Veritas File System Administrator's Guide

Solaris

5.0

N18516F

symantec™

Veritas File System Administrator's Guide

Veritas File System 5.0

PN: N18516F

Legal Notice

Symantec Corporation 20330 Stevens Creek Blvd. Cupertino, CA 95014 USA

http://www.symantec.com

Printed in the United States of America.

10 9 8 7 6 5 4 3 2 1

Contents

Chapter 1 Introducing Veritas File System

Chapter 2 VxFS performance: creating, mounting, and tuning File Systems

Chapter 3 Extent attributes

Chapter 4 VxFS I/O Overview

Chapter 1

Introducing Veritas File System

This chapter includes the following topics:

About Veritas File System

A file system is simply a method for storing and organizing computer files and the data they contain to make it easy to find and access them. More formally, a file system is a set of abstract data types (such as metadata) that are implemented for the storage, hierarchical organization, manipulation, navigation, access, and retrieval of data.

Veritas File System (VxFS) was the first commercial journaling file system. With journaling, metadata changes are first written to a log (or journal) then to disk. Since changes do not need to be to be written in multiple places, throughput is much faster as the metadata is written asynchronously.

VxFS is also an extent-based, intent logging file system. VxFS is designed for use in operating environments that require high performance and availability and deal with large amounts of data.

VxFS major components include:

- Logging
- Extents

- File system disk layouts

Logging

A key aspect of any file system is how to recover if a system crash occurs. Earlier methods required a time-consuming scan of the entire file system. A better solution is the method logging (or journaling) the metadata of files.

VxFS logs new attribute information into a reserved area of the file system, whenever file system changes occur. The file system writes the actual data to disk only after the write of the metadata to the log is complete. If and when a system crash occurs, the system recovery code analyzes the metadata log and try to clean up only those files. Without logging, a file system check (`fsck`) must look at all of the metadata.

Intent logging minimizes system downtime after abnormal shutdowns by logging file system transactions. When the system is halted unexpectedly, this log can be replayed and outstanding transactions completed. The check and repair time for file systems can be reduced to a few seconds, regardless of the file system size.

By default, VxFS file systems log file transactions before they are committed to disk, reducing time spent checking and repairing file systems after the system is halted unexpectedly.

Extents

An extent is a contiguous area of storage in a computer file system, reserved for a file. When starting to write to a file, a whole extent is allocated. When writing to the file again, the data continues where the previous write left off. This reduces or eliminates file fragmentation.

Since VxFS is an extent-based file system, addressing is done through extents (which can consist of multiple blocks) rather than in single blocks segments. Extents can therefore enhance file system throughput.

File system disk layouts

The disk layout is the way file system information is stored on disk. On VxFS, several disk layout versions, numbered 1 through 7, were created to support various new features and specific UNIX environments. Currently, only the Version 4, 5, 6, and 7 disk layouts can be created and mounted.

Veritas File System features

VxFS includes the following features:

- Extent-based allocation
  Extents allow disk I/O to take place in units of multiple blocks if storage is allocated in consecutive blocks.
- Extent attributes
  Extent attributes are the extent allocation policies associated with a file.
- Fast file system recovery
  VxFS provides fast recovery of a file system from system failure.
- 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, and large file sizes.
- Enhanced data integrity modes
  VxFS avoids the problem of uninitialized data appearing in a file by waiting until the data has been flushed to disk before updating the new file size to disk.
- Enhanced performance mode
  VxFS provides mount options to improve performance.
- Modes of temporary file systems
  VxFS supplies an option to allow users to achieve higher performance on temporary file sytems by delaying the logging for most operations.
- Improved synchronous writes
  VxFS provides superior performance for synchronous write applications.
- Large files and file systems support
  VxFS supports files larger than two terabytes and large file systems up to 256 terabytes.
- Access control lists (ACLs)
  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.
- Storage Checkpoints
  Backup and restore applications can leverage Storage Checkpoint, a disk- and I/O-efficient copying technology for creating periodic frozen images of a file system.
- Online backup
  VxFS provides online data backup using the snapshot feature.
- Quotas
  VxFS supports quotas, which allocate per-user and per-group quotas and limit the use of two principal resources: files and data blocks.

- Cluster File System
  Clustered file systems are an extension of VxFS that support concurrent direct media access from multiple systems.
- Improved database performance
  Databases can be created on the character devices to achieve the same performance as databases created on raw disks.
- Cross-platform data sharing
  Cross-platform data sharing allows data to be serially shared among heterogeneous systems where each system has direct access to the physical devices that hold the data.
- File Change Log
  The VxFS File Change Log tracks changes to files and directories in a file system.
- Multi-volume support
  The multi-volume support feature allows several volumes to be represented by a single logical object.
- Dynamic Storage Tiering
  The Dynamic Storage Tiering (DST) option allows you to configure policies that automatically relocate files from one volume to another, or relocate files by running file relocation commands, which can improve performance for applications that access specific types of files.

Note: VxFS supports all UFS file system features and facilities except for linking, removing, or renaming "." and ".." directory entries. These operations may disrupt file system operations.

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 4 TB, the block size is 1K. 2K for file systems up to 8TB, 4K for file systems up to 16 TB, 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.

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. A VxFS inode references 10 direct extents, each of which are pairs of starting block addresses and lengths in blocks.

The VxFS inode supports different types of extents, namely ext4 and typed. Inodes with ext4 extents also point to two indirect address extents, which contain the addresses of first and second extents:

first	Used for single indirection. Each entry in the extent indicates the starting block number of an indirect data extent
second	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

VxFS has an 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	Identifies uniquely 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	Is the starting file system block of the extent.
number of blocks	Is the number of contiguous blocks in the extent.

Typed extents have the following characteristics:

- 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 optimized I/O in VxFS.
- 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 and directories 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.

See the setext(1) and getext(1) manual pages.

The `vxtunefs` command allows the administrator to set or view the default indirect data extent size.

See the vxtunefs(1M) manual page.

Fast file system recovery

Most file systems rely on full structural verification by the `fsck` utility as the only means to recover from a system failure. For large disk configurations, this involves a time-consuming process of checking the entire structure, verifying that the file system is intact, and correcting any inconsistencies. VxFS provides fast recovery with the VxFS intent log and VxFS intent log resizing features.

VxFS intent log

VxFS reduces system failure recovery times by tracking file system activity in the VxFS intent log. This feature records pending changes to the file system structure in a circular intent log. The intent log recovery feature is not readily apparent to users or a system administrator except during a system failure. 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. Replaying the intent log may not completely recover the damaged file system structure if there was a disk hardware failure; hardware problems may require a complete system check using the `fsck` utility provided with VxFS.

See "The log option and data integrity" on page 18.

VxFS intent log resizing

The VxFS intent log is allocated when the file system is first created. The size of the intent log is based on the size of the file system—the larger the file system, the larger the intent log. The maximum default intent log size for disk layout Versions 4 and 5 is 16 megabytes. The maximum default intent log size for disk layout Version 6 and 7 is 64 megabytes.

With the Version 6 and 7 disk layouts, you can dynamically increase or decrease the intent log size using the logsize option of the `fsadm` command. Increasing the size of the intent log can improve system performance because it reduces the number of times the log wraps around. However, increasing the intent log size can lead to greater times required for a log replay if there is a system failure.

Note: Inappropriate sizing of the intent log can have a negative impact on system performance.

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

Extended mount options

The VxFS file system supports the following extended mount options:

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

- Large file sizes

See the mount_vxfs(1M) manual page.

Enhanced data integrity modes

For most UNIX file systems, including VxFS, the default mode for writing to a file is delayed, or buffered, meaning that the data to be written is copied to the file system cache and later flushed to disk.

A delayed write provides much better performance than synchronously writing the data to disk. However, in the event of a system failure, data written shortly before the failure may be lost since it was not flushed to disk. In addition, if space was allocated to the file as part of the write request, and the corresponding data was not flushed to disk before the system failure occurred, uninitialized data can appear in the file.

For the most common type of write, delayed extending writes (a delayed write that increases the file size), VxFS avoids the problem of uninitialized data appearing in the file by waiting until the data has been flushed to disk before updating the new file size to disk. If a system failure occurs before the data has been flushed to disk, the file size has not yet been updated to be uninitialized data, thus no uninitialized data appears in the file. The unused blocks that were allocated are reclaimed.

The blkclear option and 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.

The closesync option and 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.

The log option and data integrity

File systems are typically asynchronous in that structural changes to the file system are not immediately written to disk, which provides better performance.

However, recent changes made to a system can be lost if a system failure occurs. Specifically, attribute changes to files and recently created files may disappear.

The mount -o log intent logging option guarantees that all structural changes to the file system are logged to disk before the system call returns to the application. With this option, the `rename`(2) system call flushes the source file to disk to guarantee the persistence of the file data before renaming it. The `rename`() call is also guaranteed to be persistent when the system call returns. The changes to file system data and metadata caused by the `fsync`(2) and `fdatasync`(2) system calls are guaranteed to be persistent once the calls return.

Enhanced performance mode

VxFS has mount options that improve performance, such as delaylog.

The delaylog option and enhanced performance

The default VxFS logging mode, `mount -o delaylog`, increases performance by delaying the logging of some structural changes. However, it does not provide the equivalent data integrity as the previously described modes.however, because 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.

Modes of temporary file systems

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. 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 `mount -o datainlog` option greatly improves the performance of small synchronous writes.

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

Warning: The use of the `-o convosync=dsync` option violates POSIX semantics.

Support for large files

With VxFS, you can create, mount, and manage file systems containing large files (files larger than two terabytes).

Warning: Some applications and utilities may not work on large files.

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. ACLs are supported on cluster file systems.

See the getfacl(1) and setfacl(1) manual pages.

Storage Checkpoints

To increase availability, recoverability, and performance, 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 a 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 copies of the original 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.

Storage Checkpoint functionality is separately licensed.

Online backup

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

When changes are made to the snapped file system, the old data is 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 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`)

See "About snapshot file systems" on page 93.

Quotas

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

The hard limit represents an absolute limit on data blocks or files. A user can never exceed the hard limit under any circumstances.

The soft limit is lower than the hard limit and can be exceeded for a limited amount of time. This allows users to exceed limits temporarily as long as they fall under those limits before the allotted time expires.

See "About quota limits" on page 101.

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 Quick I/O for Databases feature with VxFS lets systems retain the benefits of having a database on a file system without sacrificing performance. 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. 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.

Cluster file systems

Veritas Storage Foundation Cluster File System (SFCFS) allows clustered severs 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 accessile through a cluster.

Beginning with SFCFS 5.0, SFCFS uses a symmetric architecture in which all nodes in the cluster can simultaneously function as metadata severs. SFCFS still as some remnants of the old master/slave or primary/secondary concept. 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. Each SFCFS node has its own intent log. File system operations, such as allocating or deleting files, can originate from any node in the cluster.

Installing VxFS and enabling the cluster feature does not create a cluster file system configuration. File system clustering requires other Veritas products to enable communication services and provide storage resources. These products are packaged with VxFS in the Storage Foundation Cluster File System to provide a complete clustering environment.

See the *Veritas Storage Foundation Cluster File System Administrator's Guide*.

To be a cluster mount, a file system must be mounted using the `mount -o cluster` option. File systems mounted without the `-o cluster` option are termed local mounts.

CFS functionality is separately licensed.

Cross-platform data sharing

Cross-platform data sharing (CDS) allows data to be serially shared among heterogeneous systems where each system has direct access to the physical devices that hold the data. This feature can be used only in conjunction with Veritas Volume Manager (VxVM).

See the *Veritas Storage Foundation Cross-Platform Data Sharing Administrator's Guide*.

File Change Log

The VxFS File Change Log (FCL) tracks changes to files and directories in a file system. The File Change Log can be used by applications such as backup products, webcrawlers, search and indexing engines, and replication software that typically scan an entire file system searching for modifications since a previous scan. FCL functionality is a separately licensed feature.

See "About the File Change Log file" on page 110.

Multi-volume support

The multi-volume support (MVS) feature allows several volumes to be represented by a single logical object. All I/O to and from an underlying logical volume is directed by way of volume sets. This feature can be used only in conjunction with VxVM. MVS functionality is a separately licensed feature.

See "About multi-volume support" on page 118.

Dynamic Storage Tiering

The Dynamic Storage Tiering (DST) option is built on multi-volume support technology. Using DST, you can map more than one volume to a single file system. You can then configure policies that automatically relocate files from one volume to another, or relocate files by running file relocation commands. Having multiple volumes lets you determine where files are located, which can improve performance for applications that access specific types of files. DST functionality is a separately licensed feature and is available with the VRTSfppm package.

See "About Dynamic Storage Tiering" on page 137.

Veritas File System performance enhancements

Traditional file systems employ block-based allocation schemes that provide adequate random access and latency for small files, but which limit throughput for larger files. As a result, they are less than optimal for commercial environments.

VxFS addresses this file system performance issue through an alternative allocation method and increased user control over allocation, I/O, and caching policies.

See "Using Veritas File System" on page 25.

VxFS provides the following performance enhancements:

- Data synchronous I/O

- Direct I/O and discovered direct I/O
- Enhanced I/O performance
- Caching advisories
- Enhanced directory features
- Explicit file alignment, extent size, and preallocation controls
- Tunable I/O parameters
- Tunable indirect data extent size
- Integration with VxVM™
- Support for large directories

Note: VxFS reduces the file lookup time in directories with an extremely large number of files.

About enhanced I/O performance

VxFS provides enhanced I/O performance by applying an aggressive I/O clustering policy, integrating with VxVM, and allowing application specific parameters to be set 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. 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

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

- 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.

See the vxtunefs(1M) and tunefstab(4) manual pages.

Using Veritas File System

There are three main methods to use, manage, modify, and tune VxFS:

- See "Veritas Enterprise Administrator Graphical User Interface" on page 25.
- See "Online system administration" on page 26.
- See "Application program interface" on page 27.

Veritas Enterprise Administrator Graphical User Interface

Enterprise Administrator (VEA) is a GUI-based application using the Java™ technology that consists of a server and a client. The server runs on a UNIX system that is running Volume Manager and VxFS. The client runs on any platform that supports the Java Runtime Environment.

