the error is `ENOENT` or `ENOTDIR`, an inconsistency was detected in the directory block. This inconsistency could be a bad free count, a corrupted hash chain, or any similar directory structure error. If the error is `EIO` or `ENXIO`, an I/O failure occurred while reading or writing the disk block. The `VX_FULLFSCK` flag is set in the super-block so that `fsck` will do a full structural check the next time it is run. ◆ Action Check the console log for I/O errors. If the problem was caused by a disk failure, replace the disk before the file system is mounted for write access. Unmount the file system and use `fsck` to run a full structural check.
010	WARNING: msgcnt *x*: vxfs: mesg 010: vx_ialloc - *mount_point* file system inode *inumber* not free ◆ Description When the kernel allocates an inode from the free inode bitmap, it checks the mode and link count of the inode. If either is non-zero, the free inode bitmap or the inode list is corrupted. The `VX_FULLFSCK` flag is set in the super-block so that `fsck` will do a full structural check the next time it is run. ◆ Action Unmount the file system and use `fsck` to run a full structural check.
011	NOTICE: msgcnt *x*: vxfs: mesg 011: vx_noinode - *mount_point* file system out of inodes ◆ Description The file system is out of inodes. ◆ Action Monitor the free inodes in the file system. If the file system is getting full, create more inodes either by removing files or by expanding the file system. File system resizing is described in “Online System Administration” on page 8, and in the `fsadm`(1M) online manual page.

Message Number	Message and Definition
012	WARNING: msgcnt *x*: vxfs: mesg 012: vx_iget - *mount_point* file system invalid inode number *inumber* ◆ Description When the kernel tries to read an inode, it checks the inode number against the valid range. If the inode number is out of range, the data structure that referenced the inode number is incorrect and must be fixed. The VX_FULLFSCK flag is set in the super-block so that `fsck` will do a full structural check the next time it is run. ◆ Action Unmount the file system and use `fsck` to run a full structural check.
013	WARNING: msgcnt *x*: vxfs: mesg 013: vx_iposition - *mount_point* file system inode *inumber* invalid inode list extent ◆ Description For a Version 2 and above disk layout, the inode list is dynamically allocated. When the kernel tries to read an inode, it must look up the location of the inode in the inode list file. If the kernel finds a bad extent, the inode can't be accessed. All of the inode list extents are validated when the file system is mounted, so if the kernel finds a bad extent, the integrity of the inode list is questionable. This is a very serious error. The VX_FULLFSCK flag is set in the super-block and the file system is disabled. ◆ Action Unmount the file system and use `fsck` to run a full structural check.
014	WARNING: msgcnt *x*: vxfs: mesg 014: vx_iget - inode table overflow ◆ Description All the system in-memory inodes are busy and an attempt was made to use a new inode. ◆ Action Look at the processes that are running and determine which processes are using inodes. If it appears there are runaway processes, they might be tying up the inodes. If the system load appears normal, increase the `vxfs_ninode` parameter in the kernel (see "Internal Inode Table Size" on page 40).

Message Number	Message and Definition
015	WARNING: msgcnt *x*: vxfs: mesg 015: vx_ibadinactive - *mount_point* file system can't mark inode *inumber* bad WARNING: msgcnt *x*: vxfs: mesg 015: vx_ilisterr - *mount_point* file system can't mark inode *inumber* bad ◆ Description An attempt to mark an inode bad on disk, and the super-block update to set the VX_FULLFSCK flag, failed. This indicates that a catastrophic disk error may have occurred since both an inode list block and the super-block had I/O failures. The file system is disabled to preserve file system integrity. ◆ Action Unmount the file system and use `fsck` to run a full structural check. Check the console log for I/O errors. If the disk failed, replace it before remounting the file system.
016	`WARNING: msgcnt` *x* `: vxfs: mesg 016: vx_ilisterr -` *mount_point* `file system error reading inode` *inumber* ◆ Description An I/O error occurred while reading the inode list. The VX_FULLFSCK flag is set. ◆ Action Check the console log for I/O errors. If the problem was caused by a disk failure, replace the disk before the file system is mounted for write access. Unmount the file system and use `fsck` to run a full structural check.
017	WARNING: msgcnt *x*: vxfs: mesg 017: vx_attr_getblk - *mount_point* file system inode *inumber* marked bad WARNING: msgcnt *x*: vxfs: mesg 017: vx_attr_iget - *mount_point* file system inode *inumber* marked bad WARNING: msgcnt *x*: vxfs: mesg 017: vx_attr_indadd - *mount_point* file system inode *inumber* marked bad WARNING: msgcnt *x*: vxfs: mesg 017: vx_attr_indtrunc - *mount_point* file system inode *inumber* marked bad WARNING: msgcnt *x*: vxfs: mesg 017: vx_attr_iremove - *mount_point* file system inode *inumber* marked bad WARNING: msgcnt *x*: vxfs: mesg 017: vx_bmap - *mount_point* file system inode *inumber* marked bad WARNING: msgcnt *x*: vxfs: mesg 017: vx_bmap_indirect_ext4 - *mount_point* file system inode *inumber* marked bad WARNING: msgcnt *x*: vxfs: mesg 017: vx_delbuf_flush - *mount_point* file system inode *inumber* marked bad WARNING: msgcnt *x*: vxfs: mesg 017: vx_dio_iovec - *mount_point* file system inode *inumber* marked bad

Message Number	Message and Definition
017 (continued)	WARNING: msgcnt *x*: vxfs: mesg 017: vx_dirbread - *mount_point* file system inode *inumber* marked bad WARNING: msgcnt *x*: vxfs: mesg 017: vx_dircreate - *mount_point* file system inode *inumber* marked bad WARNING: msgcnt *x*: vxfs: mesg 017: vx_dirlook - *mount_point* file system inode *inumber* marked bad WARNING: msgcnt *x*: vxfs: mesg 017: vx_doextop_iau - *mount_point* file system inode *inumber* marked bad WARNING: msgcnt *x*: vxfs: mesg 017: vx_doextop_now - *mount_point* file system inode *inumber* marked bad WARNING: msgcnt *x*: vxfs: mesg 017: vx_do_getpage - *mount_point* file system inode *inumber* marked bad WARNING: msgcnt *x*: vxfs: mesg 017: vx_enter_ext4 - *mount_point* file system inode *inumber* marked bad WARNING: msgcnt *x*: vxfs: mesg 017: vx_exttrunc - *mount_point* file system inode *inumber* marked bad WARNING: msgcnt *x*: vxfs: mesg 017: vx_get_alloc - *mount_point* file system inode *inumber* marked bad WARNING: msgcnt *x*: vxfs: mesg 017: vx_ilisterr - *mount_point* file system inode *inumber* marked bad WARNING: msgcnt *x*: vxfs: mesg 017: vx_indtrunc - *mount_point* file system inode *inumber* marked bad WARNING: msgcnt *x*: vxfs: mesg 017: vx_iread - *mount_point* file system inode *inumber* marked bad WARNING: msgcnt *x*: vxfs: mesg 017: vx_iremove - *mount_point* file system inode *inumber* marked bad WARNING: msgcnt *x*: vxfs: mesg 017: vx_iremove_attr - *mount_point* file system inode *inumber* marked bad WARNING: msgcnt *x*: vxfs: mesg 017: vx_logwrite_flush - *mount_point* file system inode *inumber* marked bad WARNING: msgcnt *x*: vxfs: mesg 017: vx_oltmount_iget - *mount_point* file system inode *inumber* marked bad WARNING: msgcnt *x*: vxfs: mesg 017: vx_overlay_bmap - *mount_point* file system inode *inumber* marked bad WARNING: msgcnt *x*: vxfs: mesg 017: vx_readnomap - *mount_point* file system inode *inumber* marked bad WARNING: msgcnt *x*: vxfs: mesg 017: vx_reorg_trunc - *mount_point* file system inode *inumber* marked bad WARNING: msgcnt *x*: vxfs: mesg 017: vx_stablestore - *mount_point* file system inode *inumber* marked bad

Message Number	Message and Definition
017 (continued)	WARNING: msgcnt *x*: vxfs: mesg 017: vx_tranitimes - *mount_point* file system inode *inumber* marked bad WARNING: msgcnt *x*: vxfs: mesg 017: vx_trunc - *mount_point* file system inode *inumber* marked bad WARNING: msgcnt *x*: vxfs: mesg 017: vx_write_alloc2 - *mount_point* file system inode *inumber* marked bad WARNING: msgcnt *x*: vxfs: mesg 017: vx_write_default - *mount_point* file system inode *inumber* marked bad WARNING: msgcnt *x*: vxfs: mesg 017: vx_zero_alloc - *mount_point* file system inode *inumber* marked bad ◆ Description When inode information is no longer dependable, the kernel marks it bad on disk. The most common reason for marking an inode bad is a disk I/O failure. If there is an I/O failure in the inode list, on a directory block, or an indirect address extent, the integrity of the data in the inode, or the data the kernel tried to write to the inode list, is questionable. In these cases, the disk driver prints an error message and one or more inodes are marked bad. The kernel also marks an inode bad if it finds a bad extent address, invalid inode fields, or corruption in directory data blocks during a validation check. A validation check failure indicates the file system has been corrupted. This usually occurs because a user or process has written directly to the device or used `fsdb` to change the file system. The `VX_FULLFSCK` flag is set in the super-block so `fsck` will do a full structural check the next time it is run. ◆ Action Check the console log for I/O errors. If the problem is a disk failure, replace the disk. If the problem is not related to an I/O failure, find out how the disk became corrupted. If no user or process is writing to the device, report the problem to your customer support organization. In either case, unmount the file system and use `fsck` to run a full structural check.
019	WARNING: msgcnt *x*: vxfs: mesg 019: vx_log_add - *mount_point* file system log overflow ◆ Description Log ID overflow. When the log ID reaches `VX_MAXLOGID` (approximately one billion by default), a flag is set so the file system resets the log ID at the next opportunity. If the log ID has not been reset, when the log ID reaches `VX_DISLOGID` (approximately `VX_MAXLOGID` plus 500 million by default), the file system is disabled. Since a log reset will occur at the next 60 second sync interval, this should never happen. ◆ Action Unmount the file system and use `fsck` to run a full structural check.

Message Number	Message and Definition
020	WARNING: msgcnt *x*: vxfs: mesg 020: vx_logerr - *mount_point* file system log error *errno* ◆ Description Intent log failed. The kernel will try to set the VX_FULLFSCK and VX_LOGBAD flags in the super-block to prevent running a log replay. If the super-block can't be updated, the file system is disabled. ◆ Action Unmount the file system and use `fsck` to run a full structural check. Check the console log for I/O errors. If the disk failed, replace it before remounting the file system.
021	WARNING: msgcnt *x*: vxfs: mesg 021: vx_fs_init - *mount_point* file system validation failure ◆ Description When a VxFS file system is mounted, the structure is read from disk. If the file system is marked clean, the structure is correct and the first block of the intent log is cleared. If there is any I/O problem or the structure is inconsistent, the kernel sets the VX_FULLFSCK flag and the mount fails. If the error isn't related to an I/O failure, this may have occurred because a user or process has written directly to the device or used `fsdb` to change the file system. ◆ Action Check the console log for I/O errors. If the problem is a disk failure, replace the disk. If the problem is not related to an I/O failure, find out how the disk became corrupted. If no user or process is writing to the device, report the problem to your customer support organization. In either case, unmount the file system and use `fsck` to run a full structural check.