You can perform the following administrative functions on local or remote systems using the GUI:

- Create a file system on a volume
- Create a file system on a volume set
- Remove a file system from the file system table
- Mount and unmounting a file system
- Defragment a file system
- Monitor file system capacity
- Create a snapshot copy of a file system
- Check a file system
- View file system properties
- Unmount a file system from a cluster node

- Remove resource information for a cluster file system
- Maintain the File Change Log
- Maintain Storage Checkpoints
- Use multi-volume file systems
- Set intent log options

See the *Veritas Enterprise Administrator Getting Started* manual.

See the VEA online help.

Online system administration

VxFS provides command line interface (CLI) operations that are described throughout this guide and in manual pages.

VxFS allows you to run a number of administration tasks while the file system is online. Two of the more important tasks include:

- Deframentation
- File system resizing

About defragmentation

Free resources are initially aligned and allocated to files in an order that provides 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 farther along the disk, leaving unused gaps or fragments between areas that are in use. This process is known as fragmentation and leads to degraded performance because the file system has fewer options when assigning a free extent to a file (a group of contiguous data blocks).

VxFS provides the online administration utility `fsadm` to resolve the problem of fragmentation.

The `fsadm` utility defragments a mounted file system by performing the following actions:

- 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.

About file system 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.

VxFS is capable of increasing or decreasing the file system size while in use. Many competing file systems can not do this. 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. Use the `vxresize` command when resizing both the volume and the file system. The `vxresize` command guarantees that the file system shrinks or grows along with the volume. Do not use the `vxassist` and `fsadm_vxfs` commands for this purpose.

See the vxresize(1M) manual page.

See the *Veritas Volume Manager Administrator's Guide*.

Application program interface

Veritas File System Developer's Kit (SDK) provides developers with the information necessary to use the application programming interfaces (APIs) to modify and tune various features and components of File System.

See the *Veritas File System Programmer's Reference Guide*.

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.

Expanded application facilities

VxFS provides API functions frequently associated with commercial applications that make it possible to perform the following actions:

- 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

Because these functions are provided using VxFS-specific ioctl system calls, most existing UNIX system applications do not use them. The VxFS-specific `cp`, `cpio`,

and `mv` utilities use the functions to preserve extent attributes and allocate space more efficiently. The current attributes of a file can be listed using the `getext` or VxFS-specific `ls` command. The functions can also improve performance for custom applications. For portability reasons, these applications must check which file system type they are using before using these functions.

Chapter 2

VxFS performance: creating, mounting, and tuning File Systems

This chapter includes the following topics:

mkfs command options

When you create a file system, you can select a number of characteristics.

- See "Block size" on page 29.
- See "Intent log size" on page 30.

Block size

The unit of allocation in VxFS is a block. Unlike some other UNIX file systems, VxFS does not make use of block fragments for allocation because storage is allocated in extents that consist of one or more blocks.

You specify the block size when creating a file system by using the `mkfs` –o bsize option. The block size cannot be altered after the file system is created. The smallest available block size for VxFS is 1K, which is also the default block size.

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. 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.

For 64-bit kernels, which support 32 terabyte file systems, the block size determines the maximum size of the file system you can create. File systems up to 4 TB require a 1K block size. For four to eight terabyte file systems, the block size is 2K, For file systems between 8 and 16 TB, block size is 4K, and for greater than 16 TB, the block size is 8K. If you specify the file system size when creating a file system, the block size defaults to these values.

Intent log size

You specify the intent log size when creating a file system by using the `mkfs -o logsize` option. With the Version 6 or 7 disk layout, you can dynamically increase or decrease the intent log size using the log option of the `fsadm` command. The `mkfs` utility uses a default intent log size 64 megabytes for disk layout Version 6. 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.

Choosing mount command options

In addition to the standard mount mode (delaylog mode), VxFS provides the following modes of operation:

- log
- delaylog

- tmplog
- logsize
- nodatainlog
- blkclear
- minicache
- convosync
- ioerror
- largefiles|nolorgefiles
- cio

Caching behavior can be altered with the mincache option, and the behavior of O_SYNC and D_SYNC writes can be altered with the convosync option.

See the fcntl(2) manual page.

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.

See the mount_vxfs(1M) manual page.

The log mode

In log mode, all system calls other than write(2), writev(2), and pwrite(2) are guaranteed to be persistent after the system call returns to the application.

The rename(2) system call flushes the source file to disk to guarantee the persistence of the file data before renaming it. In both the log and delaylog modes, the rename is also guaranteed to be persistent when the system call returns. This benefits shell scripts and programs that try to update a file atomically by writing the new file contents to a temporary file and then renaming it on top of the target file.

The delaylog mode

The default logging mode is delaylog. In delaylog mode, the effects of most system calls other than write(2), writev(2), and pwrite(2) are guaranteed to be persistent approximately 15 to 20 seconds after the system call returns to the application. Contrast this with the behavior of most other file systems in which most system calls are not persistent until approximately 30 seconds or more after the call has returned. Fast file system recovery works with this mode.

The rename(2) system call flushes the source file to disk to guarantee the persistence of the file data before renaming it. In the log and delaylog modes, the rename is also guaranteed to be persistent when the system call returns. This benefits shell scripts and programs that try to update a file atomically by writing the new file contents to a temporary file and then renaming it on top of the target file.

The tmplog mode

In tmplog mode, the effects of system calls have persistence guarantees that are similar to those in delaylog mode. In addition, enhanced flushing of delayed extending writes is disabled, which results in better performance but increases the chances of data being lost or uninitialized data appearing in a file that was being actively written at the time of a system failure. This mode is only recommended for temporary file systems. Fast file system recovery works with this mode.

Note: The term “effects of system calls” refers to changes to file system data and metadata caused by the system call, excluding changes to st_atime. See the stat(2) manual page.

Persistence guarantees

In all logging modes, VxFS is fully POSIX compliant. The effects of the fsync(2) and fdatasync(2) system calls are guaranteed to be persistent after the calls return. The persistence guarantees for data or metadata modified by write(2), writev(2), or pwrite(2) are not affected by the logging mount options. The effects of these system calls are guaranteed to be persistent only if the O_SYNC, O_DSYNC, VX_DSYNC, or VX_DIRECT flag, as modified by the convosync= `mount` option, has been specified for the file descriptor.

The behavior of NFS servers on a VxFS file system is unaffected by the log and tmplog mount options, but not delaylog. In all cases except for tmplog, VxFS complies with the persistency requirements of the NFS v2 and NFS v3 standard. Unless a UNIX application has been developed specifically for the VxFS file system

in log mode, it expects the persistence guarantees offered by most other file systems and experiences improved robustness when used with a VxFS file system mounted in delaylog mode. Applications that expect better persistence guarantees than that offered by most other file systems can benefit from the `log`, `mincache=`, and `closesync` `mount` options. However, most commercially available applications work well with the default VxFS `mount` options, including the delaylog mode.

The logiosize mode

The `logiosize=size` option enhances the performance of storage devices that employ a read-modify-write feature. If you specify logiosize when you mount a file system, VxFS writes the intent log in the least *`size`* bytes or a multiple of *`size`* bytes to obtain the maximum performance from such devices.

See mount_vxfs(1m) manual page.

The values for *`size`* can be 512, 1024, 2048, 4096, or 8192.

The nodatainlog mode

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 metadata 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.

The blkclear mode

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. This mode does not affect extending writes. A blkclear mode file system is approximately 10 percent slower than a standard mode VxFS file system, depending on the workload.

The mincache mode

The mincache mode has the following 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 open files can lose data. A mincache=closesync mode file system could be approximately 15 percent slower than a standard mode VxFS file system, depending on the workload.

The following describes where to use the mincache modes:

- The mincache=direct, mincache=unbuffered, and mincache=dsync modes are used in environments where applications have 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 are 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 are 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, as well as for the requirements for direct I/O.
- The mincache=direct, mincache=unbuffered, and mincache=dsync modes also flush file data on close as mincache=closesync does.

Because the mincache=direct, mincache=unbuffered, and mincache=dsync modes change non-synchronous I/O to synchronous I/O, throughput can substantially degrade for small to medium size files with most applications. Since the VX_DIRECT and VX_UNBUFFERED advisories do not allow any caching of data, applications that normally benefit from caching for reads 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, you can use the mincache=tmpcache mode. 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 the file is closed. When the mincache=tmpcache option is used, bad data can appear in a file that was being extended when a crash occurred.

The convosync mode

The convosync (convert osync) mode has the following suboptions:

- convosync=closesync

  Note: 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.

- convosync=delay
- convosync=direct
- convosync=dsync

  Note: The convosync=dsync option violates POSIX guarantees for synchronous I/O.

- convosync=unbuffered

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.

Warning: Be very careful when using the convosync=closesync or convosync=delay mode because they actually change synchronous I/O into non-synchronous I/O. Applications that use synchronous I/O for data reliability may fail if the system crashes and synchronously written data is lost.

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.

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.

The ioerror mode

This mode sets the policy for handling I/O errors on a mounted file system. I/O errors can occur while reading or writing file data or metadata. The file system can respond to these I/O errors either by halting or by gradually degrading. The ioerror option provides five policies that determine how the file system responds to the various errors. All policies limit data corruption, either by stopping the file system or by marking a corrupted inode as bad.

The policies are the following:

- disable
- nodisable
- wdisable
- mwdisable
- mdisable

The disable policy

If disable is selected, VxFS disables the file system after detecting any I/O error. You must then unmount the file system and correct the condition causing the I/O error. After the problem is repaired, run `fsck` and mount the file system again. In most cases, replay `fsck` to repair the file system. A full `fsck` is required only in cases of structural damage to the file system's metadata. Select disable in environments where the underlying storage is redundant, such as RAID-5 or mirrored disks.

The nodisable policy

If nodisable is selected, when VxFS detects an I/O error, it sets the appropriate error flags to contain the error, but continues running. Note that the degraded condition indicates possible data or metadata corruption, not the overall performance of the file system.

For file data read and write errors, VxFS sets the VX_DATAIOERR flag in the super-block. For metadata read errors, VxFS sets the VX_FULLFSCK flag in the super-block. For metadata write errors, VxFS sets the VX_FULLFSCK and VX_METAIOERR flags in the super-block and may mark associated metadata as bad on disk. VxFS then prints the appropriate error messages to the console.

See "File system response to problems" on page 225.

You should stop the file system as soon as possible and repair the condition causing the I/O error. After the problem is repaired, run `fsck` and mount the file system again. Select nodisable if you want to implement the policy that most closely resembles the error handling policy of the previous VxFS release.

The wdisable and mwdisable policies

If wdisable (write disable) or mwdisable (metadata-write disable) is selected, the file system is disabled or degraded, depending on the type of error encountered. Select wdisable or mwdisable for environments where read errors are more likely to persist than write errors, such as when using non-redundant storage. mwdisable is the default ioerror mount option for local mounts.

See the mount_vxfs(1M) manual page.

The mdisable policy

If mdisable (metadata disable) is selected, the file system is disabled if a metadata read or write fails. However the file system continues to operate if the failure is confined to data extents. mdisable is the default ioerror mount option for cluster mounts.

The largefiles|nolargefiles option

The section includes the following topics :

- See "Creating a file system with large files" on page 38.
- See "Mounting a file system with large files" on page 38.
- See "Managing a file system with large files" on page 38.

VxFS supports files larger than two terabytes. Files larger than 32 terabytes can be created only on 64-bit kernel operating systems and on a Veritas Volume Manager volume.

Note: Applications and utilities such as backup may experience problems if they are not aware of large files. In such a case, create your file system without large file capability.

Creating a file system with large files

To create a file system with a file capability, type the following command:

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

Specifying largefiles sets the largefiles flag. This lets the file system to hold files that are two terabytes or larger. This is the default option.

To clear the flag and prevent large files from being created, type the following command:

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

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.

Because the `mount` command defaults to match the current setting of the on-disk flag if specified without the `largefiles` or `nolargefiles` option, the best practice is not to specify either option. After a file system is mounted, you can use the `fsadm` utility to change the large files option.

Managing a file system with large files

Managing a file system with large files includes the following tasks:

- Determining the current status of the large files flag
- Switching capabilities on a mounted file system
- Switching capabilities on an unmounted file system

To determine the current status of the largefiles flag, type either of the following commands:

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

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

To switch capabilities on a mounted file system, type the following command;

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

To switch capabilities on an unmounted file system, type the following command:

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

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

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

The cio option

The `cio` (Concurent I/O) option specifies the file system to be mounted for concurrent readers and writers. Concurrent I/O is a licensed feature of VxFS. If `cio` is specified, but the feature is not licensed, the `mount` command prints an error message and terminates the operation without mounting the file system. The `cio` option cannot be disabled through a remount. To disable the `cio` option, the file system must be unmounted and mounted again without the `cio` option.

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.

To mount a desktop file system using options, type the following:

```
# 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, after an application has exited and its files are closed, no data is lost even if the system is immediately turned off.

To mount a temporary file system or to restore from backup, type the following:

```
 # 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. This could result in a file that contains corrupted data if the system crashes. Any file written 30 seconds or so before a crash may contain corrupted data or be missing if this mount combination is in effect. However, such a file system does significantly less disk writes than a log file system, and should have significantly better performance, depending on the application.

To mount a file system for synchronous writes, type the following:

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

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

Using kernel tunables

This section describes the following kernel tunable parameters in VxFS:

- Tuning inode table size
- vx_maxlink
- Veritas Volume Manager maximum I/O size

Tuning inode table size

Tuning the internal inode table size includes the following tasks:

- Increasing the internal inode table size
- Changing the size of the directory name lookup cache

VxFS caches inodes in an inode table. The tunable for VxFS to determine the number of entries in its inode table is *vxfs_ninode*.

VxFS 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 internal inode table size

1. Open the `/etc/system` file.
2. Update the following line in the `/etc/system` file:

```
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.

vx_maxlink

The VxFS *vx_maxlink* tunable determines the number of sub-directories that can be created under a directory.

A VxFS file system obtains the value of *vx_maxlink* from the system configuration file /etc/system. By default, *vx_maxlink* is 32K. To change the computed value of vx_maxlink, you can add an entry to the system configuration file. For example:

```
set vxfs:vx_maxlink = 65534
```

sets vx_maxlink to the maximum number of sub-directories. Valid values are 1 to 65534 (FFFE hexadecimal). Changes to *vx_maxlink* take effect after rebooting.

Veritas Volume Manager maximum I/O size

When using VxFS with 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.

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.

See the df_vxfs(1M) manual page.

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.

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 using the `cron` command.

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` command.

To determine the degree of fragmentation, use the following factors:

- Percentage of free space in extents of less than 8 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 has the following characteristics:

- Less than 1 percent of free space in extents of less than 8 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 has 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, reduce the interval between fsadm runs. If the degree of fragmentation is low, increase the interval between fsadm runs.

The “after” result is an indication of how well the reorganizer has performed. The degree of fragmentation should be close to the characteristics of an unfragmented file system. If not, it may be a good idea to resize the file system; 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 improves 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
```

Tuning I/O

The 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.