Message Number	Message and Definition
022	WARNING: msgcnt *x*: vxfs: mesg 022: vx_mountroot - root file system remount failed ◆ Description The remount of the root file system failed. The system will not be usable if the root file system can't be remounted for read / write access. When a VERITAS root file system is first mounted, it is mounted for read-only access. After `fsck` is run, the file system is remounted for read / write access. The remount fails if `fsck` completed a resize operation or modified a file that was opened before the `fsck` was run. It also fails if an I / O error occurred during the remount. Usually, the system halts or reboots automatically. ◆ Action Reboot the system. The system either remounts the root cleanly or runs a full structural `fsck` and remounts cleanly. If the remount succeeds, no further action is necessary. Check the console log for I / O errors. If the disk has failed, replace it before the file system is mounted for write access. If the system won't come up and a full structural `fsck` hasn't been run, reboot the system on a backup root and manually run a full structural `fsck`. If the problem persists after the full structural `fsck` and there are no I / O errors, contact your customer support organization.
023	WARNING: msgcnt *x*: vxfs: mesg 023: vx_unmountroot - root file system is busy and can't be unmounted cleanly ◆ Description There were active files in the file system and they caused the unmount to fail. When the system is halted, the root file system is unmounted. This happens occasionally when a process is hung and it can't be killed before unmounting the root. ◆ Action `fsck` will run when the system is rebooted. It should clean up the file system. No other action is necessary. If the problem occurs every time the system is halted, determine the cause and contact your customer support organization.

Message Number	Message and Definition
024	WARNING: msgcnt *x*: vxfs: mesg 024: vx_cutwait - *mount_point* file system current usage table update error ◆ Description Update to the current usage table (CUT) failed. For a Version 2 disk layout, the CUT contains a fileset version number and total number of blocks used by each fileset. The VX_FULLFSCK flag is set in the super-block. If the super-block can't be written, the file system is disabled. ◆ Action Unmount the file system and use `fsck` to run a full structural check.
025	WARNING: msgcnt *x*: vxfs: mesg 025: vx_wsuper - *mount_point* file system superblock update failed ◆ Description An I/O error occurred while writing the super-block during a resize operation. The file system is disabled. ◆ Action Unmount the file system and use `fsck` to run a full structural check. Check the console log for I/O errors. If the problem is a disk failure, replace the disk before the file system is mounted for write access.
026	WARNING: msgcnt *x*: vxfs: mesg 026: vx_snap_copyblk - *mount_point* primary file system read error ◆ Description Snapshot file system error. When the primary file system is written, copies of the original data must be written to the snapshot file system. If a read error occurs on a primary file system during the copy, any snapshot file system that doesn't already have a copy of the data is out of date and must be disabled. ◆ Action An error message for the primary file system prints. Resolve the error on the primary file system and rerun any backups or other applications that were using the snapshot that failed when the error occurred.

Message Number	Message and Definition
027	WARNING: msgcnt *x*: vxfs: mesg 027: vx_snap_bpcopy - *mount_point* snapshot file system write error ◆ Description A write to the snapshot file system failed. As the primary file system is updated, copies of the original data are read from the primary file system and written to the snapshot file system. If one of these writes fails, the snapshot file system is disabled. ◆ Action Check the console log for I/O errors. If the disk has failed, replace it. Resolve the error on the disk and rerun any backups or other applications that were using the snapshot that failed when the error occurred.
028	WARNING: msgcnt *x*: vxfs: mesg 028: vx_snap_alloc - *mount_point* snapshot file system out of space ◆ Description The snapshot file system ran out of space to store changes. During a snapshot backup, as the primary file system is modified, the original data is copied to the snapshot file system. This error can occur if the snapshot file system is left mounted by mistake, if the snapshot file system was given too little disk space, or the primary file system had an unexpected burst of activity. The snapshot file system is disabled. ◆ Action Make sure the snapshot file system was given the correct amount of space. If it was, determine the activity level on the primary file system. If the primary file system was unusually busy, rerun the backup. If the primary file system is no busier than normal, move the backup to a time when the primary file system is relatively idle or increase the amount of disk space allocated to the snapshot file system. Rerun any backups that failed when the error occurred.
029, 030	WARNING: msgcnt *x*: vxfs: mesg 029: vx_snap_getbp - *mount_point* snapshot file system block map write error WARNING: msgcnt *x*: vxfs: mesg 030: vx_snap_getbp - *mount_point* snapshot file system block map read error ◆ Description During a snapshot backup, each snapshot file system maintains a block map on disk. The block map tells the snapshot file system where data from the primary file system is stored in the snapshot file system. If an I/O operation to the block map fails, the snapshot file system is disabled. ◆ Action Check the console log for I/O errors. If the disk has failed, replace it. Resolve the error on the disk and rerun any backups that failed when the error occurred.

Message Number	Message and Definition
031	WARNING: msgcnt *x*: vxfs: mesg 031: vx_disable - *mount_point* file system disabled ◆ Description File system disabled, preceded by a message that specifies the reason. This usually indicates a serious disk problem. ◆ Action Unmount the file system and use `fsck` to run a full structural check. If the problem is a disk failure, replace the disk before the file system is mounted for write access.
032	WARNING: msgcnt *x*: vxfs: mesg 032: vx_disable - *mount_point* snapshot file system disabled ◆ Description Snapshot file system disabled, preceded by a message that specifies the reason. ◆ Action Unmount the snapshot file system, correct the problem specified by the message, and rerun any backups that failed due to the error.
033	WARNING: msgcnt *x*: vxfs: mesg 033: vx_check_badblock - *mount_point* file system had an I/O error, setting VX_FULLFSCK ◆ Description When the disk driver encounters an I/O error, it sets a flag in the super-block structure. If the flag is set, the kernel will set the VX_FULLFSCK flag as a precautionary measure. Since no other error has set the VX_FULLFSCK flag, the failure probably occurred on a data block. ◆ Action Unmount the file system and use `fsck` to run a full structural check. Check the console log for I/O errors. If the problem is a disk failure, replace the disk before the file system is mounted for write access.
034	WARNING: msgcnt *x*: vxfs: mesg 034: vx_resetlog - *mount_point* file system can't reset log ◆ Description The kernel encountered an error while resetting the log ID on the file system. This happens only if the super-block update or log write encountered a device failure. The file system is disabled to preserve its integrity. ◆ Action Unmount the file system and use `fsck` to run a full structural check. Check the console log for I/O errors. If the problem is a disk failure, replace the disk before the file system is mounted for write access.

Message Number	Message and Definition
035	WARNING: msgcnt *x*: vxfs: mesg 035: vx_inactive - *mount_point* file system inactive of locked inode *inumber* ◆ Description `VOP_INACTIVE` was called for an inode while the inode was being used. This should never happen, but if it does, the file system is disabled. ◆ Action Unmount the file system and use `fsck` to run a full structural check. Report as a bug to your customer support organization.
036	WARNING: msgcnt *x*: vxfs: mesg 036: vx_lctbad - *mount_point* file system link count table *lctnumber* bad ◆ Description Update to the link count table (LCT) failed. For a Version 2 and above disk layout, the LCT contains the link count for all the structural inodes. The `VX_FULLFSCK` flag is set in the super-block. If the super-block can't be written, the file system is disabled. ◆ Action Unmount the file system and use `fsck` to run a full structural check.
037	WARNING: msgcnt *x*: vxfs: mesg 037: vx_metaioerr - file system meta data error ◆ Description A read or a write error occurred while accessing file system metadata. The full `fsck` flag on the file system was set. The message specifies whether the disk I/O that failed was a read or a write. File system metadata includes inodes, directory blocks, and the file system log. If the error was a write error, it is likely that some data was lost. This message should be accompanied by another file system message describing the particular file system metadata affected, as well as a message from the disk driver containing information about the disk I/O error. ◆ Action Resolve the condition causing the disk error. If the error was the result of a temporary condition (such as accidentally turning off a disk or a loose cable), correct the condition. Check for loose cables, etc. Unmount the file system and use `fsck` to run a full structural check (possibly with loss of data). In case of an actual disk error, if it was a read error and the disk driver remaps bad sectors on write, it may be fixed when `fsck` is run since `fsck` is likely to rewrite the sector with the read error. In other cases, you replace or reformat the disk drive and restore the file system from backups. Consult the documentation specific to your system for information on how to recover from disk errors. The disk driver should have printed a message that may provide more information.

Message Number	Message and Definition
038	WARNING: msgcnt *x*: vxfs: mesg 038: vx_dataioerr - file system file data error ◆ Description A read or a write error occurred while accessing file data. The message specifies whether the disk I/O that failed was a read or a write. File data includes data currently in files and free blocks. If the message is printed because of a read or write error to a file, another message that includes the inode number of the file will print. The message may be printed as the result of a read or write error to a free block, since some operations allocate an extent and immediately perform I/O to it. If the I/O fails, the extent is freed and the operation fails. The message is accompanied by a message from the disk driver regarding the disk I/O error. ◆ Action Resolve the condition causing the disk error. If the error was the result of a temporary condition (such as accidentally turning off a disk or a loose cable), correct the condition. Check for loose cables, etc. If any file data was lost, restore the files from backups. Determine the file names from the inode number (see the `ncheck(1M)` manual page for more information.) If an actual disk error occurred, make a backup of the file system, replace or reformat the disk drive, and restore the file system from the backup. Consult the documentation specific to your system for information on how to recover from disk errors. The disk driver should have printed a message that may provide more information.
039	WARNING: msgcnt *x*: vxfs: mesg 039: vx_writesuper - file system super-block write error ◆ Description An attempt to write the file system super block failed due to a disk I/O error. If the file system was being mounted at the time, the mount will fail. If the file system was mounted at the time and the full `fsck` flag was being set, the file system will probably be disabled and Message 031 will also be printed. If the super-block was being written as a result of a `sync` operation, no other action is taken. ◆ Action Resolve the condition causing the disk error. If the error was the result of a temporary condition (such as accidentally turning off a disk or a loose cable), correct the condition. Check for loose cables, etc. Unmount the file system and use `fsck` to run a full structural check. If an actual disk error occurred, make a backup of the file system, replace or reformat the disk drive, and restore the file system from backups. Consult the documentation specific to your system for information on how to recover from disk errors. The disk driver should have printed a message that may provide more information.

Message Number	Message and Definition
040	WARNING: msgcnt *x*: vxfs: mesg 040: vx_dqbad - *mount_point* file system quota file update error for id *id* ◆ Description An update to the user quotas file failed for the user ID. The quotas file keeps track of the total number of blocks and inodes used by each user, and also contains soft and hard limits for each user ID. The VX_FULLFSCK flag is set in the super-block. If the super-block cannot be written, the file system is disabled. ◆ Action Unmount the file system and use `fsck` to run a full structural check. Check the console log for I/O errors. If the disk has a hardware failure, it should be repaired before the file system is mounted for write access.
041	WARNING: msgcnt *x*: vxfs: mesg 041: vx_dqget - *mount_point* file system user quota file can't read quota for id *id* ◆ Description A read of the user quotas file failed for the `uid`. The quotas file keeps track of the total number of blocks and inodes used by each user, and contains soft and hard limits for each user ID. The VX_FULLFSCK flag is set in the super-block. If the super-block cannot be written, the file system is disabled. ◆ Action Unmount the file system and use `fsck` to run a full structural check. Check the console log for I/O errors. If the disk has a hardware failure, it should be repaired before the file system is mounted for write access.
042	WARNING: msgcnt *x*: vxfs: mesg 042: vx_bsdquotaupdate - *mount_point* file system user id disk limit reached ◆ Description The hard limit on blocks was reached. Further attempts to allocate blocks for files owned by the user will fail. ◆ Action Remove some files to free up space.
043	WARNING: msgcnt *x*: vxfs: mesg 043: vx_bsdquotaupdate - *mount_point* file system user id disk quota exceeded too long ◆ Description The soft limit on blocks was exceeded continuously for longer than the soft quota time limit. Further attempts to allocate blocks for files will fail. ◆ Action Remove some files to free up space.

Message Number	Message and Definition
044	WARNING: msgcnt *x*: vxfs: mesg 044: vx_bsdquotaupdate - *mount_point* file system user id disk quota exceeded ◆ Description The soft limit on blocks is exceeded. Users can exceed the soft limit for a limited amount of time before allocations begin to fail. After the soft quota time limit has expired, subsequent attempts to allocate blocks for files fail. Note: ◆ Action Remove some files to free up space.
045	WARNING: msgcnt *x*: vxfs: mesg 045: vx_bsdiquotaupdate - *mount_point* file system user id inode limit reached ◆ Description The hard limit on inodes was exceeded. Further attempts to create files owned by the user will fail. ◆ Action Remove some files to free inodes.
046	`WARNING: msgcnt` *x* `: vxfs: mesg 046: vx_bsdiquotaupdate -` *mount_point* `file system user id inode quota exceeded too long` ◆ Description The soft limit on inodes has been exceeded continuously for longer than the soft quota time limit. Further attempts to create files owned by the user will fail. ◆ Action Remove some files to free inodes.
047	WARNING: msgcnt *x*: vxfs: mesg 047: vx_bsdiquotaupdate - warning: *mount_point* file system user id inode quota exceeded ◆ Description The soft limit on inodes was exceeded. The soft limit can be exceeded for a certain amount of time before attempts to create new files begin to fail. Once the time limit has expired, further attempts to create files owned by the user will fail. ◆ Action Remove some files to free inodes.

Message Number	Message and Definition
048, 049	WARNING: msgcnt *x*: vxfs: mesg 048: vx_dqread - warning: *mount_point* file system external user quota file read failed WARNING: msgcnt *x*: vxfs: mesg 049: vx_dqwrite - warning: *mount_point* file system external user quota file write failed ◆ Description To maintain reliable usage counts, VxFS maintains the user quotas file as a structural file in the structural fileset. These files are updated as part of the transactions that allocate and free blocks and inodes. For compatibility with the quota administration utilities, VxFS also supports the standard user visible quota files. When quotas are turned off, synced, or new limits are added, VxFS tries to update the external quota files. When quotas are enabled, VxFS tries to read the quota limits from the external quotas file. If these reads or writes fail, the external quotas file is out of date. ◆ Action Determine the reason for the failure on the external quotas file and correct it. Recreate the quotas file.
050	WARNING: msgcnt *x*: vxfs: mesg 050: vx_ldlogwrite - *mount_point* file system log write failed ◆ Description A write to VERITAS QuickLog log failed. This marks the log bad and sets the full file system check flag in the super block. ◆ Action No immediate action required. When the file system is unmounted, run a full file system check using `fsck` before mounting it again.
051	WARNING: msgcnt *x*: vxfs: mesg 051: vx_ldlog_start - *mount_point* file system log start failed ◆ Description vx_ldlog_start failed. QuickLog logging is disabled and file system continues to use its own log. ◆ Action No corrective action required on the file system. Determine why the log didn't start and do administrative tasks on QuickLog (for more information on QuickLog, see Chapter 10“).

Message Number	Message and Definition
052	WARNING: msgcnt *x*: vxfs: mesg 052: vx_ldlog_stop - *mount_point* file system log stop failed ◆ Description QuickLog copies the log back to the file system after stopping logging activity. If the stop failed, VxFS treats the failure as the log going bad. ◆ Action No immediate action required. When the file system is unmounted, run a full file system check using `fsck` before mounting it again.
053	WARNING: msgcnt *x*: vxfs: mesg 053: vx_ldlog_suspend - *mount_point* file system log suspend failed ◆ Description When the file system is frozen, QuickLog is suspended; it is activated again on thaw. If this operation fails, the kernel marks the log bad and sets the full file system check flag in the super block. ◆ Action No immediate action required. When the file system is unmounted, run a full file system check using `fsck` before mounting it again.
054	WARNING: msgcnt *x*: vxfs: mesg 054: vx_ldlog_resume - *mount_point* file system log resume failed ◆ Description When the file system is thawed, QuickLog must be resumed. If this operation fails, the kernel marks the log bad and sets the full file system check flag in the super block. ◆ Action No immediate action required. When the file system is unmounted, run a full file system check using `fsck` before mounting it again.

Message Number	Message and Definition
056	WARNING: msgcnt *x*: vxfs: mesg 056: vx_mapbad - *mount_point* file system extent allocation unit state bitmap number *number* marked bad ◆ Description If there is an I/O failure while writing a bitmap, the map is marked bad. The kernel considers the maps to be invalid, so does not do any more resource allocation from maps. This situation can cause the file system to report "out of space" or "out of inode" error messages even though `df` may report an adequate amount of free space. This error may also occur due to bitmap inconsistencies. If a bitmap fails a consistency check, or blocks are freed that are already free in the bitmap, the file system has been corrupted. This may have occurred because a user or process wrote directly to the device or used `fsdb` to change the file system. The VX_FULLFSCK flag is set. If the VX_FULLFSCK flag can't be set, the file system is disabled. ◆ Action Check the console log for I/O errors. If the problem is a disk failure, replace the disk. If the problem is not related to an I/O failure, find out how the disk became corrupted. If no user or process was writing to the device, report the problem to your customer support organization. Unmount the file system and use `fsck` to run a full structural check.
057	WARNING: msgcnt *x*: vxfs: mesg 057: vx_esum_bad - *mount_point* file system extent allocation unit summary number *number* marked bad ◆ Description An I/O error occurred reading or writing an extent allocation unit summary. The VX_FULLFSCK flag is set. If the VX_FULLFSCK flag can't be set, the file system is disabled. ◆ Action Check the console log for I/O errors. If the problem is a disk failure, replace the disk. If the problem is not related to an I/O failure, find out how the disk became corrupted. If no user or process was writing to the device, report the problem to your customer support organization. Unmount the file system and use `fsck` to run a full structural check.

Message Number	Message and Definition
058	WARNING: msgcnt *x*: vxfs: mesg 058: vx_isum_bad - *mount_point* file system inode allocation unit summary number *number* marked bad ◆ Description An I/O error occurred reading or writing an inode allocation unit summary. The VX_FULLFSCK flag is set. If the VX_FULLFSCK flag cannot be set, the file system is disabled. ◆ Action Check the console log for I/O errors. If the problem is a disk failure, replace the disk. If the problem is not related to an I/O failure, find out how the disk became corrupted. If no user or process was writing to the device, report the problem to your customer support organization. Unmount the file system and use `fsck` to run a full structural check.
059	WARNING: msgcnt *x*: vxfs: mesg 059: vx_snap_getbitbp - *mount_point* snapshot file system bitmap write error ◆ Description An I/O error occurred while writing to the snapshot file system bitmap. There is no problem with the snapped file system, but the snapshot file system is disabled. ◆ Action Check the console log for I/O errors. If the problem is a disk failure, replace the disk. If the problem is not related to an I/O failure, find out how the disk became corrupted. If no user or process was writing to the device, report the problem to your customer support organization. Restart the snapshot on an error free disk partition. Rerun any backups that failed when the error occurred.
060	WARNING: msgcnt *x*: vxfs: mesg 060: vx_snap_getbitbp - *mount_point* snapshot file system bitmap read error ◆ Description An I/O error occurred while reading the snapshot file system bitmap. There is no problem with snapped file system, but the snapshot file system is disabled. ◆ Action Check the console log for I/O errors. If the problem is a disk failure, replace the disk. If the problem is not related to an I/O failure, find out how the disk became corrupted. If no user or process was writing to the device, report the problem to your customer support organization. Restart the snapshot on an error free disk partition. Rerun any backups that failed when the error occurred.

Message Number	Message and Definition
061	WARNING: msgcnt *x*: vxfs: mesg 061: vx_resize - *mount_point* file system remount failed ◆ Description During a file system resize, the remount to the new size failed. The VX_FULLFSCK flag is set and the file system is disabled. ◆ Action Unmount the file system and use `fsck` to run a full structural check. After the check, the file system shows the new size.
062	NOTICE: msgcnt *x*: vxfs: mesg 062: vx_attr_creatop - invalid disposition returned by attribute driver ◆ Description A registered extended attribute intervention routine returned an invalid return code to the VxFS driver during extended attribute inheritance. ◆ Action Determine which vendor supplied the registered extended attribute intervention routine and contact their customer support organization.
063	WARNING: msgcnt *x*: vxfs: mesg 063: vx_fset_markbad - *mount_point* file system *mount_point* fileset (index *number*) marked bad ◆ Description An error occurred while reading or writing a fileset structure. VX_FULLFSCK flag is set. If the VX_FULLFSCK flag can't be set, the file system is disabled. ◆ Action Unmount the file system and use `fsck` to run a full structural check.
064	WARNING: msgcnt *x*: vxfs: mesg 064: vx_ivalidate - *mount_point* file system inode *number* version number exceeds fileset's ◆ Description During inode validation, a discrepancy was found between the inode version number and the fileset version number. The inode may be marked bad, or the fileset version number may be changed, depending on the ratio of the mismatched version numbers. VX_FULLFSCK flag is set. If the VX_FULLFSCK flag can't be set, the file system is disabled. ◆ Action Check the console log for I/O errors. If the problem is a disk failure, replace the disk. If the problem is not related to an I/O failure, find out how the disk became corrupted. If no user or process is writing to the device, report the problem to your customer support organization. In either case, unmount the file system and use `fsck` to run a full structural check.

Message Number	Message and Definition
066	NOTICE: msgcnt *x*: vxfs: mesg 066: DMAPI mount event - *buffer* ◆ Description An HSM (Hierarchical Storage Management) agent responded to a DMAPI mount event and returned a message in *buffer*. ◆ Action Consult the HSM product documentation for the appropriate response to the message.
067	WARNING: msgcnt *x*: vxfs: mesg 067: mount of *device_path* requires HSM agent ◆ Description The file system mount failed because the file system was marked as being under the management of an HSM agent, and no HSM agent was found during the mount. ◆ Action Restart the HSM agent and try to mount the file system again.
068	WARNING: msgcnt *x*: vxfs: mesg 068: ncsize parameter is greater than 80% of the vxfs_ninode parameter; increasing the value of vxfs:vxfs_ninode ◆ Description The value auto-tuned for the `vxfs_ninode` parameter is less than 125% of the `ncsize` parameter. ◆ Action To prevent this message from occurring, set `vxfs_ninode` to at least 125% of the value of `ncsize`. The best way to do this is to adjust `ncsize` down, rather than adjusting `vxfs_ninode` up. For more information on performance and tuning, see Chapter 4.
069	WARNING: msgcnt *x*: vxfs: mesg 069: memory usage specified by the vxfs:vxfs_ninode and vxfs:vx_bc_bufhwm parameters exceeds available memory; the system may hang under heavy load ◆ Description The value of the system tunable parameters—`vxfs_ninode` and `vx_bc_bufhwm`—add up to a value that is more than 66% of the kernel virtual address space or more than 50% of the physical system memory. VxFS inodes require approximately one kilobyte each, so both values can be treated as if they are in units of one kilobyte. ◆ Action To avoid a system hang, reduce the value of one or both parameters to less than 50% of physical memory or to 66% of kernel virtual memory. For more information on performance and tuning, see Chapter 4.