Note: The following tunables and the techniques work on a per file system basis. Use them judiciously based on the underlying device properties and characteristics of the applications that use the file system.

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 superior to a single disk. Typically, data streaming applications that access large files see the largest benefit from tuning the file system.

VxVM queries

VxVM receives the following queries during configuration:

- The file system queries VxVM to determine the geometry of the underlying volume and automatically sets the I/O parameters.

  Note: When using file systems in multiple volume sets, VxFS sets the VxFS tunables based on the geometry of the first component volume (volume 0) in the volume set.

- The `mkfs` command queries VxVM when the file system is created to automatically align the file system to the volume geometry. If the default alignment from `mkfs` is not acceptable, the `-o align=n` option can be used to override alignment information obtained from VxVM.
- The `mount` command 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.

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.

To print the values, type the following command:

```
# vxtunefs -p mount_point
```

The following is an example `tunefstab` file:

```
/dev/vx/dsk/userdg/netbackup
 read_pref_io=128k,write_pref_io=128k,read_nstream=4,write_nstream=4
 /dev/vx/dsk/userdg/metasave
 read_pref_io=128k,write_pref_io=128k,read_nstream=4,write_nstream=4
 /dev/vx/dsk/userdg/solbuild
 read_pref_io=64k,write_pref_io=64k,read_nstream=4,write_nstream=4
 /dev/vx/dsk/userdg/solrelease
 read_pref_io=64k,write_pref_io=64k,read_nstream=4,write_nstream=4
 /dev/vx/dsk/userdg/solpatch
 read_pref_io=128k,write_pref_io=128k,read_nstream=4,write_nstream=4
```

Tunable I/O parameters

Table 2-1 provides a list and description of these parameters.

Table 2-1 Tunable VxFS I/O parameters

Parameter	Description
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.

Table 2-1 Tunable VxFS I/O parameters *(continued)*

Parameter	Description
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. After 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 containing many large files, the 8K indirect extent size is too small. The files that get into indirect extents use many 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 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.

Table 2-1 Tunable VxFS I/O parameters *(continued)*

Parameter	Description
fcl_keeptime	Specifies the minimum amount of time, in seconds, that the VxFS File Change Log (FCL) keeps records in the log. When the oldest 8K block of FCL records have been kept longer than the value of fcl_keeptime, they are purged from the FCL and the extents nearest to the beginning of the FCL file are freed. This process is referred to as "punching a hole." Holes are punched in the FCL file in 8K chunks. If the fcl_maxalloc parameter is set, records are purged from the FCL if the amount of space allocated to the FCL exceeds fcl_maxalloc, even if the elapsed time the records have been in the log is less than the value of fcl_keeptime. If the file system runs out of space before fcl_keeptime is reached, the FCL is deactivated. Either or both of the fcl_keeptime or fcl_maxalloc parameters must be set before the File Change Log can be activated. fcl_keeptime does not apply to disk layout Versions 1 through 5.
fcl_maxalloc	Specifies the maximum amount of space that can be allocated to the VxFS File Change Log (FCL). The FCL file is a sparse file that grows as changes occur in the file system. When the space allocated to the FCL file reaches the fcl_maxalloc value, the oldest FCL records are purged from the FCL and the extents nearest to the beginning of the FCL file are freed. This process is referred to as "punching a hole." Holes are punched in the FCL file in 8K chunks. If the file system runs out of space before fcl_maxalloc is reached, the FCL is deactivated. Either or both of the fcl_maxalloc or fcl_keeptime parameters must be set before the File Change Log can be activated. fcl_maxalloc does not apply to disk lay out Versions 1 through 5.

Table 2-1 Tunable VxFS I/O parameters *(continued)*

Parameter	Description
fcl_winterval	Specifies the time, in seconds, that must elapse before the VxFS File Change Log (FCL) records a data overwrite, data extending write, or data truncate for a file. The ability to limit the number of repetitive FCL records for continuous writes to the same file is important for file system performance and for applications processing the FCL. fcl_winterval is best set to an interval less than the shortest interval between reads of the FCL by any application. This way all applications using the FCL can be assured of finding at least one FCL record for any file experiencing continuous data changes. fcl_winterval is enforced for all files in the file system. Each file maintains its own time stamps, and the elapsed time between FCL records is per file. This elapsed time can be overridden using the VxFS FCL sync public API. See the vxfs_fcl_sync(3) manual page. fcl_winterval does not apply to disk layout Versions 1 through 5.
hsm_write_ prealloc	For a file managed by a hierarchical storage management (HSM) application, hsm_write_prealloc preallocates disk blocks before data is migrated back into the file system. An HSM application usually migrates the data back through a series of writes to the file, each of which allocates a few blocks. By setting hsm_write_prealloc (hsm_write_prealloc=1), a sufficient number of disk blocks are allocated on the first write to the empty file so that no disk block allocation is required for subsequent writes. This improves the write performance during migration. The hsm_write_prealloc parameter is implemented outside of the DMAPI specification, and its usage has limitations depending on how the space within an HSM-controlled file is managed. It is advisable to use hsm_write_prealloc only when recommended by the HSM application controlling the file system.

Table 2-1 Tunable VxFS I/O parameters *(continued)*

Parameter	Description
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 with each allocation. See max_seqio_extent_size). 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 starts from a much larger initial size and the file system does not allocate a set of small extents at the start of file. Use this parameter only on file systems that have a very large average file size. On these file systems it results in fewer extents per file and less fragmentation. initial_extent_size is measured in file system blocks.
inode_aging_count	Specifies the maximum number of inodes to place on an inode aging list. Inode aging is used in conjunction with file system Storage Checkpoints to allow quick restoration of large, recently deleted files. The aging list is maintained in first-in-first-out (fifo) order up to maximum number of inodes specified by inode_aging_count. As newer inodes are placed on the list, older inodes are removed to complete their aging process. For best performance, it is advisable to age only a limited number of larger files before completion of the removal process. The default maximum number of inodes to age is 2048.
inode_aging_size	Specifies the minimum size to qualify a deleted inode for inode aging. Inode aging is used in conjunction with file system Storage Checkpoints to allow quick restoration of large, recently deleted files. For best performance, it is advisable to age only a limited number of larger files before completion of the removal process. Setting the size too low can push larger file inodes out of the aging queue to make room for newly removed smaller file inodes.

Table 2-1 Tunable VxFS I/O parameters *(continued)*

Parameter	Description
max_direct_iosz	The maximum size of a direct I/O request that are 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 are blocked until the amount of data being flushed decreases. Although this does not 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 performs aggressive read-ahead on the files.

Table 2-1 Tunable VxFS I/O parameters *(continued)*

Parameter	Description
read_ahead	The default for all VxFS read operations is to perform sequential read ahead. You can specify the read_ahead cache advisory to implement the VxFS enhanced read ahead functionality. This allows read aheads to detect more elaborate patterns (such as increasing or decreasing read offsets or multithreaded file accesses) in addition to simple sequential reads. You can specify the following values for read_ahead: 0—Disables read ahead functionality 1—Retains traditional sequential read ahead behavior 2—Enables enhanced read ahead for all reads The default is 1—VxFS detects only sequential patterns. read_ahead detects patterns on a per-thread basis, up to a maximum determined by vx_era_nthreads parameter. The default number of threads is 5, but you can change the default value by setting the vx_era_nthreads parameter in the /etc/system configuration file.

Table 2-1 Tunable VxFS I/O parameters *(continued)*

Parameter	Description
write_throttle	The write_throttle parameter is useful in special situations where a computer system has a combination of a large amount 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 generates 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 is zero, which puts no limit on the number of dirty pages per file. If non-zero, VxFS limits the number of dirty pages per file to write_throttle pages. The default value typically generates a large number 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 completes 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.

File system tuning guidelines

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

Note: VxFS does not query VxVM with multiple volume sets. To improve I/O performance when using multiple volume sets, use the `vxtunefs` command.

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 striped 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 use the following formula to issue read requests:

- read requests = read_nstream x 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, using unbuffered I/O avoids removing useful data out of the cache and lessens CPU overhead.

Chapter 3

Extent attributes

This chapter includes the following topics:

About extent attributes

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. 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 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 the following attribute properties:

- 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 incorporate the allocated space immediately

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 57.

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.

See "About VxFS I/O" on page 61.

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.

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

Fixed extent size

The VxFS default allocation policy uses a variety of methods 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 large extents 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 the following conditions:

- 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. 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.

See the mkfs_vxfs(1M) manual page.

See "About VxFS I/O" on page 61.

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 cause a file system to return an error message reporting insufficient space if no suitably sized (or aligned) extent is available. This can happen even if the file system has sufficient 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 that the reservation does not become a persistent attribute of the file. The 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. For users who do not have the appropriate privileges, there is a variant request that prevents such users from viewing uninitialized data.

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 setext(1) and getext(1) manual pages.

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

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 one of the following 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.

Chapter 4

VxFS I/O Overview

This chapter includes the following topics:

About VxFS I/O

VxFS processes two basic types of file system I/O:

- Sequential
- Random or I/O that is not sequential

For sequential I/O, VxFS employs a read-ahead policy by default when the application is reading data. For writing, it allocates contiguous blocks if possible. In most cases, VxFS handles I/O that is sequential through buffered I/O. VxFS handles random or nonsequential I/O using direct I/O without buffering.

VxFS provides a set of I/O cache advisories for use when accessing files.

See the *Veritas File System Programmer's Reference Guide*.

See the vxfsio(7) manual page.

Buffered and Direct I/O

VxFS responds with read-ahead for sequential read I/O. This results in buffered I/O. The data is prefetched and retained in buffers for the application. The data

buffers are commonly referred to as VxFS buffer cache. This is the default VxFS behavior.

On the other hand, direct I/O does not buffer the data when the I/O to the underlying device is completed. This saves system resources like memory and CPU usage. Direct I/O is possible only when alignment and sizing criteria are satisfied.

See "Direct I/O requirements" on page 62.

All the supported platforms have a VxFS buffered cache. Each platform also has either a page cache or its own buffer cache. These caches are commonly known as the file system caches.

Direct I/O does not use these caches. The memory used for direct I/O is discarded after the I/O is complete,

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.

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

Direct I/O requirements

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 requirements for direct I/O are as follows:

- The starting file offset must be aligned to a 512-byte boundary.
- The ending file offset must be aligned to a 512-byte boundary, or the length must be a multiple of 512 bytes.
- The memory buffer must start on an 8-byte boundary.

Direct I/O versus synchronous I/O

Because 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.

Direct I/O CPU overhead

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.

Discovered Direct I/O

Discovered Direct I/O is a file system tunable that is set using the `vxtunefs` command. 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.

See "Tuning I/O" on page 44..

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 I/O. For 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.

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.

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

Data synchronous I/O vs. synchronous I/O

Like direct I/O, the data synchronous I/O feature can provide significant application performance gains. Because 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.

Cache advisories

VxFS allows an application to set cache advisories for use when accessing files. VxFS cache advisories enable applications to help monitor the buffer cache and provide information on how better to tune the buffer cache to improve performance gain.

The basic function of the cache advisory is to let you know whether you could have avoided a later re-read of block X if the buffer cache had been a little larger. Conversely, the cache advisory can also let you know that you could safely reduce the buffer cache size without putting block X into jeopardy.

These advisories are in memory only and do not persist across reboots. Some advisories are currently maintained on a per-file, not a per-file-descriptor, basis. Only one set of advisories can be in effect for all accesses to the file. If two conflicting applications set different advisories, both must use the advisories that were last 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.

See the vxfsio(7) manual page.

Freezing and thawing a file system

Freezing a file system is a necessary step for obtaining a stable and consistent image of the file system at the volume level. Consistent volume-level file system

images can be obtained and used with a file system snapshot tool. The freeze operation flushes all buffers and pages in the file system cache that contain dirty metadata and user data. The operation then suspends any new activity on the file system until the file system is thawed.

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.

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.

Getting the I/O size

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.

See the vxtunefs(1M) and vxfsio(7) manual pages.

See "Tuning I/O" on page 44.

Chapter 5

Storage Checkpoints

This chapter includes the following topics:

About Storage Checkpoints

Veritas File System provides a Storage Checkpoint feature that 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 "Copy-on-write" on page 71.

Storage Checkpoints provide:

- Persistence through reboots and crashes.
- The ability for data to be immediately writeable by preserving the file system metadata, the directory hierarchy, and user data.

Storage Checkpoints are actually data objects that are managed and controlled by the file system. You can create, remove, and rename Storage Checkpoints because they are data objects with associated names.

See "How a Storage Checkpoint works" on page 69.

Unlike a disk-based mirroring technology that requires a separate storage space, Storage Checkpoints minimize the use of disk space by using a Storage Checkpoint within the same free space available to the file system.

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. A Storage Checkpoint preserves not only the name space (directory hierarchy) of the file system, but also the user data as it existed at the moment the file system image was captured.

You can use a Storage checkpoint in many ways. For example, you can use them to:

- Create a stable image of the file system that can be backed up to tape.
- Provide a mounted, on-disk backup of the file system so that end users can restore their own files in the event of accidental deletion. This is especially useful in a home directory, engineering, or email environment.
- Create a copy of an application's binaries before installing a patch to allow for rollback in case of problems.
- Create an on-disk backup of the file system in that can be used addition to a traditional tape-based backup to provide faster backup and restore capabilities.

How Storage Checkpoints differ from snapshots

Storage Checkpoints differ from Veritas File System snapshots in the following ways because they:

- Allow write operations to the Storage Checkpoint itself.
- Persist after a system reboot or failure.
- Share the same pool of free space as the file system.
- Maintain a relationship with other Storage Checkpoints by identifying changed file blocks since the last Storage Checkpoint.
- Have multiple, read-only Storage Checkpoints that reduce I/O operations and required storage space because the most recent Storage Checkpoint is the only one that accumulates updates from the primary file system.

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 that 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.

Storage Checkpoints can be taken in environments with a large number of files, such as file servers with millions of files, with little adverse impact on performance. Because the file system does not remain frozen during Storage Checkpoint creation, applications can access the file system even while the Storage Checkpoint is taken. However, Storage Checkpoint creation may take several minutes to complete depending on the number of files in the file system.