Message Number	Message and Definition
070	WARNING: msgcnt *x*: vxfs: mesg 070: checkpoint *checkpoint_name* removed from file system *mount_point* ◆ Description The file system ran out of space while updating a Storage Checkpoint. The Storage Checkpoint was removed to allow the operation to complete. ◆ Action Increase the size of the file system. If the file system size cannot be increased, remove files to create sufficient space for new Storage Checkpoints. Monitor capacity of the file system closely to ensure it does not run out of space. See the `fsadm_vxfs`(1M) manual page more information.
071	NOTICE: msgcnt *x*: vxfs: mesg 071: cleared data I/O error flag in *mount_point* file system ◆ Description The user data I/O error flag was reset when the file system was mounted. This message indicates that a read or write error occurred (see Message Number 038) while the file system was previously mounted. ◆ Action Informational only, no action required.
075	WARNING: msgcnt *x*: vxfs: mesg 075: replay fsck failed for *mount_point* file system ◆ Description The log replay failed during a failover or while migrating the CFS primary-ship to one of the secondary cluster nodes. The file system was disabled. ◆ Action Unmount the file system from the cluster. Use `fsck` to run a full structural check and mount the file system again.
076	NOTICE: msgcnt *x*: vxfs: mesg 076: checkpoint asynchronous operation on *mount_point* file system still in progress ◆ Description An EBUSY message was received while trying to unmount a file system. The unmount failure was caused by a pending asynchronous fileset operation, such as a fileset removal or fileset conversion to a nodata Storage Checkpoint. ◆ Action The operation may take a considerable length of time. You can do a forced unmount (see `umount_vxfs`(1M)), or simply wait for the operation to complete so file system can be unmounted cleanly.

Message Number	Message and Definition
077	WARNING: msgcnt *x*: vxfs: mesg 077: vx_fshdchange - *mount_point* file system *number* fileset fileset header : checksum failed ◆ Description Disk corruption was detected while changing fileset headers. This can occur when writing a new inode allocation unit, preventing the allocation of new inodes in the fileset. ◆ Action Unmount the file system and use `fsck` to run a full structural check.
078	WARNING: msgcnt *x*: vxfs: mesg 078: vx_ilealloc - *mount_point* file system *mount_point* fileset (index *number*) ilist corrupt ◆ Description The inode list for the fileset was corrupted and the corruption was detected while allocating new inodes. The failed system call returns an `ENOSPC` error. Any subsequent inode allocations will fail unless a sufficient number of files are removed. ◆ Action Unmount the file system and use `fsck` to run a full structural check.

Disk Layout C

Introduction

The following topics are covered in this chapter:

- Disk Space Allocation
- The VxFS Version 1 Disk Layout
  - Overview
  - Super-Block
  - Intent Log
  - Allocation Unit
- The VxFS Version 2 Disk Layout
  - Overview
  - Basic Layout
  - Filesets and Structural Files
  - Locating Dynamic Structures
- The VxFS Version 4 Disk Layout

Three disk layouts are available with the VERITAS File System (VxFS):

Version 1	The Version 1 disk layout is the original VxFS disk layout provided with pre-2.0 versions of VxFS.
Version 2	The Version 2 disk layout was designed to support features such as filesets, dynamic inode allocation, and enhanced security. The Version 2 layout is available with and without quotas support.
Version 4	The Version 4 disk layout encompasses all file system structural information in files, rather than at fixed locations on disk, allowing for greater scalability. Version 4 supports files up to two terabytes in size and file systems up to one terabyte in size.

Note The Version 3 disk layout is not supported on Solaris.

After VxFS Release 3.3.2 or later is installed on a system, new file systems are created with the Version 4 layout by default. Although `mkfs` allows the user to specify other disk layouts, it is generally preferable to use the Version 4 layout for new file systems.

The `vxupgrade` command is provided to upgrade an existing VxFS file system to the Version 4 layout while the file system remains online. See the `vxupgrade`(1M) manual page for details on upgrading VxFS file systems.

Disk Space Allocation

Disk space is allocated by the system in 512-byte sectors. An integral number of sectors are grouped together to form a logical block. VxFS supports logical block sizes of 1024, 2048, 4096, and 8192 bytes. The default block size is 1024 bytes. The block size may be specified as an argument to the `mkfs` utility and may vary between VxFS file systems mounted on the same system. VxFS allocates disk space to files in extents. An extent is a set of contiguous blocks.

The VxFS Version 1 Disk Layout

This section describes the VxFS Version 1 disk layout.

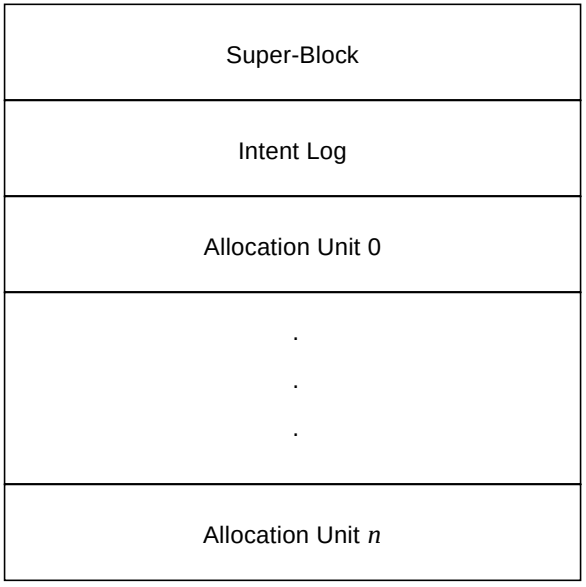
Overview

The VxFS Version 1 disk layout, as shown in Figure 6, includes:

- The super-block
- The intent log
- One or more allocation units

These elements are discussed in detail in the sections that follow.

Figure 6. VxFS Version 1 Disk Layout



Super-Block

The super-block contains important information about the file system, such as:

- The file system type
- Creation and modification dates
- Label information
- Information about the size and layout of the file system
- The count of available resources
- The file system disk layout version number

Refer to the `fs_vxfs`(4) manual page for details on the contents of the super-block.

The super-block is always in a fixed location, offset from the start of the file system by 8192 bytes. This fixed location enables utilities to easily locate the super-block when necessary. The super-block is 1024 bytes long.

Copies of the super-block are kept in allocation unit headers: these copies can be used for recovery purposes if the super-block is corrupted or destroyed (see the `fsck`(1M) manual page for more details).

Intent Log

In the event of system failure, VxFS uses intent logging to guarantee file system integrity.

The *intent log* is a circular activity log with a default size of 1024 blocks. If the file system is smaller than 4 MB, the default log size is reduced (by `mkfs`) to avoid wasting space. The intent log contains records of the intention of the system to update a file system structure. An update to the file system structure (a *transaction*) is divided into separate sub-functions for each data structure that needs to be updated. A composite log record of the transaction is created, containing the sub-functions constituting the transaction.

For example, the creation of a file that would expand the directory in which the file is contained would produce a transaction consisting of the following sub-functions:

- A free extent map update for the allocation of the new directory block
- A directory block update
- An inode modification for the directory size change
- An inode modification for the new file
- A free inode map update for the allocation of the new file

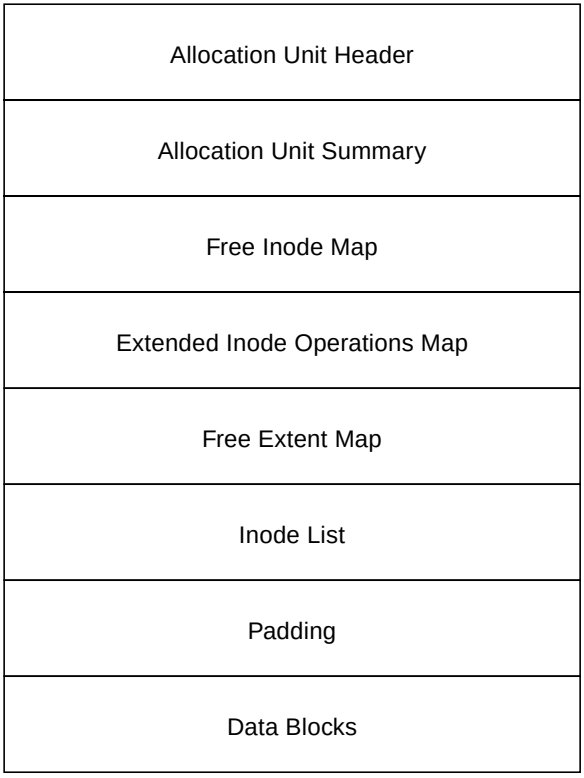
VxFS maintains log records in the intent log for all pending changes to the file system structure and ensures that the log records are written to disk in *advance* of the changes to the file system. Once the intent log has been written, the transaction's other updates to the file system can be written in any order. In the event of a system failure, the pending changes to the file system are either nullified or completed by the `fsck` utility. The VxFS intent log generally only records changes to the file system structure. File data changes are not normally logged.

Note Using QuickLog does not affect the general operation of the intent log.

Allocation Unit

An allocation unit is a group of consecutive blocks in a file system that contain a resource summary, free resource maps, inodes, data blocks, and a copy of the super-block. An allocation unit in VxFS is similar in concept to the UFS “cylinder group.” Each component of an allocation unit begins on a block boundary. The VxFS Version 1 allocation unit is shown in Figure 7.

Figure 7. Allocation Unit Structure



One or more allocation units exist per file system. Allocation units are located immediately after the intent log. The number and size of allocation units can be specified when the file system is made. All of the allocation units, except possibly the last one, are of equal size. If space is limited, the last allocation unit can have a partial set of data blocks to allow use of all remaining blocks.

Allocation Unit Header

The allocation unit header contains a copy of the file system's super-block that is used to verify that the allocation unit matches the super-block of the file system. The super-block copies contained in allocation unit headers can also be used for recovery purposes if the super-block is corrupted or destroyed. The allocation unit header occupies the first block of each allocation unit.

Allocation Unit Summary

The allocation unit summary contains the number of inodes with extended operations pending, the number of free inodes, and the number of free extents in the allocation unit.

Free Inode Map

The free inode map is a bitmap that indicates which inodes are free and which are allocated. A free inode is indicated by the bit being on. Inodes zero and one are reserved by the file system; inode two is the inode for the root directory; inode three is the inode for the `lost+found` directory.

Extended Inode Operations Map

The extended inode operations map keeps track of inodes on which operations would remain pending for too long to reside in the intent log. The extended inode operations map is in the same format as the free inode map. To prevent the intent log from wrapping and the transaction from getting overwritten, the required operations are stored in the affected inode (if the transaction has not completed, it does not get overwritten, the new log waits and the file system is frozen). This map is then updated to identify the inodes that have extended operations that need to be completed.

Free Extent Map

The free extent map is a series of independent 512-byte bitmaps that are each referred to as a free extent map section. Each section is broken down into multiple regions. The first region, of 2048 bits, represents a section of 2048 one-block extents. The second region, of 1024 bits, represents a section of 1024 two-block extents. This regioning continues for all powers of 2 up to the single bit that represents one 2048 block extent.

The file system uses this bitmapping scheme to find an available extent closest in size to the space required. This keeps files as contiguous as possible for faster performance.

Inode List

An inode is a data structure that contains information about a file. The VxFS inode size is 256 bytes. Each inode stores information about a particular file such as:

- File length
- Link count
- Owner and group IDs
- Access privileges
- Time of last access
- Time of last modification
- Pointers to the extents that contain the file's data

There are up to ten direct extent address size pairs per inode. Each direct extent address indicates the starting block number of a direct extent; direct extent sizes can vary. If all of the direct extents are used, two indirect address extents are available for use in each inode:

- The first indirect address extent is used for single indirection, where each entry in the extent indicates the starting block number of an indirect data extent.
- The second indirect address extent is used for double indirection, where each entry in the extent indicates the starting block number of a single indirect address extent.

Each indirect address extent is 8K long and contains 2048 entries. All indirect data extents for a given file have the same size, which is determined when the file's first indirect data extent is allocated.

The inode list is a series of inodes. There is one inode in the list for every file in the file system.

Padding

It may be desirable to align data blocks to a physical boundary. To facilitate this, the system administrator may specify that a gap be left between the end of the inode list and the first data block.

Data Blocks

The balance of the allocation unit is occupied by data blocks. Data blocks contain the actual data stored in files and directories.

The VxFS Version 2 Disk Layout

This section describes the VxFS Version 2 disk layout.

Due to the relatively complex nature of the Version 2 layout, the sections that follow are arranged to cover the following general areas:

- Structural elements of the file system that exist in fixed locations. These elements are discussed in "Basic Layout" on page 176.
- Structural elements of the file system that do not exist in fixed locations. These elements are discussed in "Filesets and Structural Files" on page 180.
- The location and use of the various structural elements when the file system is mounted. This is discussed in "Locating Dynamic Structures" on page 190.

Overview

Many aspects of the Version 1 disk layout are preserved in the Version 2 disk layout. However, the Version 2 layout differs from the Version 1 layout in that it includes support for the following features:

- Filesets (sets of files within a file system)
- Dynamic inode allocation (allocation of inodes on an as-needed basis)
- Enhanced security

The addition of filesets and dynamic allocation of inodes has affected the disk layout in various ways. In particular, many of the file system structures are now located in files (referred to as *structural files*) rather than in fixed disk areas. This provides a simple mechanism for dynamic growth of structures. For example, inodes are now stored in structural files and allocated as needed. In general, file system structures that deal with space allocation are still in fixed disk locations, while most other structures are dynamically allocated and have become clients of the file system's disk space allocation scheme.

The Version 2 disk layout for VxFS 2.3 differs from previous VxFS releases because of the addition of quota support. The differences include the fileset header structure modification to store a quota inode and preallocation of an internal quotas file.

Basic Layout

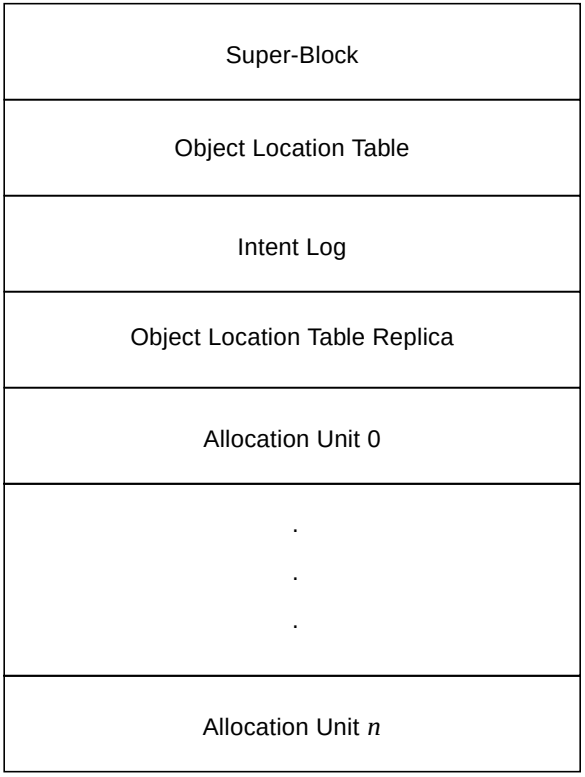
This section describes the structural elements of the file system that exist in fixed locations on the disk.

The VxFS Version 2 disk layout is illustrated in Figure 8 and is composed of:

- The super-block
- The object location table
- The intent log
- A replica of the object location table
- One or more allocation units

These and other elements are discussed in detail in the sections that follow.

Figure 8. VxFS Version 2 Disk Layout



Super-Block

The super-block contains important information about the file system, such as

- The file system type
- Creation and modification dates
- Label information
- Information about the size and layout of the file system
- The count of available resources
- The file system disk layout version number
- Pointers to the object location table and its replica

The super-block is always in a fixed location, offset from the start of the file system by 8192 bytes. This fixed location enables utilities to easily locate the super-block when necessary. The super-block is 1024 bytes long.

Copies of the super-block are kept in allocation unit headers: these copies can be used for recovery purposes if the super-block is corrupted or destroyed (see the `fsck`(1M) manual page).

Object Location Table

The object location table (OLT) can be considered an extension of the super-block. The OLT contains information used at mount time to locate file system structures that are not in fixed locations. The OLT is typically located immediately after the super-block and is 8K long. However, if a Version 1 file system is upgraded to Version 2, the placement of the OLT depends on the availability of space.

The OLT is replicated and its replica is located immediately after the intent log. The OLT and its replica are separated in order to minimize the potential for losing both copies of the vital OLT information in the event of localized disk damage.

The contents and use of the OLT are described in “Locating Dynamic Structures” on page 190.

Intent Log

VxFS uses intent logging to guarantee file system integrity in the event of system failure

The *intent log* is a circular activity log with a default size of 512 blocks. If the file system is less than 4 MB, the log size will be reduced to avoid wasting space. The intent log contains records of the intention of the system to update a file system structure. An update to the file system structure (a *transaction*) is divided into separate sub-functions for each data structure that needs to be updated. A composite log record of the transaction is created that contains the subventions that constitute the transaction.

For example, the creation of a file that would expand the directory in which the file is contained will produce a transaction consisting of the following subventions:

- A free extent map update for the allocation of the new directory block
- A directory block update
- An inode modification for the directory size change
- An inode modification for the new file
- A free inode map update for the allocation of the new file

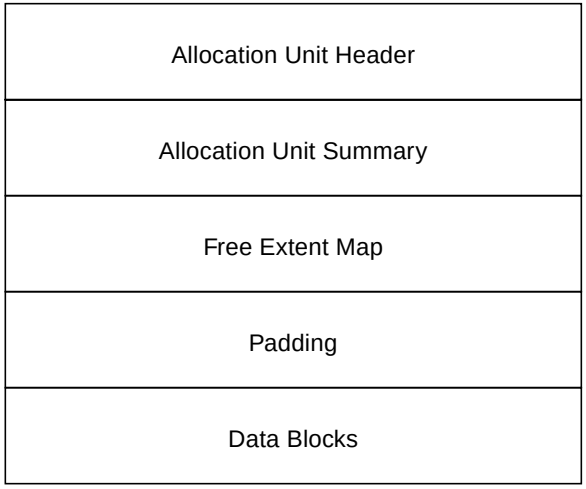
VxFS maintains log records in the intent log for all pending changes to the file system structure, and ensures that the log records are written to disk in *advance* of the changes to the file system. Once the intent log has been written, the transaction's other updates to the file system can be written in any order. In the event of a system failure, the pending changes to the file system are either nullified or completed by the `fsck` utility. The VxFS intent log generally only records changes to the file system structure. File data changes are not normally logged.

Note Using QuickLog does not affect the general operation of the intent log.

Allocation Unit

An allocation unit is a group of consecutive blocks in a file system that contain a resource summary, a free resource map, data blocks, and a copy of the super-block. An allocation unit in VxFS is similar in concept to the UFS "cylinder group." Each component of an allocation unit begins on a block boundary. All of the Version 2 allocation unit components deal with the allocation of disk space. Those components of the Version 1 allocation unit that deal with inode allocation have been relocated elsewhere for Version 2. In particular, the inode list now resides in an inode list file and the inode allocation information now resides in an inode allocation unit (described later). The VxFS Version 2 allocation unit is depicted in Figure 9.

Figure 9. Allocation Unit Structure



One or more allocation units exist per file system. Allocation units are located after the OLT replica. The number and size of allocation units can be specified when the file system is made. All of the allocation units, except possibly the last one, are of equal size. If space is limited, the last allocation unit can have a partial set of data blocks to allow use of all remaining blocks.

Allocation Unit Header

The allocation unit header contains a copy of the file system's super-block that is used to verify that the allocation unit matches the super-block of the file system. The super-block copies contained in allocation unit headers can also be used for recovery purposes if the super-block is corrupted or destroyed. The allocation unit header occupies the first block of each allocation unit.

Allocation Unit Summary

The allocation unit summary summarizes the resources (data blocks) used in the allocation unit. This includes information such as the number of free extents of each size in the allocation unit.

Free Extent Map

The free extent map is a series of independent 512-byte bitmaps that are each referred to as a free extent map section. Each section is broken down into multiple regions. The first region of 2048 bits represents a section of 2048 one-block extents. The second region of 1024 bits represent a section of 1024 two-block extents. This regioning continues for all powers of 2 up to the single bit that represents one 2048 block extent.

The file system uses this bitmapping scheme to find an available extent closest in size to the space required. This keeps files as contiguous as possible for faster performance.

Padding

It may be desirable to align data blocks to a physical boundary. To facilitate this, the system administrator may specify that a gap be left between the end of the free extent map and the first data block. See Chapter 5 for additional information on alignment.

Data Blocks

The balance of the allocation unit is occupied by data blocks. Data blocks contain the actual data stored in files and directories.

Filesets and Structural Files

This section describes the structural elements of the file system that are not necessarily in fixed locations on the disk.

With the Version 2 layout, many structural elements of the file system are encapsulated in files to allow dynamic allocation of the file system structure. Files that store this file system structural data are referred to as *structural files*. As the file system grows, more space is allocated to the structural files. Structural files are intended for file system use only and are not generally visible to users.

The Version 2 layout supports *filesets*, which are collections of files that exist within a file system. In the current release, each file system contains two filesets:

attribute fileset	A special fileset that stores the structural elements of the file system in the form of structural files. These files are a property of the file system and are not normally visible to the user.
primary fileset	A fileset that contains files that are visible to and accessible by users.

Structural files exist in the attribute fileset only and include the following:

fileset header file	A file that contains a series of fileset headers.
inode list file	A file that contains a series of inodes.
inode allocation unit (IAU) file	A file that contains a series of inode allocation units.
current usage table (CUT) file	A file that contains a series of fileset usage entries.
link count table file	A file that contains a link count for each inode in the attribute fileset.
quotas file	A file containing user quota information (for the primary fileset only).

Structural files and their components are discussed in the sections that follow.

Although structural files are contained in the structural fileset, they can “belong” to another fileset. For example, the inode list file for the primary fileset is in the structural fileset, but the structural details that it contains are only applicable to the primary fileset.