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, and the freezing process 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 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 Storage Checkpoint of multiple file systems simultaneously freezes the file systems, creates a Storage Checkpoint on all of the 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 there is a system crash while the operation is in progress.

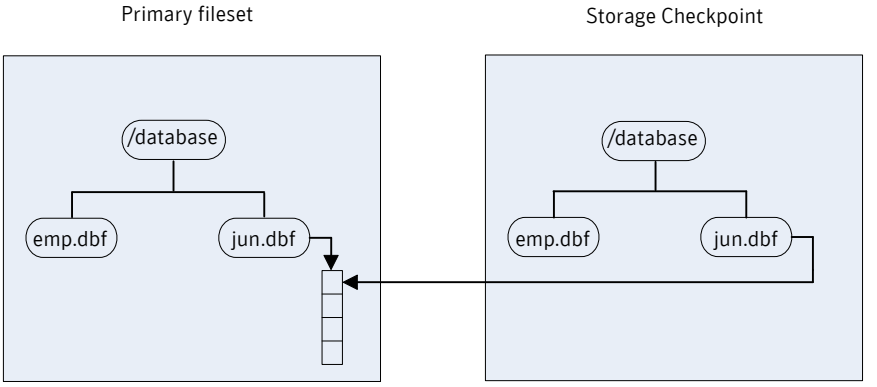
Note: The calling application is responsible for cleaning up Storage Checkpoints after a system crash.

A Storage Checkpoint of the primary fileset initially contains a pointer to the file system block map rather than to any actual data. The block map points to the data on the primary fileset.

Figure 5-1 shows the file system `/database` and its Storage Checkpoint.

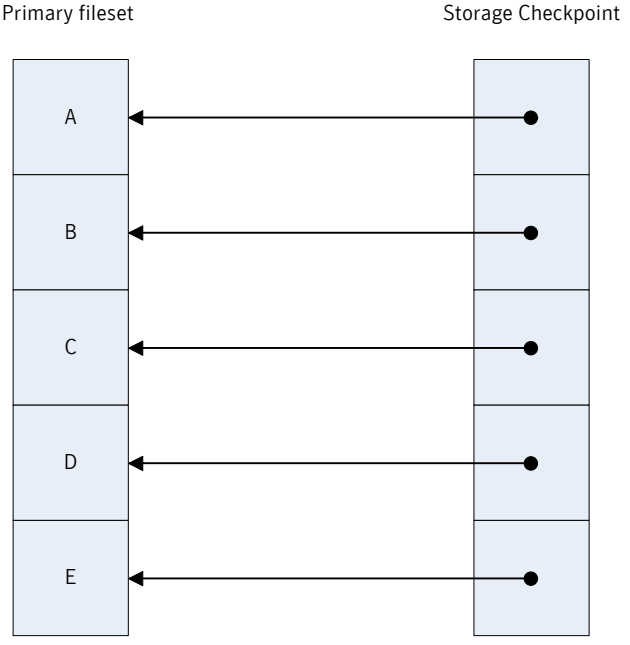
The Storage Checkpoint is logically identical to the primary fileset when the Storage Checkpoint is created, but it does not contain any actual data blocks.

Figure 5-1 Primary fileset and its Storage Checkpoint



In Figure 5-2, a square represents each block of the file system. This figure shows a Storage Checkpoint containing pointers to the primary fileset at the time the Storage Checkpoint is taken, as in Figure 5-1.

Figure 5-2 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.

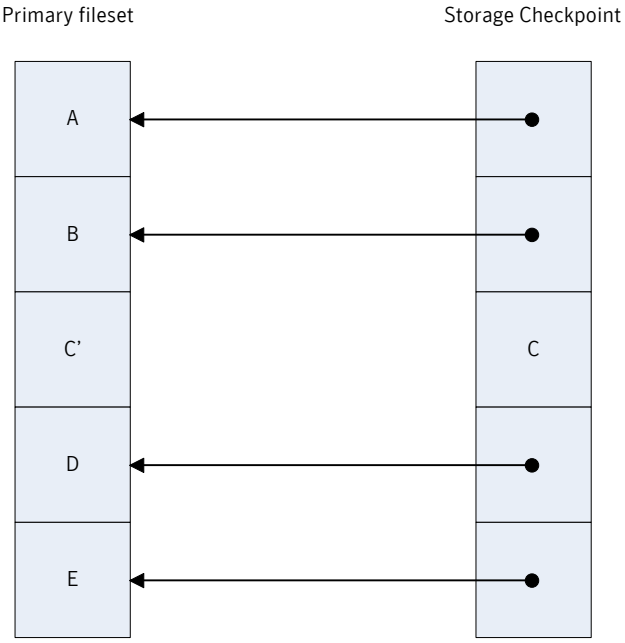
Copy-on-write

In Figure 5-3, 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 5-3 Updates to the primary fileset



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 that 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,

limit the life of data Storage Checkpoints to minimize the impact on system resources.

See "Showing the difference between a data and a nodata Storage Checkpoint" on page 79.

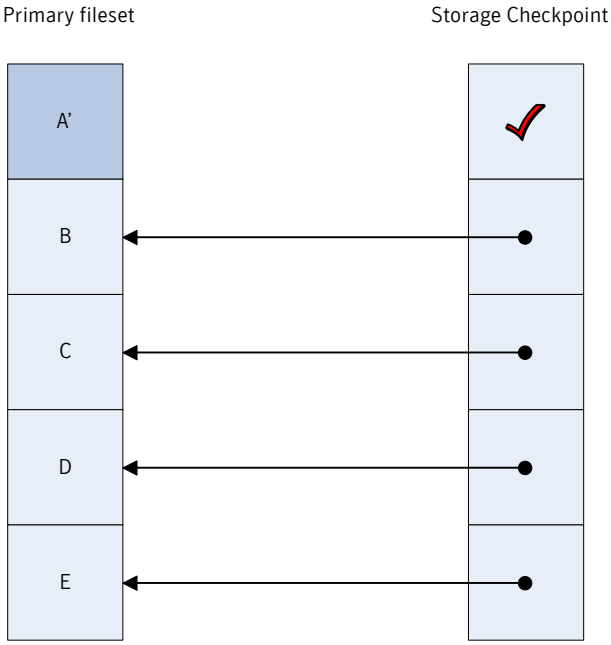
nodata Storage Checkpoints

A nodata Storage Checkpoint only contains file system metadata—no 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.

In Figure 5-4, the first block originally containing A is updated.

The original data is not copied to the storage checkpoint, but the changed block is marked in the Storage Checkpoint. The marker indicates which data has changed.

Figure 5-4 Updates to a nodata clone



See "Showing the difference between a data and a nodata Storage Checkpoint" on page 79.

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 86.

After encountering certain out-of-space (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

You can create Storage Checkpoints that cannot be mounted by using the `fsckptadm set nomount` command.

See the fsckptadm(1M) manual page.

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 the fsckptadm(1M) manual page.

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 (:).

Storage Checkpoints require some space for metadata on the volume or set of volumes specified by the file system allocation policy or Storage Checkpoint allocation policy. The `fsckptadm` utility displays an error if the volume or set of volumes does not have enough free space to contain the metadata. You can roughly approximate the amount of space required by the metadata using a method that depends on the disk layout version of the file system.

For disk layout Version 5 or prior, multiply the number of inodes (# of inodes) by the inode size (*inosize*) in bytes, and add 1 or 2 megabytes to get the approximate amount of space required. You can determine the number of inodes with the `fsckptadm` utility, and the inode size with the `mkfs` command:

```
# fsckptadm -v info '' /mnt0
  UNNAMED:
```

```
    ctime      =  Thu 3 Mar 2005 7:00:17 PM PST
    mtime      =  Thu 3 Mar 2005 7:00:17 PM PST
    flags      =  largefiles, mounted
  # of inodes      =  23872
  # of blocks      =  27867
  .
  .
  .
  # of overlay bmaps   =  0
# mkfs -m /mnt0
# mkfs -F vxfs -o \
  bsize=1024,version=7,inosize=256,logsize=65536,\
  largefiles /mnt0
```

In this example, the approximate amount of space required by the metadata is 7 or 8 megabytes (23,872 x 256 bytes, plus 1 or 2 megabytes).

For disk layout Version 6 or 7, multiply the number of inodes by 1 byte, and add 1 or 2 megabytes to get the approximate amount of space required. You can determine the number of inodes with the `fsckptadm` utility as above. Using the output from the example for disk layout Version 5, the approximate amount of space required by the metadata is just over one or two megabytes (23,872 x 1 byte, plus 1 or 2 megabytes).

Use the `fsvoladm` command to determine if the volume set has enough free space.

See the fsvoladm(1M) manual page.

```
# fsvoladm list /mnt0
devid     size       used    avail       name
0     20971520       8497658    12473862       mnt1
1     20971520       6328993    14642527       mnt2
2     20971520       4458462    16513058       mnt3
```

Creating a Storage Checkpoint

The following example shows the creation of a nodata Storage Checkpoint 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 3 Mar 2005 7:00:17 PM PST
 mtime   = Thu 3 Mar 2005 7:00:17 PM PST
 flags   = nodata, largefiles
```

See "Space management considerations" on page 86.

The following example shows the creation of a removable Storage Checkpoint named thu_8pm on /mnt0 and lists all Storage Checkpoints of the /mnt0 file system:

```
# fsckptadm -r create thu_8pm /mnt0
# fsckptadm list /mnt0
/mnt0
thu_8pm:
 ctime    = Thu 3 Mar 2005 8:00:19 PM PST
 mtime    = Thu 3 Mar 2005 8:00:19 PM PST
 flags    = largefiles, removable
thu_7pm:
 ctime    = Thu 3 Mar 2005 7:00:17 PM PST
 mtime    = Thu 3 Mar 2005 7:00:17 PM PST
 flags    = nodata, largefiles
```

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 is removed:

```
# fsckptadm remove thu_8pm /mnt0
# fsckptadm list /mnt0
/mnt0
thu_7pm:
 ctime    = Thu 3 Mar 2005 7:00:17 PM PST
 mtime    = Thu 3 Mar 2005 7:00:17 PM PST
 flags    = nodata, largefiles
# fsckptadm -s remove thu_7pm /mnt0
# fsckptadm list /mnt0
/mnt0
```

Accessing a Storage Checkpoint

You can mount Storage Checkpoints using the `mount` command with the mount option `-o ckpt=ckpt_name`.

See the mount_vxfs(1M) manual page.

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.

Warning: 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
```

Note: The `vol1` 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 mount this Storage Checkpoint automatically when the system starts up, put the following entries in the `/etc/vfstab` file:

```
#device to mount   device to     mount point FS   fsck mount   mount
                   fsck                      type pass at boot options
/dev/vx/dsk/fsvol/ /dev/vx/rdsk/ /fsvol      vxfs 1    yes     -
  vol1               fsvol/vol1
/dev/vx/dsk/fsvol/ -             /fsvol_     vxfs 0    yes     ckpt=may_23
  vol1:may_23                      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 cluster-wide if the file system that the Storage Checkpoint belongs to is also mounted cluster-wide. 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 the umount_vxfs(1M) manual page.

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 Storage Checkpoint pseudo devices because pseudo devices are part of 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 72.

You can use either the synchronous or asynchronous method to 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.

Showing the difference between a data and a nodata Storage Checkpoint

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

Note: A nodata Storage Checkpoint does not contain actual file data.

See “Converting a data Storage Checkpoint to a nodata Storage Checkpoint” on page 78.

To show the difference between Storage Checkpoints

1 Create a file system and mount it on `/mnt0`, as in the following example:

```
# mkfs -F vxfs /dev/vx/rdsk/dg1/test0

version 7 layout
134217728 sectors, 67108864 blocks of size 1024, log \

size  65536 blocks, largefiles supported
# mkfs -F /dev/vx/rdsk/dg1/test0 /mnt0
```

2 Create a small file with a known content, as in the following example.

```
# echo "hello, world" > /mnt0/file
```

3 Create a Storage Checkpoint and mount it on `/mnt0@5_30pm`, as in the following example:

```
# fsckptadm create ckpt@5_30pm /mnt0
# mkdir /mnt0@5_30pm
# mount -F vxfs -o ckpt=ckpt@5_30pm \
/dev/vx/dsk/dg1/test0:ckpt@5_30pm  /mnt0@5_30pm
```

4 Examine the content of the original file and the Storage Checkpoint file:

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

5 Change the content of the original file:

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

6 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
```

7 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/dg1/test0:ckpt@5_30pm /mnt0@5_30pm
```

8 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, 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 13 Jul 13 17:13 \
mnt0@5_30pm/file
# cat /mnt0@5_30pm/file
cat: /mnt0@5_30pm/file: I/O error

# ls -l /mnt0@5_30pm/file
-rw-r--r--   1 root     other 13 Jul 13 17:13 \
# cat /mnt0@5_30pm/file
cat: read error: No such file or directory

# ls -l /mnt0@5_30pm/file
-rw-r--r--   1 root     other 13 Jul 13 17:13 \
# cat /mnt0@5_30pm/file
cat: /mnt0@5_30pm/file: Input/output error

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

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

Converting multiple Storage Checkpoints

You can convert Storage Checkpoints to nodata Storage Checkpoints, when dealing with older Storage Checkpoints on the same file system.

To convert multiple Storage Checkpoints

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

```
# mkfs -F vxfs /dev/vx/rdsk/dg1/test0
version 7 layout
13417728 sectors, 67108864 blocks of size 1024, log \
size 65536 blocks largefiles supported
# mount -F vxfs /dev/vx/dsk/dg1/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 26 Jul 11:56:55 2004
  mtime                =  Mon 26 Jul 11:56:55 2004
  flags                =  largefiles
old:
  ctime                =  Mon 26 Jul 11:56:51 2004
  mtime                =  Mon 26 Jul 11:56:51 2004
  flags                =  largefiles
older:
  ctime                =   Mon 26 Jul 11:56:46 2004
  mtime                =  Mon 26 Jul 11:56:46 2004
  flags                =  largefiles
oldest:
  ctime                =  Mon 26 Jul 11:56:41 2004
  mtime                =  Mon 26 Jul 11:56:41 2004
  flags                =  largefiles
```

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 Checkpoints `old`, `older`, and `oldest`:

```
# fsckptadm -s set nodata latest /mnt0
UX:vxfs fsckptadm: ERROR: V-3-24632: Storage 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.

```
# fsckptadm set nodata latest /mnt0
```

5 List the Storage Checkpoints, as in the following example. You will see that the `latest` Storage Checkpoint is marked for conversion in the future.

```
# fsckptadm list /mnt0
/mnt0
latest:
  ctime                = Mon 26 Jul 11:56:55 2004
  mtime                = Mon 26 Jul 11:56:55
  flags                = nodata, largefiles, delayed
old:
  ctime                = Mon 26 Jul 11:56:51 2004
  mtime                = Mon 26 Jul 11:56:51 2004
  flags                = largefiles
older:
  ctime                = Mon 26 Jul 11:56:46 2004
  mtime                = Mon 26 Jul 11:56:46 2004
  flags                = largefiles
oldest:
  ctime                = Mon 26 Jul 11:56:41 2004
  mtime                = Mon 26 Jul 11:56:41 2004
  flags                = largefiles
```

Creating a delayed nodata Storage Checkpoint

You can combine the three 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.

To create a delayed nodata Storage Checkpoint

1 Remove the `latest` Storage Checkpoint.

```
# fsckptadm remove latest /mnt0
# fsckptadm list /mnt0
/mnt0
old:
  ctime                = Mon 26 Jul 11:56:51 2004
  mtime                = Mon 26 Jul 11:56:51 2004
  flags                = largefiles
older:
  ctime                = Mon 26 Jul 11:56:46 2004
  mtime                = Mon 26 Jul 11:56:46 2004
  flags                = largefiles
oldest:
  ctime                = Mon 26 Jul 11:56:41 2004
  mtime                = Mon 26 Jul 11:56:41 2004
  flags                = largefiles
```

2 Recreate the `latest` Storage Checkpoint as a nodata Storage Checkpoint.

```
# fsckptadm -n create latest /mnt0
# fsckptadm list /mnt0
/mnt0
latest:
  ctime                = Mon 26 Jul 12:06:42 2004
  mtime                = Mon 26 Jul 12:06:42 2004
  flags                = nodata, largefiles, delayed
old:
  ctime                = Mon 26 Jul 11:56:51 2004
  mtime                = Mon 26 Jul 11:56:51 2004
  flags                = largefiles
older:
  ctime                = Mon 26 Jul 11:56:46 2004
  mtime                = Mon 26 Jul 11:56:46 2004
  flags                = largefiles
oldest:
  ctime                = Mon 26 Jul 11:56:41 2004
  mtime                = Mon 26 Jul 11:56:41 2004
  flags                = largefiles
```

3 Convert the `oldest` Storage Checkpoint to a nodata Storage Checkpoint because no older Storage Checkpoints exist that contain data in the file system.

Note: This step can be done synchronously.

```
# fsckptadm -s set nodata oldest /mnt0
# fsckptadm list /mnt0
/mnt0
latest:
  ctime                = Mon 26 Jul 12:06:42 2004
  mtime                = Mon 26 Jul 12:06:42 2004
  flags                = nodata, largefiles, delayed
old:
  ctime                = Mon 26 Jul 11:56:51 2004
  mtime                = Mon 26 Jul 11:56:51 2004
  flags                = largefiles
older:
  ctime                = Mon 26 Jul 11:56:46 2004
  mtime                = Mon 26 Jul 11:56:46 2004
  flags                = largefiles
oldest:
  ctime                = Mon 26 Jul 11:56:41 2004
  mtime                = Mon 26 Jul 11:56:41 2004
  flags                = nodata, largefiles
```

4 Remove the `older` and `old` Storage Checkpoints.

```
# fsckptadm remove older /mnt0
# fsckptadm remove old /mnt0
# fsckptadm list /mnt0
/mnt0
latest:
  ctime                = Mon 26 Jul 12:06:42 2004
  mtime                = Mon 26 Jul 12:06:42 2004
  flags                = nodata, largefiles
oldest:
  ctime                = Mon 26 Jul 11:56:41 2004
  mtime                = Mon 26 Jul 11:56:41 2004
  flags                = nodata, largefiles
```

Note: After you remove the `older` and `old` Storage Checkpoints, 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:

Space management considerations

Several operations, such as removing or overwriting a file, can fail when a file system containing Storage Checkpoints runs out of space. 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`. from failing under an out-of-space (`ENOSPC`) condition.

See the rm(1m) manual page.

When the kernel automatically removes the Storage Checkpoints, it applies the following policies:

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

- Remove the oldest Storage Checkpoint first.

Restoring a file system from a Storage Checkpoint

Mountable data Storage Checkpoints on a consistent and undamaged file system can be used by backup and restore applications to restore either individual files or an entire file system. Restoration from Storage Checkpoints can also help recover incorrectly modified files, but typically cannot recover from hardware damage or other file system integrity problems.

Note: For hardware or other integrity problems, Storage Checkpoints must be supplemented by backups from other media.

Files can be restored by copying the entire file from a mounted Storage Checkpoint back to the primary fileset. To restore an entire file system, you can designate a mountable data Storage Checkpoint as the primary fileset using the `fsckpt_restore` command.

See the fsckpt_restore(1M) manual page.

When using the `fsckpt_restore` command to restore a file system from a Storage Checkpoint, all changes made to that file system after that Storage Checkpoint's creation date are permanently lost. The only Storage Checkpoints and data preserved are those that were created at the same time, or before, the selected Storage Checkpoint's creation. The file system cannot be mounted when `fsckpt_restore` is invoked.

Note: Files can be restored very efficiently by applications using the fsckpt_fbmap(3) library function to restore only modified portions of a files data.

Restoring a file from a Storage Checkpoint

The following example restores a file, `MyFile.txt`, which resides in your home directory, from the Storage Checkpoint `CKPT1` to the device `/dev/vx/dsk/vol-01`. The mount point for the device is `/home`.

To restore a file from a Storage Checkpoint

1 Create the Storage Checkpoint `CKPT1` of `/home`.

```
$ fckptadm create CKPT1 /home
```

2 Mount Storage Checkpoint `CKPT1` on the directory `/home/checkpoints/mar_4`.

```
$ mount -F vxfs -o ckpt=CKPT1 /dev/vx/dsk/dg1/vol- \
01:CKPT1 /home/checkpoints/mar_4
```

3 Delete the file `MyFile.txt` from your home directory.

```
$ cd /home/users/me
$ rm MyFile.txt
```

4 Go to the `/home/checkpoints/mar_4/users/me` directory, which contains the image of your home directory.

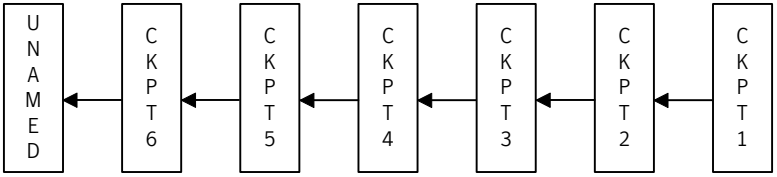
```
$ cd /home/checkpoints/mar_4/users/me
$ ls -l
-rw-r--r--  1  me  staff  14910  Mar 4  17:09  MyFile.txt
```

5 Copy the file `MyFile.txt` to your home directory.

```
$ cp MyFile.txt /home/users/me
$ cd /home/users/me
$ ls -l
-rw-r--r--   1 me  staff  14910  Mar 4  18:21  MyFile.txt
```

Restoring a file system from a Storage Checkpoint

The following example restores a file system from the Storage Checkpoint `CKPT3`. The filesets listed before the restoration show an unnamed root fileset and six Storage Checkpoints.



To restore a file system from a Storage Checkpoint

1 Run the `fsckpt_restore` command:

```
# fsckpt_restore -l /dev/vx/dsk/dg1/vol2
/dev/vx/dsk/dg1/vol2:
UNNAMED:
  ctime        = Thu 08 May 2004 06:28:26 PM PST
  mtime        = Thu 08 May 2004 06:28:26 PM PST
  flags        = largefiles, file system root
CKPT6:
   ctime        = Thu 08 May 2004 06:28:35 PM PST
  mtime        = Thu 08 May 2004 06:28:35 PM PST
  flags        = largefiles
CKPT5:
  ctime        = Thu 08 May 2004 06:28:34 PM PST
  mtime        = Thu 08 May 2004 06:28:34 PM PST
  flags        = largefiles, nomount
CKPT4:
  ctime        = Thu 08 May 2004 06:28:33 PM PST
  mtime        = Thu 08 May 2004 06:28:33 PM PST
  flags        = largefiles
CKPT3:
  ctime        = Thu 08 May 2004 06:28:36 PM PST
  mtime        = Thu 08 May 2004 06:28:36 PM PST
  flags        = largefiles
CKPT2:
  ctime        = Thu 08 May 2004 06:28:30 PM PST
  mtime        = Thu 08 May 2004 06:28:30 PM PST
  flags        = largefiles
CKPT1:
  ctime        = Thu 08 May 2004 06:28:29 PM PST
  mtime        = Thu 08 May 2004 06:28:29 PM PST
  flags        = nodata, largefiles
```

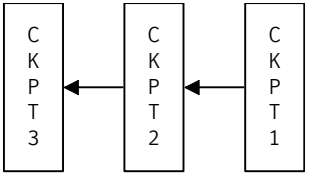
2 In this example, select the Storage Checkpoint `CKPT3` as the new root fileset:

```
Select Storage Checkpoint for restore operation
 or <Control/D> (EOF) to exit
 or <Return> to list Storage Checkpoints: CKPT3
CKPT3:
  ctime        = Thu 08 May 2004 06:28:31 PM PST
  mtime        = Thu 08 May 2004 06:28:36 PM PST
  flags        = largefiles
UX:vxfs fsckpt_restore: WARNING: V-3-24640: Any file system
changes or Storage Checkpoints made after
Thu 08  May 2004 06:28:31 PM PST will be lost.
```

3 Type **y** to restore the file system from CKPT3:

```
Restore the file system from Storage Checkpoint CKPT3 ?
(ynq) y
(Yes)
UX:vxfs fsckpt_restore: INFO: V-3-23760: File system
restored from CKPT3
```

If the filesets are listed at this point, it shows that the former UNNAMED root fileset and CKPT6, CKPT5, and CKPT4 were removed, and that CKPT3 is now the primary fileset. CKPT3 is now the fileset that will be mounted by default.



4 Run the fsckpt_restore command:

```
# fsckpt_restore -l /dev/vx/dsk/dg1/vol2
/dev/vx/dsk/dg1/vol2:
CKPT3:
  ctime        = Thu 08 May 2004 06:28:31 PM PST
  mtime        = Thu 08 May 2004 06:28:36 PM PST
  flags        = largefiles, file system root
CKPT2:
  ctime        = Thu 08 May 2004 06:28:30 PM PST
  mtime        = Thu 08 May 2004 06:28:30 PM PST
  flags        = largefiles
CKPT1:
  ctime        = Thu 08 May 2004 06:28:29 PM PST
  mtime        = Thu 08 May 2004 06:28:29 PM PST
  flags        = nodata, largefiles
Select Storage Checkpoint for restore operation
 or <Control/D> (EOF) to exit
 or <Return> to list Storage Checkpoints:
```

Storage Checkpoint quotas

VxFS provides options to the `fsckptadm` command interface to administer Storage Checkpoint quotas. Storage Checkpoint quotas set the following limits on the number of blocks used by all Storage Checkpoints of a primary file set:

hard limit	An absolute limit that cannot be exceeded. If a hard limit is exceeded, all further allocations on any of the Storage Checkpoints fail, but existing Storage Checkpoints are preserved.
soft limit	Must be lower than the hard limit. If a soft limit is exceeded, no new Storage Checkpoints can be created. The number of blocks used must return below the soft limit before more Storage Checkpoints can be created. An alert and console message are generated.

In case of a hard limit violation, various solutions are possible, enacted by specifying or not specifying the `-f` option for the `fsckptadm` utility.

See the fsckptadm(1M) manual page.

Specifying or not specifying the `-f` option has the following effects:

- If the `-f` option is not specified, one or many removable Storage Checkpoints are deleted to make space for the operation to succeed. This is the default solution.
- If the `-f` option is specified, all further allocations on any of the Storage Checkpoints fail, but existing Storage Checkpoints are preserved.

Note: Sometimes if a file is removed while it is opened by another process, the removal process is deferred until the last close. Because the removal of a file may trigger pushing data to a “downstream” Storage Checkpoint (that is, the next older Storage Checkpoint), a fileset hard limit quota violation may occur. In this scenario, the hard limit is relaxed to prevent an inode from being marked bad. This is also true for some asynchronous inode operations.

Chapter 6

Online backup using file system snapshots

This chapter includes the following topics:

About snapshot file systems

A snapshot file system is an exact image of a VxFS file system, referred to as the snapped file system, that provides a mechanism for making backups. The snapshot is a consistent view of the file system "snapped" at the point in time the snapshot is made. You can select files to back up from the snapshot using a standard utility such as `cpio` or `cp`, or back up the entire file system image using the `vxdump` or `fscat` utilities.

You use the mount command to create a snapshot file system; the `mkfs` command is not required. A snapshot file system is always read-only. A snapshot file system exists only as long as it and the snapped file system are mounted and ceases to exist when unmounted. A snapped file system cannot be unmounted until all of

its snapshots are unmounted. Although 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.

Note: A snapshot file system ceases to exist when unmounted. If mounted again, it is actually a fresh snapshot of the snapped file system. A snapshot file system must be unmounted before its dependent snapped file system can be unmounted. Neither the `fuser` command nor the `mount` command will indicate that a snapped file system cannot be unmounted because a snapshot of it exists.

On cluster file systems, snapshots can be created on any node in the cluster, and backup operations can be performed from that node. The snapshot of a cluster file system is accessible only on the node where it is created, that is, the snapshot file system itself cannot be cluster mounted.

See the *Veritas Storage Foundation Cluster File System Administrator's Guide*.

Snapshot file system backups

After a snapshot file system is created, the snapshot maintains a consistent backup of data in the snapped file system.

Backup programs, such as `cpio`, that back up a standard file system tree can be used without modification on a snapshot file system because the snapshot presents the same data as the snapped file system. Backup programs, such as `vxdump`, that access the disk structures of a file system require some modifications to handle a snapshot file system.

VxFS utilities recognize snapshot file systems and modify their behavior so that they operate the same way on snapshots as they do on standard file systems. Other backup programs that typically read the raw disk image cannot work on snapshots without altering the backup procedure.