Each fileset is defined by structural files as follows:

- An inode list file, which contains the inodes belonging to the fileset
- An inode allocation unit file, which contains a series of inode allocation units
- An entry in the fileset header file, which contains one fileset header per fileset
- An entry in the current usage table file, which contains usage information for each fileset

In addition, the primary fileset has a user quotas file and the structural fileset has a link count table file.

Fileset metadata that cannot be reconstructed using the inode list is replicated to help `fsck` reconstruct the file system in the event of disk damage.

Figure 10 shows a fileset and the structural files by which it is defined.

Fileset Header

Each fileset has a header containing information about the fileset’s contents and characteristic. All fileset headers are stored in a single fileset header file in the attribute fileset. The fileset header file contains one fileset header per fileset (see Figure 11). Each fileset header entry is one block long. The fileset header file is replicated because fileset headers cannot be rebuilt from other data structures.

Figure 10. Filesets and Structural Files

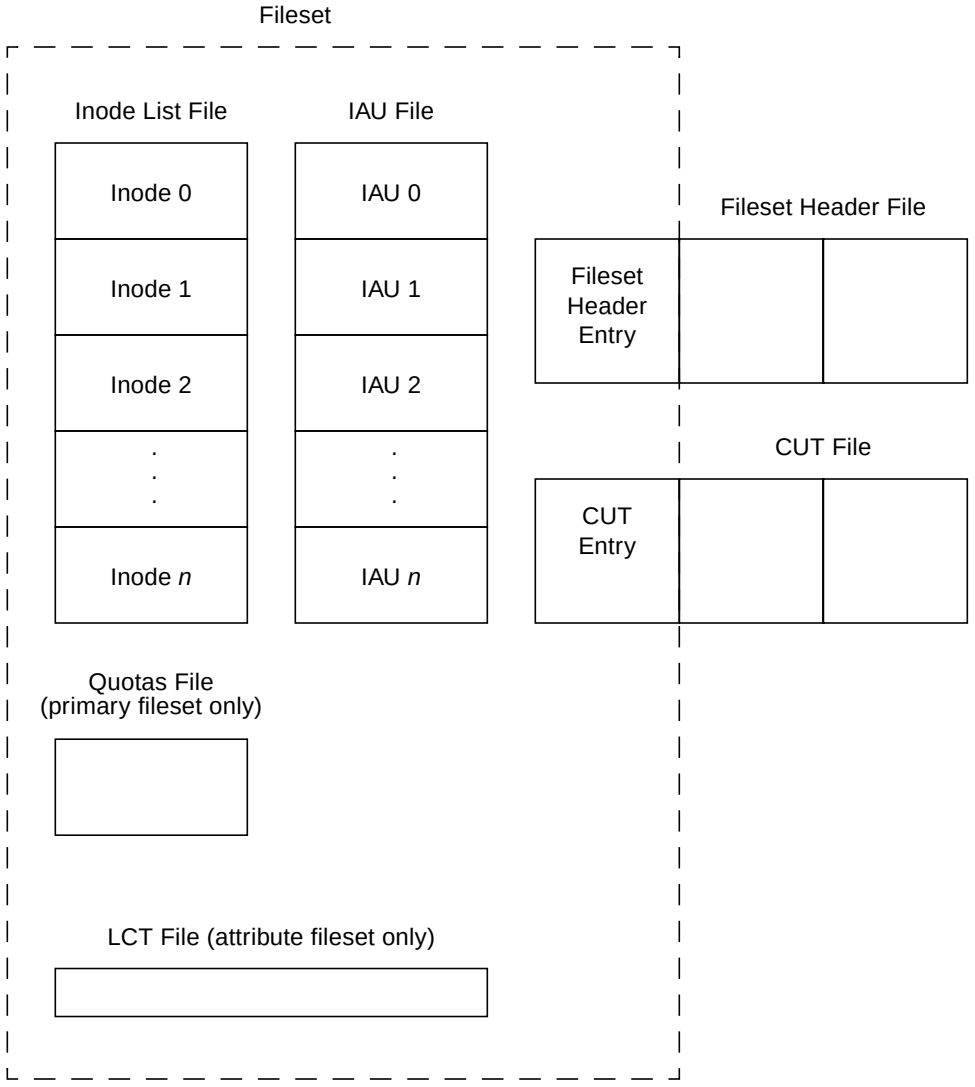
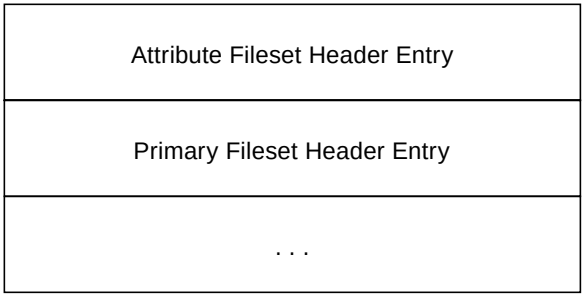


Figure 11. Fileset Header File



The fileset header for a given fileset includes information such as:

- The fileset index (1 for the attribute fileset and 999 for the primary fileset)
- The fileset name
- The inode numbers of the fileset's inode list file and its replica
- The total number of allocated inodes
- The maximum number of inodes allowed in the fileset
- The inode list extent size (in blocks)
- The inode number of the file containing the inode allocation units for the fileset
- The inode number of the fileset's link count table (attribute fileset only)
- The inode number of the fileset's quotas file (primary fileset only)

Note The quotas file inode is present only in VxFS 2.3 and later and is not applicable to earlier releases of VxFS.

Inodes

An inode is a data structure that contains information about a file. The VxFS inode size is 256 bytes. Each inode stores information about a particular file such as:

- File length
- Link count
- Owner and group IDs
- Access privileges
- Time of last access
- Time of last modification
- Pointers to the extents that contain the file's data

Refer to the `inode_vxfs`(4) manual page for details on the contents of a VxFS inode.

There are up to ten direct extent address size pairs per inode. Each direct extent address indicates the starting block number of a direct extent; direct extent sizes can vary. If all of the direct extents are used, two indirect address extents are available for use in each inode. The first indirect address extent is used for single indirection, where each entry in the extent indicates the starting block number of an indirect data extent. The second indirect address extent is used for double indirection, where each entry in the extent indicates the starting block number of a single indirect address extent. Each indirect address extent is 8K long and contains 2048 entries. All indirect data extents for a given file have the same size, which is determined when the file's first indirect data extent is allocated.

Version 2 inodes differ from Version 1 inodes in that they are located in structural files to facilitate *dynamic inode allocation*, which is the allocation of inodes on an as-needed basis. Instead of allocating a fixed number of inodes into the file system, `mkfs` allocates a minimum number of inodes. Additional inodes are later allocated as the file system needs them.

The *inode list* is a series of inodes located in the inode list file. There is one inode in the list for every file in a given fileset. For recovery purposes, the inode list file is referenced by two inodes that point to the *same* set of data blocks. Although the inode addresses are replicated for recovery purposes, the inodes themselves are not.

An *inode extent* is an extent that contains inodes and is 8K long, by default. Inode extents are dynamically allocated to store inodes as they are needed.

Initial Inode List Extents

The initial inode list extents contain the inodes first allocated by `mkfs` for each fileset in a file system. During file system use, inodes are allocated as needed and are added into the inode list files for the filesets.

Figure 12 shows the initial inode list extents allocated for the primary and attribute filesets. Each of these extents contain 32 inodes and is 8K long.

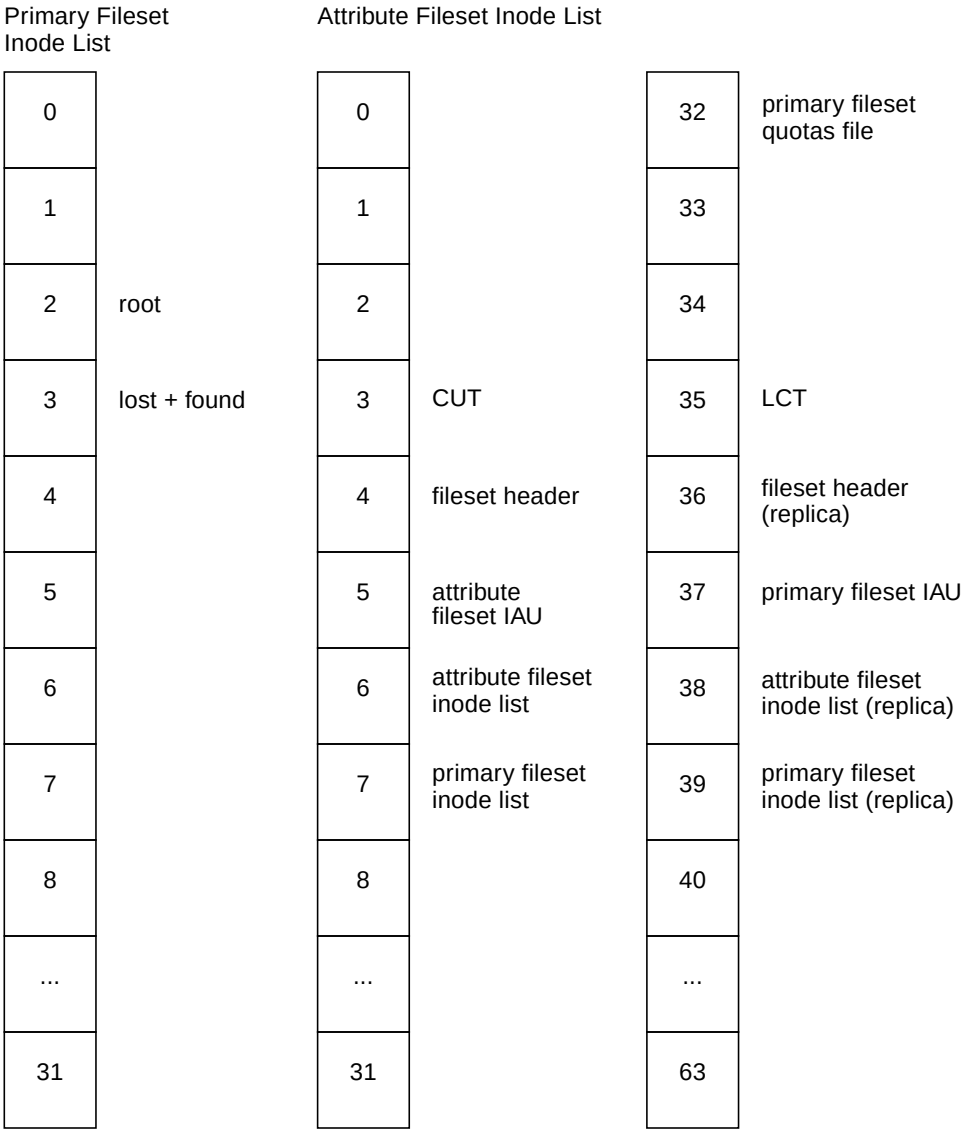
The construction of the primary fileset's inode list resembles that of the VxFS Version 1 file system layout, with the first two inodes reserved and inodes 2 and 3 pre-assigned to the `root` and `lost+found` directories. The structural fileset's inode list is similarly constructed, with certain inodes allocated for specific files and other inodes reserved or unallocated.

There are two initial inode list extents for the attribute fileset. These contain the inodes for all structural files needed to find and set up the file system.

Some of the entries in the structural fileset's inode list are replicas of one another. For example, inodes 4 and 36 both reference copies of the fileset header file. The replicated inodes are used by `fsck` to reconstruct the file system in the event of damage to either one of the replicas. Although the two initial inode list extents belonging to the attribute fileset are logically contiguous, they are physically separated. This helps to ensure the integrity of the replicated information and reduces the chance that localized disk damage might result in complete loss of the file system.