These other backup programs can use the `fscat` command to obtain a raw image of the entire file system that is identical to an image obtainable by running a `dd` command on 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 that are obtainable 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 much different than the results that would be obtained by a `dd` or `read` command on the disk device of an active file system.

Creating a snapshot file system

You create a snapshot file system by using the `-o snapof=` option of the `mount` command. The `-o snapsize=` option may also be required if the device you are mounting does not identify the device size in its disk label, or if you want a size smaller than the entire device.

You must make the snapshot file system 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 runs out of blocks to hold copied data, the snapshot is disabled and further attempts to access the snapshot file system fail.

During periods of low activity (such as nights and weekends), a snapshot typically requires about two to six percent of the blocks of the snapped file system. During a period of high activity, the snapshot of a typical file system may require 15 percent of the blocks of the snapped file system. Most file systems do not turn over 15 percent of data in a single day. These approximate percentages tend to be lower for larger file systems and higher for smaller file systems. You can allocate blocks to a snapshot based on characteristics such as file system usage and duration of backups.

Warning: Any existing data on the device used for the snapshot is overwritten.

To create a snapshot file system

- Mount the file system with the `-o snapof=` option:

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

Backup examples

In the following examples, the `vxdump` utility is used to ascertain whether `/dev/vx/dsk/fsvol/vol1` is a snapshot mounted as `/backup/home` and does the appropriate work to get the snapshot data through the mount point.

These are typical examples of making a backup of a 300,000 block file system named `/home` using a snapshot file system on `/dev/vx/dsk/fsvol/vol1` with a snapshot mount point of `/backup/home`.

To create a backup using a snapshop file system

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

```
# mount -F vxfs -o snapof=/home,snapsize=100000 \
  /dev/vx/dsk/fsvol/vol1 /backup/home
# cd /backup
# find home -ctime -7 -depth -print | cpio -oc > \
  /dev/rmt/c0s0
# umount /backup/home
```

2 To do a level 3 backup of `/dev/vx/dsk/fsvol/vol1` and collect those files that have changed in the current directory:

```
# vxdump 3f - /dev/vx/dsk/fsvol/vol1 | vxrestore -xf -
```

3 To do a full backup of `/home`, which exists on disk `/dev/vx/dsk/fsvol/vol1`, and use `dd` to control blocking of output onto tape device using `vxdump`:

```
# mount -F vxfs -o snapof=/home,snapsize=100000 \
/dev/vx/dsk/fsvol/vol1 /backup/home
# vxdump f - /dev/vx/dsk/fsvol/vol1 | dd bs=128k > \
/dev/rmt/c0s0
```

Snapshot file system performance

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.

The performance of reads from the snapped file system are generally not affected. However, writes to the snapped file system, typically average two to three times as long as without a snapshot. This is because the initial write to a data block requires reading the old data, writing the data to the snapshot, and then writing the new data to the snapped file system. If there are multiple snapshots of the same snapped file system, writes are even slower. Only the initial write to a block experiences this delay, so operations such as 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 because the snapshot reads are slowed by 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. For example, a database

application running an online transaction processing (OLTP) workload on a snapped file system was measured at about 15 to 20 percent slower than a file system that was not snapped.

Differences between snapshots and Storage Checkpoints

While snapshots and Storage Checkpoints both create a point-in-time image of a file system and only the changed data blocks are updated, there are significant differences between the two technologies:

Table 6-1 Differences between snapshots and Storage Checkpoints

Snapshots	Storage Checkpoints
Require a separate device for storage	Reside on the same device as the original file system
Are read-only	Can be read-only or read-write
Are transient	Are persistent
Cease to exist after being unmounted	Can exist and be mounted on their own
Track changed blocks on the file system level	Track changed blocks on each file in the file system

Storage Checkpoints also serve as the enabling technology for two other Veritas features: Block-Level Incremental Backups and Storage Rollback, which are used extensively for backing up databases.

See "About Storage Checkpoints" on page 67.

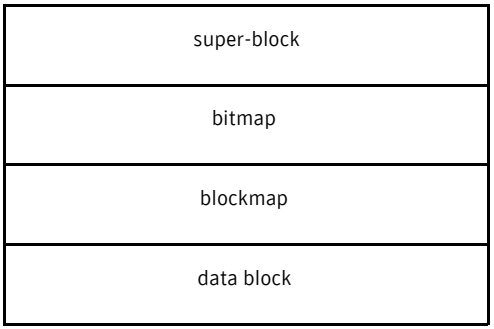
About snapshot file system disk structure

A snapshot file system consists of:

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

The following figure shows the disk structure of a snapshot file system.

Figure 6-1 The Snapshot Disk Structure



The super-block is similar to the super-block of a standard VxFS file system, but the magic number is different and many of the fields are not applicable.

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 references the copied block.

The blockmap 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 are filled by data copied from the snapped file system, starting from the beginning 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. After 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.

See “Freezing and thawing a file system” on page 64.

Initially, the snapshot file system satisfies read requests by 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 snapshot 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, instead of from block n on the snapped file system. This technique is called copy-on-write. 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, and so forth, are handled in this fashion so that the snapshot can present a consistent view of all file system structures on the snapped file system for the time when the snapshot was created. As data blocks are changed on the snapped file system, the snapshot gradually fills 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.

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

Chapter 7

Quotas

This chapter includes the following topics:

About quota limits

Veritas File System (VxFS) supports user and group 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 and groups to limit their usage.

You can set the following kinds of limits for each of the two resources:

hard limit	An absolute limit that cannot be exceeded under any circumstances.
soft limit	Must be lower than the hard limit, and 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.

Soft limits are typically used 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` command to set limits.

See "Using quotas" on page 104.

Although file and data block limits can be set individually for each user and group, the time limits apply to the file system as a whole. The quota limit information is associated with user and group IDs and is stored in a user or group quota file.

See "About quota files on Veritas File System" on page 102.

The quota soft limit can be exceeded when VxFS preallocates space to a file.

See "About extent attributes" on page 55.

About quota files on Veritas File System

A quotas file (named `quotas`) must exist in the root directory of a file system for any of the quota commands to work. For group quotas to work, there must be a `quotas.grp` file. The files in the file system's mount point are 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 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 and changes to quotas are registered in the internal quotas file. When quotas are turned off, the contents of the internal quotas file are copied into the external quotas file so that all data between the two files is synchronized.

VxFS supports group quotas in addition to user quotas. Just as user quotas limit file system resource (disk blocks and the number of inodes) usage on individual users, group quotas specify and limit resource usage on a group basis. As with user quotas, group quotas provide a soft and hard limit for file system resources. If both user and group quotas are enabled, resource utilization is based on the most restrictive of the two limits for a given user.

To distinguish between group and user quotas, VxFS quota commands use a `-g` and `-u` option. The default is user quotas if neither option is specified. One exception to this rule is when quotas are specified as a `mount` command option. In this case, both user and group quotas are enabled. Support for group quotas also requires a separate group quotas file. The VxFS group quota file is named `quotas.grp`. The VxFS user quotas file is named `quotas`. This name was used to distinguish it from the `quotas.user` file used by other file systems under Solaris.

About quota commands

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.

Note: Most of the quota commands in VxFS are similar to BSD quota commands. However, the `quotacheck` command is an exception; VxFS does not support an equivalent command.

See "About quota checking with Veritas File System" on page 103.

VxFS supports the following quota-related commands:

`vxedquota`	Edits quota limits for users and groups. 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`	Views quota limits and usage.
`vxquotaon`	Turns quotas on for a mounted VxFS file system.
`vxquotaoff`	Turns quotas off for a mounted VxFS file system.

Beside 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.

Note: When VxFS file systems are exported via NFS, the VxFS quota commands on the NFS client cannot query or edit quotas. You can use the VxFS quota commands on the server to query or edit quotas.

About quota checking with Veritas File System

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 and group. 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 the fsck_vxfs(1M) manual page.

A quota check generally reads information for each inode on 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

The VxFS quota commands are used to manipulate quotas.

Turning on quotas

To use the quota functionality on a file system, quotas must be turned on. You can turn quotas 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 for user quotas named `quotas`, and a file for group quotas named `quotas.grp` owned by root.

To turn on quotas

1 To turn on user and group quotas for a VxFS file system, enter:

```
# vxquotaon /mount_point
```

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

```
# vxquotaon -u /mount_point
```

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

```
# vxquotaon -g /mount_point
```

Turning on quotas at mount time

Quotas can be turned on with the `mount` command when you mount a file system.

To turn on quotas at mount time

1 To turn on user or group quotas for a file system at mount time, enter:

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

2 To turn on only user quotas, enter:

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

3 To turn on only group quotas, enter:

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

Editing user and group quotas

You can set up user and group quotas using the vxedquota command. You must have superuser privileges to edit quotas.

`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 are applicable only after quotas are turned on for a given file system.

To edit quotas

1 Specify the `-u` option to edit the quotas of one or more users specified by *username*:

```
# vxedquota [-u] username
```

Editing the quotas of one or more users is the default behavior if the `-u` option is not specified.

2 Specify the `-g` option to edit the quotas of one or more groups specified by *groupname*:

```
# vxedquota -g groupname
```

Modifying time limits

The soft and hard limits can be modified or assigned values. For any user or group, usage can never exceed the hard limit after quotas are turned on.

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

To modify time limits

1 Specify the `-t` option to modify time limits for any user:

```
# vxedquota [-u] -t
```

2 Specify the `-g` and `-t` options to modify time limits for any group:

```
# vxedquota -g -t
```

Viewing disk quotas and usage

Use the `vxquota` command to view a user's or group's disk quotas and usage on VxFS file systems.

To display disk quotas and usage

1 To display a user's quotas and disk usage on all mounted VxFS file systems where the quotas file exists, enter:

```
# vxquota -v [-u] username
```

2 To display a group's quotas and disk usage on all mounted VxFS file systems where the `quotas.grp` file exists, enter:

```
# vxquota -v -g groupname
```

Displaying blocks owned by users or groups

Use the `vxquot` command to display the number of blocks owned by each user or group in a file system.

To display the number of blocks owned by users or groups

1 To display the number of files and the space owned by each user, enter:

```
# vxquot [-u] -f filesystem
```

2 To display the number of files and the space owned by each group, enter:

```
# vxquot -g -f filesystem
```

Turning off quotas

Use the `vxquotaoff` command to turn off quotas.

To turn off quotas

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

```
# vxquotaoff /mount_point
```

2 To turn off only user quotas for a VxFS file system, enter:

```
# vxquotaoff -u /mount_point
```

3 To turn off only group quotas for a VxFS file system, enter:

```
# vxquotaoff -g /mount_point
```

Chapter 8

File Change Log

This chapter includes the following topics:

About File Change Log

The VxFS File Change Log (FCL) tracks changes to files and directories in a file system. Applications that typically use the FCL are usually required to:

- scan an entire file system or a subset
- discover changes since the last scan

These applications may include: backup utilities, webcrawlers, search engines, and replication programs.

Note: The FCL tracks when the data has changed and records the change type, but does not track the actual data changes. It is the responsibility of the application to examine the files to determine the changed data.

FCL functionality is a separately licensable feature.

See the *Veritas Storage Foundation Release Notes* .

About the File Change Log file

File Change Log records file system changes such as creates, links, unlinks, renaming, data appended, data overwritten, data truncated, extended attribute modifications, holes punched, and miscellaneous file property updates.

Note: FCL is supported only on disk layout Version 6 and later.

FCL stores changes in a sparse file in the file system namespace. The FCL file is located in `mount_point/lost+found/changelog`. The FCL file behaves like a regular file, but some operations are prohibited. The standard system calls open(2), lseek(2), read(2) and close(2) can access the data in the FCL, while the write(2), mmap(2) and rename(2) calls are not allowed.

Warning: Although some standard system calls are currently supported, the FCL file might be pulled out of the namespace in future VxFS release and these system calls may no longer work. It is recommended that all new applications be developed using the programmatic interface.

The FCL log file contains both the information about the FCL, which is stored in the FCL superblock, and the changes to files and directories in the file system, which is stored as FCL records.

See "File Change Log programmatic interface" on page 113.

In 4.1, the structure of the File Change Log file was exposed through the `/opt/VRTS/include/sys/fs/fcl.h` header file. In this release, the internal structure of the FCL file is opaque. The recommended mechanism to access the FCL is through the API described by the `/opt/VRTSfssdk/5.0/include/vxfsutil.h` header file.

The `/opt/VRTS/include/sys/fs/fcl.h` header file is included in this release to ensure that applications accessing the FCL with the 4.1 header file do not break. New applications should use the new FCL API described in `/opt/VRTSfssdk/5.0/include/vxfsutil.h`. Existing applications should also be modified to use the new FCL API.

With the addition of new record types, the FCL version in this release has been updated to 4. To provide backward compatibility for the existing applications, this release supports multiple FCL versions. Users now have the flexibility of specifying of specifying the FCL version for new FCLs. The default FCL version is 4.

See the fcladm(1M) man page.

File Change Log administrative interface

The FCL can be set up and tuned through the `fcladm` and `vxtunefs` VxFS administrative commands.

See the fcladm(1M) and vxtunefs(1M) manual pages.

The FCL keywords for `fcladm` are as follows:

clear	Disables the recording of the audit, open, close, and statistical events after it has been set.
dump	Creates a regular file image of the FCL file that can be downloaded too an off-host processing system. This file has a different format than the FCL file.
on	Activates the FCL on a mounted file system. VxFS 5.0 supports either FCL Versions 3 or 4. If no version is specified, the default is Version 4. Use `fcladm on` to specify the version.
print	Prints the contents of the FCL file starting from the specified offset.
restore	Restores the FCL file from the regular file image of the FCL file created by the `dump` keyword.
rm	Removes the FCL file. You must first deactivate the FCL with the `off` keyword, before you can remove the FCL file.
set	Enables the recording of events specified by the 'eventlist' option. See the fcladm(1M) manual page.
state	Writes the current state of the FCL to the standard output.
sync	Brings the FCL to a stable state by flushing the associated data of an FCL recording interval.

The FCL tunable parameters for `vxtunefs` are as follows:

fcl_keeptime — Specifies the duration in seconds that FCL records stay in the FCL file before they can be purged. The first records to be purged are the oldest ones, which are located at the beginning of the file. Additionally, records at the beginning of the file can be purged if allocation to the FCL file exceeds *fcl_maxalloc* bytes. The default value is 0. If the *fcl_maxalloc* parameter is set, records are purged from the FCL file if the amount of space allocated to the FCL file exceeds *fcl_maxalloc*. This is true even if the elapsed time the records have been in the log is less than the value of *fcl_keeptime*.

fcl_maxalloc — Specifies the maximum number of spaces in bytes to be allocated to the FCL file. When the space allocated exceeds *fcl_maxalloc*, a hole is punched at the beginning of the file. As a result, records are purged and the first valid offset (fc_foff) is updated. In addition, *fcl_maxalloc* may be violated if the oldest record has not reached *fcl_keeptime*.

The minimum value of *fcl_maxalloc* is 4 MB. The default value is *fs_size*/33.

fcl_winterval — Specifies the time in seconds that must elapse before the FCL records an overwrite, extending write, or a truncate. This helps to reduce the number of repetitive records in the FCL. The *fcl_winterval* timeout is per inode. If an inode happens to go out of cache and returns, its write interval is reset. As a result, there could be more than one write record for that file in the same write interval. The default value is 3600 seconds.

fcl_ointerval — The time interval in seconds within which subsequent opens of a file do not produce an additional FCL record. This helps to reduce the number of repetitive records logged in the FCL file. If the tracking of access information is also enabled, a subsequent file open even within the *fcl_ointerval* may produce a record, if it is opened by a different user. Similarly, if the inode is bumped out of cache, this may also produce more than one record within the same open interval.

The default value is 600 sec.

Either or both *fcl_maxalloc* and *fcl_keeptime* must be set to activate the FCL feature. The following are examples of using the `fcladm` command.

To activate FCL for a mounted file system, type the following:

```
# fcladm on mount_point
```

To deactivate the FCL for a mounted file system, type the following:

```
# fcladm off mount_point
```

To remove the FCL file for a mounted file system, on which FCL must be turned off, type the following:

```
# fcladm rm mount_point
```

To obtain the current FCL state for a mounted file system, type the following:

```
# fcladm state mount_point
```

To enable tracking of the file opens along with access information with each event in the FCL, type the following:

```
# fcladm set fileopen,accessinfo mount_point
```

To stop tracking file I/O statistics in the FCL, type the following:

```
# fcladm clear filestats mount_point
```

Print the on-disk FCL super-block in text format to obtain information about the FCL file by using offset 0. Because the FCL on-disk super-block occupies the first block of the FCL file, the first and last valid offsets into the FCL file can be determined by reading the FCL super-block and checking the *fc_foff* field. Enter:

```
# fcladm print 0 mount_point
```

To print the contents of the FCL in text format, of which the offset used must be 32-byte aligned, enter:

```
# fcladm print offset mount_point
```

File Change Log programmatic interface

VxFS provides an enhanced API to simplify reading and parsing the FCL file in two ways:

Simplified reading
: The API simplifies user tasks by reducing additional code needed to parse FCL file entries. In 4.1, to obtain event information such as a remove or link, the user was required to write additional code to get the name of the removed or linked file. In this release, the API allows the user to directly read an assembled record. The API also allows the user to specify a filter to indicate a subset of the event records of interest.

Backward compatibility	Providing API access for the FCL feature allows backward compatibility for applications. The API allows applications to parse the FCL file independent of the FCL layout changes. Even if the hidden disk layout of the FCL changes, the API automatically translates the returned data to match the expected output record. As a result, the user does not need to modify or recompile the application due to changes in the on-disk FCL layout.

See "Reverse path name lookup" on page 115.

The following sample code fragment reads the FCL superblock, checks that the state of the FCL is `VX_FCLS_ON`, issues a call to `vxfs_fcl_sync` to obtain a finishing offset to read to, determines the first valid offset in the FCL file, then reads the entries in 8K chunks from this offset. The section process fcl entries is what an application developer must supply to process the entries in the FCL file.

```
#include <stdint.h>
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/fcntl.h>
#include <errno.h>
#include <fcl.h>
#include <vxfsutil.h>
#define FCL_READSZ 8192
char* fclname = "/mnt/lost+found/changelog";
int read_fcl(fclname) char* fclname;
{
    struct fcl_sb fclsb;
    uint64_t off, lastoff;
    size_t size;
    char buf[FCL_READSZ], *bufp = buf;
    int fd;
    int err = 0;
    if ((fd = open(fclname, O_RDONLY)) < 0) {
        return ENOENT;
    }
    if ((off = lseek(fd, 0, SEEK_SET)) != 0) {
        close(fd);
        return EIO;
    }
    size = read(fd, &fclsb, sizeof (struct fcl_sb));
    if (size < 0) {
        close(fd);
```

```
        return EIO;
    }
    if (fclsb.fc_state == VX_FCLS_OFF) {
        close(fd);
        return 0;
    }
    if (err = vxfs_fcl_sync(fclname, &lastoff)) {
        close(fd);
        return err;
    }
    if ((off = lseek(fd, off_t, uint64_t)) != uint64_t) {
        close(fd);
        return EIO;
    }
    while (off < lastoff) {
        if ((size = read(fd, bufp, FCL_READSZ)) <= 0) {
            close(fd);
            return errno;
        }
        /* process fcl entries */
        off += size;
    }
    close(fd);
    return 0;
}
```

Reverse path name lookup

The reverse path name lookup feature obtains the full path name of a file or directory from the inode number of that file or directory. The inode number is provided as an argument to the `vxlsino` administrative command, or the vxfs_inotopath_gen(3) application programming interface library function.

The reverse path name lookup feature can be useful for a variety of applications, such as for clients of the VxFS File Change Log feature, in backup and restore utilities, and for replication products. Typically, these applications store information by inode numbers because a path name for a file or directory can be very long, thus the need for an easy method of obtaining a path name.

An inode is a unique identification number for each file in a file system. An inode contains the data and metadata associated with that file, but does not include the file name to which the inode corresponds. It is therefore relatively difficult to determine the name of a file from an inode number. The `ncheck` command provides

a mechanism for obtaining a file name from an inode identifier by scanning each directory in the file system, but this process can take a long period of time. The VxFS reverse path name lookup feature obtains path names relatively quickly.

Note: Because symbolic links do not constitute a path to the file, the reverse path name lookup feature cannot track symbolic links to files.

Because of the possibility of errors with processes renaming or unlinking and creating new files, it is advisable to perform a `lookup` or `open` with the path name and verify that the inode number matches the path names obtained.

See the vxlsino(1M), vxfs_inotopath_gen(3), and vxfs_inotopath(3) manual pages.

Chapter 9

Multi-volume file systems

This chapter includes the following topics:

About multi-volume support

VxFS provides support for multi-volume file systems when used in conjunction with the Veritas Volume Manager. Using multi-volume support (MVS), a single file system can be created over multiple volumes, each volume having its own properties. For example, it is possible to place metadata on mirrored storage while placing file data on better-performing volume types such as RAID-1+0 (striped and mirrored).

The MVS feature also allows file systems to reside on different classes of devices, so that a file system can be supported from both inexpensive disks and from expensive arrays. Using the MVS administrative interface, you can control which data goes on which volume types.

Note: MVS is available only on file systems using disk layout Version 6 or later.

See "About disk layouts" on page 275.

About volume types

VxFS utilizes two types of volumes, one of which contains only data, referred to as dataonly, and the other of which can contain metadata or data, referred to as metadataok.

Data refers to direct extents, which contain user data, of regular files and named data streams in a file system.

Metadata refers to all data allocated when the file system was created with the `mkfs` command or allocated at a later time. Metadata includes extents allocated to structural files, blocks reserved for the super block, the volume label, indirect extents, extents belonging to the File Change Log file, the history log file, extended attributes, directories, access control lists, and so on.

A volume availability flag is set to specify if a volume is dataonly or metadataok. The volume availability flag can be set, cleared, and listed with the `fsvoladm` command.

See the fsvoladm(1M) manual page.

Features implemented using multi-volume support

The following features can be implemented using multi-volume support:

- Controlling where files are stored can be selected at multiple levels so that specific files or file hierarchies can be assigned to different volumes. This

functionality is available in the Veritas File System Dynamic Storage Tiering (DST) feature.

See "About Dynamic Storage Tiering" on page 137.

- Placing the VxFS intent log on its own volume to minimize disk head movement and thereby increase performance. This functionality can be used to migrate from the Veritas QuickLog™ feature.
- Separating Storage Checkpoints so that data allocated to a Storage Checkpoint is isolated from the rest of the file system.
- Separating metadata from file data.
- Encapsulating volumes so that a volume appears in the file system as a file. This is particularly useful for databases that are running on raw volumes.
- Guaranteeing the availability of some volumes even when others are unavailable.

To use the multi-volume file system features, Veritas Volume Manager must be installed and the volume set feature must be accessible.

Volume availability

MVS guarantees the availability of some volumes even when others are unavailable. This allows you to mount a multi-volume file system even if one or more component dataonly volumes are missing.

The volumes are separated by whether metadata is allowed on the volume. An I/O error on a dataonly volume does not affect access to any other volumes. All VxFS operations that do not access the missing dataonly volume function normally, including:

- Mounting the multi-volume file system, regardless if the file system is read-only or read/write.
- Kernel operations.
- Performing a `fsck` replay. Logged writes are converted to normal writes if the corresponding volume is dataonly.
- Performing a full `fsck`.
- Using all other commands that do not access data on a missing volume.

Some operations that could fail if a dataonly volume is missing include:

- Reading or writing file data if the file's data extents were allocated from the missing dataonly volume.
- Using the `vxdump` command.

Volume availability is supported only on a file system with disk layout Version 7 or later.

Note: Do not mount a multi-volume system with the `ioerror=disable` or `ioerror=wdisable` mount options if the volumes have different availability properties. Symantec recommends the `ioerror=mdisable` mount option for cluster mounts and `ioerror=mwdisable` for local mounts.

About volume sets

Veritas Volume Manager exports a data object called a volume set. Unlike the traditional Volume Manager volume, which can be used for raw I/O access or to contain a file system, a volume set is a container for multiple different volumes. Each volume can have its own geometry.

The Volume Manager vxvset command is used to create and manage volume sets. Volume sets cannot be empty. When the last entry is removed, the volume set itself is removed.

Creating and managing volume sets

The following command examples show how to create and manage volume sets.

To create and manage volume sets

1 Create a new volume set from vol1:

```
# vxassist make vol1 10m
# vxvset make myvset vol1
```

2 Create two new volumes and add them to the volume set:

```
# vxassist make vol2 50m
# vxassist make vol3 50m
# vxvset addvol myvset vol2
# vxvset addvol myvset vol3
```

3 List the component volumes of the previously created volume set:

```
# vxvset list myvset
VOLUME    INDEX    LENGTH     STATE     CONTEXT
vol1      0        20480      ACTIVE    -
vol2      1        102400     ACTIVE    -
vol3      2        102400     ACTIVE    -
```

4 Use the `ls` command to see that when a volume set is created, the volumes contained by the volume set are removed from the namespace and are instead accessed through the volume set name:

```
# ls -l /dev/vx/rdsk/rootdg/myvset
1 root  root  108,70009 May 21 15:37 /dev/vx/rdsk/rootdg/myvset
```

5 Create a volume, add it to the volume set, and use the `ls` command to see that when a volume is added to the volume set, it is no longer visible in the namespace:

```
# vxassist make vol4 50m
# ls -l /dev/vx/rdsk/rootdg/vol4
crw-- 1 root root 108,70012 May 21 15:43
                            /dev/vx/rdsk/rootdg/vol4
# vxvset addvol myvset vol4
# ls -l /dev/vx/rdsk/rootdg/vol4
/dev/vx/rdsk/rootdg/vol4: No such file or directory
```

Creating multi-volume file systems

When a multi-volume file system is created, all volumes are dataonly, except volume zero. The volume availability flag of volume zero cannot be set to dataonly.

As metadata cannot be allocated from dataonly volumes, enough metadata space should be allocated using metadataok volumes. The "file system out of space" error occurs if there is insufficient metadata space available, even if the `df` command shows that there is free space in the file system. The `fsvoladm` command can be used to see the free space in each volume and set the availability flag of the volume.

With the exception of adding and deleting volumes, the file system commands operate the same on volumes within a volume set.

Example of creating a multi-volume file system

The following procedure is an example of creating a multi-volume file system.

To create a multi-volume file system

1 After a volume set is created, create a VxFS file system by specifying the volume set name as an argument to `mkfs`:

```
# mkfs -F vxfs /dev/vx/rdsk/rootdg/myvset
version 7 layout
327680 sectors, 163840 blocks of size 1024,
log size 1024 blocks largefiles supported
```

After the file system is created, VxFS allocates space from the different volumes within the volume set.

2 List the component volumes of the volume set using of the `fsvoladm` command:

```
# mount -F vxfs /dev/vx/dsk/rootdg/myvset /mnt1
# fsvoladm list /mnt1
devid   size     used   avail   name
0       10240    1280   8960    vol1
1       51200    16     51184   vol2
2       51200    16     51184   vol3
3       51200    16     51184   vol4
```

3 Add a new volume by adding the volume to the volume set, then adding the volume to the file system:

```
# vxassist make vol5 50m
# vxvset addvol myvset vol5
# fsvoladm add /mnt1 vol5 50m
# fsvoladm list /mnt1
devid   size     used   avail   name
0       10240    1300   8940    vol1
1       51200    16     51184   vol2
2       51200    16     51184   vol3
3       51200    16     51184   vol4
4       51200    16     51184   vol5
```

4 List the volume availability flags using the `fsvoladm` command:

```
# fsvoladm queryflags /mnt1
volname    flags
vol1       metadataok
vol2       dataonly
vol3       dataonly
vol4       dataonly
vol5       dataonly
```

5 Increase the metadata space in the file system using the `fsvoladm` command:

```
# fsvoladm clearflags dataonly /mnt1 vol2
# fsvoladm queryflags /mnt1
volname    flags
vol1       metadataok
vol2       metadataok
vol3       dataonly
vol4       dataonly
vol5       dataonly
```

Converting a single volume file system to a multi-volume file system

The following procedure converts a traditional, single volume file system, `/mnt1`, on a single volume `vol1` in the diskgroup `dg1` to a multi-volume file system.