Note that inodes 6 and 38 in the attribute fileset reference the inode list file for the attribute fileset. In a newly created file system, this file contains the two inode extents pictured for the attribute fileset. Likewise, the attribute fileset inodes 7 and 39 reference the inode list file for the primary fileset. In a newly created file system, this file contains the single extent pictured for the primary fileset. All of the unused inodes in the initial extents of the structural inode list are reserved for future use.

Figure 12. Inode Lists



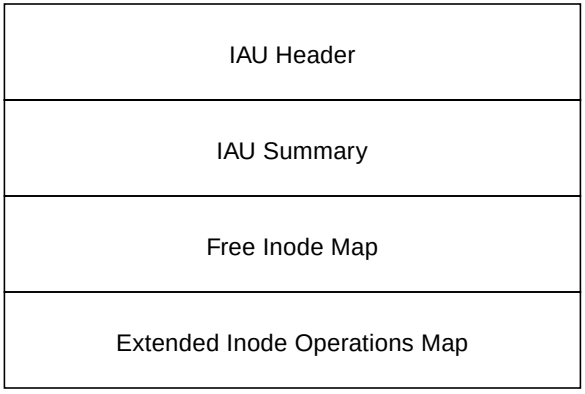
Inode Allocation Unit

An Inode Allocation Unit (IAU) contains inode allocation information for a given fileset. Each fileset contains one or more IAUs, each of which details allocation for a set number of inodes. The number of inodes per IAU varies, depending on the block size being used. One IAU exists for every 16,384 inodes in a fileset with the default block size (1024 bytes). If an IAU is damaged, the information that it contains can be reconstructed by examining the fileset's inode list.

The IAUs for a fileset are stored in sequential order in the fileset's IAU file. The fileset header identifies the attribute fileset inode associated with that fileset's IAU file.

Figure 13 shows the inode allocation unit structure. All IAU components begin on a block boundary.

Figure 13. Inode Allocation Unit (IAU) Structure



IAU Header

The IAU header verifies that the inode allocation unit matches the fileset. The IAU header occupies the first block of each inode allocation unit. If damaged, the IAU can be reconstructed from inodes and other information.

IAU Summary

The IAU summary summarizes the resources used in the IAU. It includes information on the number of free inodes in the IAU and the number of inodes with extended operation sets in the IAU. The IAU summary is 1 block long.

Free Inode Map

The free inode map is a bitmap that indicates which inodes are free and which are allocated. A free inode is indicated by the bit being on. The length of the free inode map is 2K for file systems with 1K or 2K block sizes and is equal to the block size for file systems with larger block sizes.

Extended Inode Operations Map

The extended inode operations map keeps track of inodes on which operations would remain pending for too long to reside in the intent log. The extended inode operations map is in the same format as the free inode map. To prevent the intent log from wrapping and the transaction from getting overwritten, the required operations are stored in the affected inode. This map is then updated to identify the inodes that have extended operations that need to be completed. This map allows the `fsck` utility to quickly identify which inodes had extended operations pending at the time of a system failure. The length of the extended inode operations map is 2K for file systems with 1K or 2K block sizes and is equal to the block size for file systems with larger block sizes.

Link Count Table

The link count table (LCT) contains a reference count for each inode in the associated fileset. This reference count is identical to the conventional link field of an inode. Each LCT entry contains the actual reference count for the associated fileset inode. The link count field in an inode itself is set to either 0 or 1, and the actual number of links is stored in the LCT entry for the associated fileset inode.

The link count table can be reconstructed using the inode list, so it is not replicated.

The current layout only uses the LCT for inodes in the attribute fileset. The LCT supports quick updates of the link count for structural fileset inodes.

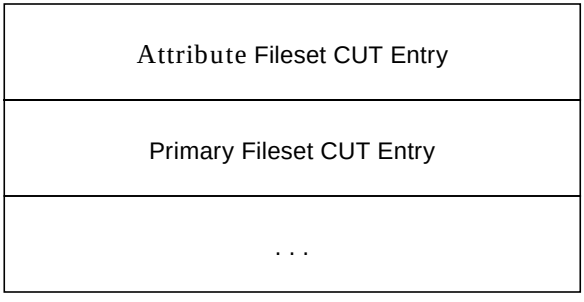
Current Usage Table

The current usage table (CUT) is a file that contains usage related information for each fileset. The information contained in the CUT changes frequently and is not replicated. The information in the CUT can, however, be reconstructed using the inode list if the CUT is damaged.

The CUT file contains one entry per fileset (see Figure 14). The CUT entry for a given fileset contains information such as the following:

- The number of blocks currently used by the fileset.
- The *fileset version number*, which is a 64-bit integer that is guaranteed to be at least as large as the largest inode version number. An *inode version number* is a 64-bit integer that is incremented every time its inode is modified or written to disk and can be used to indicate whether an inode has been modified in any way since the last time it was examined. It is possible to find out which inodes have been modified since a specific time by saving the fileset version number and then later looking for inodes with a larger version number.

Figure 14. Current Usage Table (CUT) File



Quotas File

VxFS supports user quotas on some system resources. The quota limitations are stored in a quotas file. The quotas file keeps track of resources such as soft limits, hard limits, block usage, and inode usage. Quotas are set on a per-file system basis by each user's user ID.

Because quotas apply to mountable filesets only, the attribute fileset does not have quotas. However, the primary fileset's quotas file exists as a structural file in the attribute fileset. The primary fileset's user quotas file is referenced by the structural fileset's initial inode list extent.

Note The quotas file is present only in VxFS 2.3 and later and is not applicable to earlier releases of VxFS.

Locating Dynamic Structures

The existence of dynamic structures in the Version 2 disk layout makes the task of initially locating those structures difficult. The object location table (OLT) contains information needed to initially locate important file system structural elements. In particular, the OLT records the starting block numbers of the initial inode list extents for the attribute fileset and indicates which inodes within those initial extents reference the fileset header file.

Object Location Table Contents

The OLT is composed of records for the following:

fileset header inodes	This record identifies the inode numbers of the fileset header file and its replica.
initial inode list extent addresses	This record identifies the addresses of the beginning of each of two 8K inode extents. These are the initial inode list extents for the attribute fileset, which contain the inodes for all structural files belonging to the attribute fileset.
current usage table inode	This record identifies the inode number of the file that contains the current usage table.

Mounting and the Object Location Table

At mount time, the object location table provides essential information about the location of key file system components. The super-block plays an important role in locating the OLT, in that it contains pointers to both the OLT and its replica.

Using the OLT, the process of mounting a VxFS Version 2 file system is:

1. Read in the super-block. Validate the super-block and its replicas (located in the allocation unit headers).
2. Read and validate the OLT and its replica at the locations recorded in the super-block.
3. Obtain the addresses of the initial inode list extents for the attribute fileset from the OLT. Read in these initial inode extents.
4. Find the fileset header file, based on the fileset header file inode number recorded in the OLT.
5. Read the contents of the fileset header file. Each fileset header file entry represents a particular fileset and indicates the inode numbers of its inode list file and IAU file. The attribute fileset is set up first so that subsequent references to its inode list can be resolved.

The VxFS Version 4 Disk Layout

The Version 4 disk layout allows the file system to scale easily to accommodate large files and large file systems.

The Version 1 and 2 disk layouts divided up the file system space into allocation units. The first AU started part way into the file system which caused potential alignment problems depending on where the first AU started. Each allocation unit also had its own summary, bitmaps, and data blocks. Because this AU structural information was stored at the start of each AU, this also limited the maximum size of an extent that could be allocated. By replacing the allocation unit model of previous versions, the need for alignment of allocation units and the restriction on extent sizes was removed.

The VxFS Version 4 disk layout divides the entire file system space into fixed size allocation units. The first allocation unit starts at block zero and all allocation units are a fixed length of 32K blocks. (An exception may be the last AU, which occupies whatever space remains at the end of the file system). Because the first AU starts at block zero instead of part way through the file system as in previous versions, there is no longer a need for explicit AU alignment or padding to be added when creating a file system (see `mkfs`(1M)).

The Version 4 file system also moves away from the model of storing AU structural data at the start of an AU and puts all structural information in files. So expanding the file system structures simply requires extending the appropriate structural files. This removes the extent size restriction imposed by the Version 1 and Version 2 layouts.

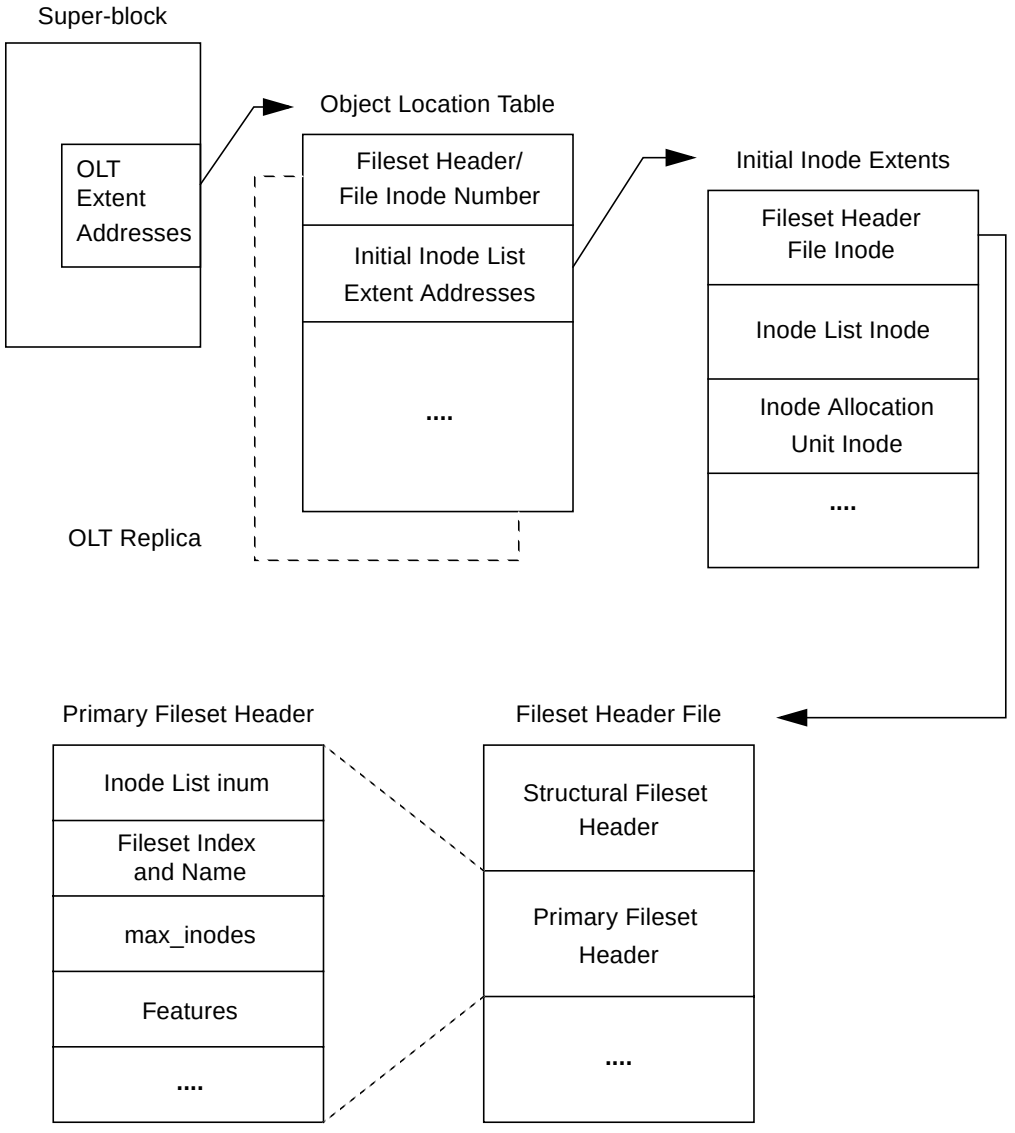
All Version 4 structural files reside in the *structural fileset*, which is similar to the Version 2 attribute fileset. The structural files in the Version 4 disk layout are:

object location table file	Contains the object location table (OLT). As with the Version 2 disk layout, the OLT, which is referenced from the super-block, is used to locate the other structural files.
label file	Encapsulates the super-block and super-block replicas. Although the location of the primary super-block is known, the label file can be used to locate super-block copies if there is structural damage to the file system.
device file	Records device information such as volume length and volume label, and contains pointers to other structural files.
fileset header file	Holds information on a per-fileset basis. This may include the inode of the fileset's inode list file, the maximum number of inodes allowed, an indication of whether the file system supports large files, and the inode number of the quotas file if the fileset supports quotas. When a file system is created, there are two filesets—the *structural fileset* defines the file system structure, the *primary fileset* contains user data.
inode list file	Both the primary fileset and the structural fileset have their own set of inodes stored in an inode list file. Only the inodes in the primary fileset are visible to users. When the number of inodes is increased, the kernel increases the size of the inode list file.
inode allocation unit file	Holds the free inode map, extended operations map, and a summary of inode resources.
log file	Maps the block used by the file system intent log.
extent allocation unit state file	Indicates the allocation state of each AU by defining whether each AU is free, allocated as a whole (no bitmaps allocated), or expanded, in which case the bitmaps associated with each AU determine which extents are allocated.
extent allocation unit summary file	Contains the AU summary for each allocation unit, which contains the number of free extents of each size. The summary for an extent is created only when an allocation unit is expanded for use.
free extent map file	Contains the free extent maps for each of the allocation units.
quotas file	If the file system supports quotas, there is a quotas file which is used to track the resources allocated to each user.

Figure 15 shows how the kernel and utilities build information about the structure of the file system. The super-block location is in a known location from which the OLT can be located. From the OLT, the initial extents of the structural inode list can be located along with the inode number of the fileset header file. The initial inode list extents contain the inode for the fileset header file from which the extents associated with the fileset header file are obtained.

As an example, when mounting the file system, the kernel needs to access the primary fileset in order to access its inode list, inode allocation unit, quotas file and so on. The required information is obtained by accessing the fileset header file from which the kernel can locate the appropriate entry in the file and access the required information.

Figure 15. VxFS Version 4 Disk Layout



Glossary

access control list (ACL)

The information that identifies specific users or groups and their access privileges for a particular file or directory.

agent

A process that manages predefined VERITAS Cluster Server (VCS) resource types. Agents bring resources online, take resources offline, and monitor resources to report any state changes to VCS. When an agent is started, it obtains configuration information from VCS and periodically monitors the resources and updates VCS with the resource status.

allocation unit

A group of consecutive blocks on a file system that contain resource summaries, free resource maps, and data blocks. Allocation units also contain copies of the super-block.

asynchronous writes

A delayed write in which the data is written to a page in the system's page cache, but is not written to disk before the write returns to the caller. This improves performance, but carries the risk of data loss if the system crashes before the data is flushed to disk.

atomic operation

An operation that either succeeds completely or fails and leaves everything as it was before the operation was started. If the operation succeeds, all aspects of the operation take effect at once and the intermediate states of change are invisible. If any aspect of the operation fails, then the operation aborts without leaving partial changes.

Block-Level Incremental Backup (BLI Backup)

A backup capability that does not store and retrieve entire files. Instead, only the data blocks that have changed since the previous backup are backed up.

buffered I/O

During a read or write operation, data usually goes through an intermediate kernel buffer before being copied between the user buffer and disk. If the same data is repeatedly read or written, this kernel buffer acts as a cache, which can improve performance. See *unbuffered I/O* and *direct I/O*.

CFS

VERITAS Cluster File System.

contiguous file

A file in which data blocks are physically adjacent on the underlying media.

current usage table

A table containing fileset information, such as the number of blocks currently used by the fileset. Not used in the Version 4 disk layout.

CVM

The cluster functionality of VERITAS Volume Manager.

data block

A block that contains the actual data belonging to files and directories.

data synchronous writes

A form of synchronous I/O that writes the file data to disk before the write returns, but only marks the inode for later update. If the file size changes, the inode will be written before the write returns. In this mode, the file data is guaranteed to be on the disk before the write returns, but the inode modification times may be lost if the system crashes.

defragmentation

The process of reorganizing data on disk by making file data blocks physically adjacent to reduce access times.

direct extent

An extent that is referenced directly by an inode.

direct I/O

An unbuffered form of I/O that bypasses the kernel's buffering of data. With direct I/O, the file system transfers data directly between the disk and the user-supplied buffer. See *buffered I/O* and *unbuffered I/O*.

discovered direct I/O

Discovered Direct I/O behavior is similar to direct I/O and has the same alignment constraints, except writes that allocate storage or extend the file size do not require writing the inode changes before returning to the application.

encapsulation

A process that converts existing partitions on a specified disk to volumes. If any partitions contain file systems, `/etc/vfstab` entries are modified so that the file systems are mounted on volumes instead. Encapsulation is not applicable on some systems.

extent

A group of contiguous file system data blocks treated as a single unit. An extent is defined by the address of the starting block and a length.

extent attribute

A policy that determines how a file allocates extents.

external quotas file

A quotas file (named `quotas`) must exist in the root directory of a file system for quota-related commands to work. See *quotas file* and *internal quotas file*.

file system block

The fundamental minimum size of allocation in a file system. This is equivalent to the UFS fragment size.

fileset

A collection of files within a file system.

fixed extent size

An extent attribute used to override the default allocation policy of the file system and set all allocations for a file to a specific fixed size.

GB

Gigabyte (2^{30} bytes or 1024 megabytes).

hard limit

The hard limit is an absolute limit on system resources for individual users for file and data block usage on a file system. See *quota*.

indirect address extent

An extent that contains references to other extents, as opposed to file data itself. A *single* indirect address extent references indirect data extents. A *double* indirect address extent references single indirect address extents.

indirect data extent

An extent that contains file data and is referenced via an indirect address extent.

inode

A unique identifier for each file within a file system that contains the data and metadata associated with that file.

inode allocation unit

A group of consecutive blocks containing inode allocation information for a given fileset. This information is in the form of a resource summary and a free inode map.

intent logging

A method of recording pending changes to the file system structure. These changes are recorded in a circular *intent log* file.

internal quotas file

VxFS maintains an internal quotas file for its internal usage. The internal quotas file maintains counts of blocks and inodes used by each user. See *quotas* and *external quotas file*.

K

Kilobyte (2^{10} bytes or 1024 bytes).

large file

A file larger than two gigabytes. VxFS supports files up to two terabytes in size.

large file system

A file system more than two gigabytes in size. VxFS supports file systems up to a terabyte in size.

latency

For file systems, this typically refers to the amount of time it takes a given file system operation to return to the user.

metadata

Structural data describing the attributes of files on a disk.

MB

Megabyte (2^{20} bytes or 1024 kilobytes).

mirror

A duplicate copy of a volume and the data therein (in the form of an ordered collection of subdisks). Each mirror is one copy of the volume with which the mirror is associated.

node

One of the hosts in a cluster.

node abort

A situation where a node leaves a cluster (on an emergency basis) without attempting to stop ongoing operations.

node join

The process through which a node joins a cluster and gains access to shared disks.

object location table (OLT)

The information needed to locate important file system structural elements. The OLT is written to a fixed location on the underlying media (or disk).

object location table replica

A copy of the OLT in case of data corruption. The OLT replica is written to a fixed location on the underlying media (or disk).

page file

A fixed-size block of virtual address space that can be mapped onto any of the physical addresses available on a system.

preallocation

A method of allowing an application to guarantee that a specified amount of space is available for a file, even if the file system is otherwise out of space.

primary fileset

The files that are visible and accessible to the user.

Quick I/O file

A regular VxFS file that is accessed using the `::cdev:vxfs:` extension.

Quick I/O for Databases

Quick I/O is a VERITAS File System feature that improves database performance by minimizing read/write locking and eliminating double buffering of data. This allows online transactions to be processed at speeds equivalent to that of using raw disk devices, while keeping the administrative benefits of file systems.

QuickLog

VERITAS QuickLog is a high performance mechanism for receiving and storing intent log information for VxFS file systems. QuickLog increases performance by exporting intent log information to a separate physical volume.

quotas

Quota limits on system resources for individual users for file and data block usage on a file system. See *hard limit* and *soft limit*.

quotas file

The quotas commands read and write the external quotas file to get or change usage limits. When quotas are turned on, the quota limits are copied from the external quotas file to the internal quotas file. See *quotas*, *internal quotas file*, and *external quotas file*.

reservation

An extent attribute used to preallocate space for a file.

root disk group

A special private disk group that always exists on the system. The root disk group is named `rootdg`.

shared disk group

A disk group in which the disks are shared by multiple hosts (also referred to as a cluster-shareable disk group).

shared volume

A volume that belongs to a shared disk group and is open on more than one node at the same time.

snapshot file system

An exact copy of a mounted file system at a specific point in time. Used to do online backups.

snapped file system

A file system whose exact image has been used to create a snapshot file system.

soft limit

The soft limit is lower than a hard limit. The soft limit can be exceeded for a limited time. There are separate time limits for files and blocks. See *hard limit* and *quota*.

storage checkpoint

A facility that provides a consistent and stable view of a file system or database image and keeps track of modified data blocks since the last checkpoint.

structural fileset

The files that define the structure of the file system. These files are not visible or accessible to the user.

super-block

A block containing critical information about the file system such as the file system type, layout, and size. The VxFS super-block is always located 8192 bytes from the beginning of the file system and is 8192 bytes long.

synchronous writes

A form of synchronous I/O that writes the file data to disk, updates the inode times, and writes the updated inode to disk. When the write returns to the caller, both the data and the inode have been written to disk.

TB

Terabyte (2^{40} bytes or 1024 gigabytes).

transaction

Updates to the file system structure that are grouped together to ensure they are all completed

throughput

For file systems, this typically refers to the number of I/O operations in a given unit of time.

ufs

The UNIX file system type. Used as parameter in some commands.

UFS

The UNIX file system; derived from the 4.2 Berkeley Fast File System.

Unbuffered I/O

I/O that bypasses the kernel cache to increase I/O performance. This is similar to direct I/O, except when a file is extended; for direct I/O, the inode is written to disk synchronously, for unbuffered I/O, the inode update is delayed. See *buffered I/O* and *direct I/O*.

VCS

VERITAS Cluster Server.

volume

A virtual disk which represents an addressable range of disk blocks used by applications such as file systems or databases.

vxfs

The VERITAS File System type. Used as parameter in some commands.

VxFS

The VERITAS File System.

VxVM

The VERITAS Volume Manager.

Index