To convert a single volume file system

1 Determine the version of the volume's diskgroup:

```
# vxdg list dg1 | grep version: | awk '{ print $2 }'
105
```

2 If the version is less than 110, upgrade the diskgroup:

```
# vxdg upgrade dg1
```

3 Determine the disk layout version of the file system:

```
# vxupgrade /mnt1
Version 4
```

4 If the disk layout version is less than 6, upgrade to Version 7:

```
# vxupgrade -n 7 /mnt1
```

5 Unmount the file system:

```
# umount /mnt1
```

6 Convert the volume into a volume set:

```
# vxvset -g dg1 make vset1 vol1
```

7 Edit the `/etc/vfstab` file to replace the volume device name, `vol1`, with the volume set name, `vset1`.

8 Mount the file system:

```
# mount -F vxfs /dev/vx/dsk/dg1/vset1 /mnt1
```

9 As necessary, create and add volumes to the volume set:

```
# vxassist -g dg1 make vol2 256M
# vxvset -g dg1 addvol vset1 vol2
```

10 As necessary, set the placement class tags on the volumes:

```
# vxvoladm -g dg1 settag vol1 vxfs.placement_class.tier1
# vxvoladm -g dg1 settag vol2 vxfs.placement_class.tier2
```

11 Add the volumes to the file system:

```
# fsvoladm add /mnt1 vol1 256m
# fsvoladm add /mnt1 vol2 256m
```

Removing a volume from a multi-volume file system

Use the `fsvoladm remove` command to remove a volume from a multi-volume file system. The `fsvoladm remove` command fails if an allocation policy exists that has only a target device.

Forcibly removing a volume

If you must forcibly remove a volume from a file system, such as if a volume is permanently destroyed and you want to clean up the dangling pointers to the lost

volume, use the `fsck -o zapvol=volname` command. The `zapvol` option performs a full file system check and zaps all inodes that refer to the specified volume. The `fsck` command prints the inode numbers of all files that the command destroys; the file names are not printed. The `zapvol` option only affects regular files if used on a dataonly volume. However, it could destroy structural files if used on a metadataok volume, which can make the file system unrecoverable. Therefore, the `zapvol` option should be used with caution on metadataok volumes.

Moving volume 0

You can remove volume 0 from a volume in a multi-volume file system and move volume 0 to another volume with the `vxassist move` command. The `vxassist` command creates any necessary temporary mirrors and cleans up the mirrors at the end of the operation.

To move volume 0

- Move volume 0:

```
# vxassist move vol1 !mydg
```

About allocation policies

To make full use of multi-volume support features, VxFS provides support for allocation policies that allow files or groups of files to be assigned to specified volumes within the volume set.

A policy specifies a list of volumes and the order in which to attempt allocations. A policy can be assigned to a file, a file system, or a Storage Checkpoint created from a file system. When policies are assigned to objects in the file system, you must specify how the policy maps to both metadata and file data. For example, if a policy is assigned to a single file, the file system must know where to place both the file data and metadata. If no policies are specified, the file system places data randomly.

Assigning allocation policies

The following example shows how to assign allocation policies. The example volume set contains two volumes from different classes of storage.

To assign allocation policies

1 List the volumes in the volume set:

```
# vxvset -g rootdg list myvset
VOLUME   INDEX   LENGTH    STATE   CONTEXT
vol1     0       102400    ACTIVE  -
vol2     1       102400    ACTIVE  -
```

2 Create a file system on the `myvset` volume set and mount it:

```
# mkfs -F vxfs /dev/vx/rdsk/rootdg/myvset
version 7 layout
204800 sectors, 102400 blocks of size 1024,
log size 1024 blocks
largefiles supported
# mount -F vxfs /dev/vx/dsk/rootdg/myvset /mnt1
```

3 Define two allocation policies called `datapolicy` and `metadatapolicy` to refer to the `vol1` and `vol2` volumes:

```
# fsapadm define /mnt1 datapolicy vol1
# fsapadm define /mnt1 metadatapolicy vol2
```

4 Assign the policies at the file system level. The data policy must be specified before the metadata policy:

```
# fsapadm assignfs /mnt1 datapolicy metadatapolicy
# fsvoladm list /mnt1
devid   size     used   avail   name
0       51200    1250   49950   vol1
1       51200    16     51184   vol2
```

The assignment of the policies on a file system-wide basis ensures that any metadata allocated is stored on the device with the policy `metadatapolicy` (`vol2`) and all user data is be stored on `vol1` with the associated `datapolicy` policy.

Querying allocation policies

Querying an allocation policy displays the definition of the allocation policy.

The following example shows how to query allocation policies. The example volume set contains two volumes from different classes of storage.

To query allocation policies

- Query the allocation policy:

```
# fsapadm query /mnt1 datapolicy
```

Assigning pattern tables to directories

A pattern table contains patterns against which a file's name and creating process' UID and GID are matched as a file is created in a specified directory. The first successful match is used to set the allocation policies of the file, taking precedence over inheriting per-file allocation policies.

See the fsapadm(1M) manual page.

The following example shows how to assign pattern tables to a directory in a volume set that contains two volumes from different classes of storage. The pattern table matches all files created in the directory `dir1` with the `.mp3` extension for any user or group ID and assigns the `mp3data` data policy and `mp3meta` metadata policy.

To assign pattern tables to directories

1. Define two allocation policies called `mp3data` and `mp3meta` to refer to the `vol1` and `vol2` volumes:

```
# fsapadm define /mnt1 mp3data vol1
# fsapadm define /mnt1 mp3meta vol2
```

2. Assign the pattern table:

```
# fsapadm assignfilepat dir1 *.mp3///mp3data/mp3meta/
```

Assigning pattern tables to file systems

A pattern table contains patterns against which a file's name and creating process' UID and GID are matched as a file is created in a directory. If the directory does not have its own pattern table or an inheritable allocation policy, the file system's pattern table takes effect. The first successful match is used to set the allocation policies of the file.

See the fsapadm(1M) manual page.

The following example shows how to assign pattern tables to a file system in a volume set that contains two volumes from different classes of storage. The pattern table is contained within the pattern file `mypatternfile`.

To assign pattern tables to directories

1 Define two allocation policies called `mydata` and `mymeta` to refer to the `vol1` and `vol2` volumes:

```
# fsapadm define /mnt1 mydata vol1
# fsapadm define /mnt1 mymeta vol2
```

2 Assign the pattern table:

```
# fsapadm assignfspat -F mypatternfile /mnt1
```

Allocating data

The following script creates a large number of files to demonstrate the benefit of allocating data:

```
i=1
while [ $i -lt 1000 ]
do
    dd if=/dev/zero of=/mnt1/$i bs=65536 count=1
    i=‘expr $i + 1‘
done
```

Before the script completes, `vol1` runs out of space even though space is still available on the `vol2` volume:

```
# fsvoladm list /mnt1
devid   size     used   avail   name
0       51200    51200  0       vol1
1       51200    221    50979   vol2
```

The solution is to assign an allocation policy that allocates user data from the `vol1` volume to `vol2` if space runs out.

You must have system administrator privileges to create, remove, change policies, or set file system or Storage Checkpoint level policies. Users can assign a pre-existing policy to their files if the policy allows that.

Policies can be inherited for new files. A file will inherit the allocation policy of the directory in which it resides if you run the `fsapadm assignfile -f inherit` command on the directory.

Allocating data from vol1 to vol2

- Assign an allocation policy that allocates user data from `vol1` to `vol2` if space runs out on `vol1`:

```
# fsapadm define /mnt1 datapolicy vol1 vol2
```

Volume encapsulation

Multi-volume support enables the ability to encapsulate an existing raw volume and make the volume contents appear as a file in the file system.

Encapsulating a volume involves the following actions:

- Adding the volume to an existing volume set.
- Adding the volume to the file system using `fsvoladm`

Encapsulating a volume

The following example illustrates how to encapsulate a volume.

To encapsulate a volume

1 List the volumes:

```
# vxvset list myvset
VOLUME   INDEX   LENGTH   STATE   CONTEXT
vol1     0       102400   ACTIVE  -
vol2     1       102400   ACTIVE  -
```

The volume set has two volumes.

2 Create a third volume and copy the passwd file to the third volume:

```
# vxassist make dbvol 100m
# dd if=/etc/passwd of=/dev/vx/rdsk/rootdg/dbvol count=1
1+0 records in
1+0 records out
```

The third volume will be used to demonstrate how the volume can be accessed as a file, as shown later.

3 Create a file system on the volume set:

```
# mkfs -F vxfs /dev/vx/rdsk/rootdg/myvset
version 7 layout
204800 sectors, 102400 blocks of size 1024,
log size 1024 blocks
largefiles supported
```

4 Mount the volume set:

```
# mount -F vxfs /dev/vx/dsk/rootdg/myvset /mnt1
```

5 Add the new volume to the volume set:

```
# vxvset addvol myvset dbvol
```

6 Encapsulate `dbvol`:

```
# fsvoladm encapsulate /mnt1/dbfile dbvol 100m
# ls -l /mnt1/dbfile
-rw------- 1 root other 104857600 May 22 11:30 /mnt1/dbfile
```

7 Examine the contents of `dbfile` to see that it can be accessed as a file:

```
# head -2 /mnt1/dbfile
root:x:0:1:Super-User:/:/sbin/sh
daemon:x:1:1::/:
```

The passwd file that was written to the raw volume is now visible in the new file.

Note: If the encapsulated file is changed in any way, such as if the file is extended, truncated, or moved with an allocation policy or resized volume, or the volume is encapsulated with a bias, the file cannot be de-encapsulated.

Deencapsulating a volume

The following example illustrates how to deencapsulate a volume.

To deencapsulate a volume

1 List the volumes:

```
# vxvset list myvset
VOLUME    INDEX    LENGTH    STATE    CONTEXT
vol1      0        102400    ACTIVE   -
vol2      1        102400    ACTIVE   -
dbvol     2        102400    ACTIVE   -
```

The volume set has three volumes.

2 Deencapsulate `dbvol`:

```
# fsvoladm deencapsulate /mnt1/dbfile
```

Reporting file extents

MVS feature provides the capability for file-to-volume mapping and volume-to-file mapping via the `fsmap` and `fsvmap` commands. The `fsmap` command reports the volume name, logical offset, and size of data extents, or the volume name and size of indirect extents associated with a file on a multi-volume file system. The `fsvmap` command maps volumes to the files that have extents on those volumes.

See the fsmap(1M) and fsvmap(1M) manual pages.

The `fsmap` command requires `open`() permission for each file or directory specified. Root permission is required to report the list of files with extents on a particular volume.

Examples of reporting file extents

The following examples show typical uses of the `fsmap` and `fsvmap` commands.

Using the fsmap command

- Use the `find` command to descend directories recursively and run `fsmap` on the list of files:

```
# find . | fsmap -
Volume  Extent Type    File
vol2    Data           ./file1
vol1    Data           ./file2
```

Using the fsvmap command

1 Report the extents of files on multiple volumes:

```
# fsvmap /dev/vx/rdsk/fstest/testvset vol1 vol2
vol1  /.
vol1  /ns2
vol1  /ns3
vol1  /file1
vol2  /file1
vol2  /file2
```

2 Report the extents of files that have either data or metadata on a single volume in all Storage Checkpoints, and indicate if the volume has file system metadata:

```
# fsvmap -mvC /dev/vx/rdsk/fstest/testvset vol1
Meta  Structural   vol1  //volume has filesystem metadata//
Data  UNNAMED      vol1  /.
Data  UNNAMED      vol1  /ns2
Data  UNNAMED      vol1  /ns3
Data  UNNAMED      vol1  /file1
Meta  UNNAMED      vol1  /file1
```

Load balancing

An allocation policy with the balance allocation order can be defined and assigned to files that must have their allocations distributed at random between a set of specified volumes. Each extent associated with these files are limited to a maximum size that is defined as the required chunk size in the allocation policy. The distribution of the extents is mostly equal if none of the volumes are full or disabled.

Load balancing allocation policies can be assigned to individual files or for all files in the file system. Although intended for balancing data extents across volumes, a load balancing policy can be assigned as a metadata policy if desired, without any restrictions.

Note: If a file has both a fixed extent size set and an allocation policy for load balancing, certain behavior can be expected. If the chunk size in the allocation policy is greater than the fixed extent size, all extents for the file are limited by the chunk size. For example, if the chunk size is 16 MB and the fixed extent size is 3 MB, then the largest extent that satisfies both the conditions is 15 MB. If the fixed extent size is larger than the chunk size, all extents are limited to the fixed extent size. For example, if the chunk size is 2 MB and the fixed extent size is 3 MB, then all extents for the file are limited to 3 MB.

Defining and assigning a load balancing allocation policy

The following example defines a load balancing policy and assigns the policy to the file, `/mnt/file.db`.

To define and assign the policy

1 Define the policy by specifying the `-o balance` and `-c` options:

```
# fsapadm define -o balance -c 2m /mnt loadbal vol1 vol2 vol3 vol4
```

2 Assign the policy:

```
# fsapadm assign /mnt/filedb loadbal meta
```

Rebalancing extents

Extents can be rebalanced by strictly enforcing the allocation policy. Rebalancing is generally required when volumes are added or removed from the policy or when the chunk size is modified. When volumes are removed from the volume set, any extents on the volumes being removed are automatically relocated to other volumes within the policy.

The following example redefines a policy that has four volumes by adding two new volumes, removing an existing volume, and enforcing the policy for rebalancing.

To rebalance extents

1 Define the policy by specifying the `-o balance` and `-c` options:

```
# fsapadm define -o balance -c 2m /mnt loadbal vol1 vol2 vol4 \
vol5 vol6
```

2 Assign the policy:

```
# fsapadm enforcefile -f strict /mnt/filedb
```

Converting a multi-volume file system to a single volume file system

Because data can be relocated among volumes in a multi-volume file system, you can convert a multi-volume file system to a traditional, single volume file system by moving all file system data onto a single volume. Such a conversion is useful to users who would like to try using a multi-volume file system or Dynamic Storage Tiering, but are not committed to using a multi-volume file system permanently.

See "About Dynamic Storage Tiering" on page 137.

There are three restrictions to this operation:

- The single volume must be the first volume in the volume set
- The first volume must have sufficient space to hold all of the data and file system metadata
- The volume cannot have any allocation policies that restrict the movement of data

Converting to a single volume file system

The following procedure converts an existing multi-volume file system, `/mnt1`, of the volume set `vset1`, to a single volume file system, `/mnt1`, on volume `vol1` in diskgroup `dg1`.

Note: Steps 5, 6, and 8 are optional, and can be performed if you prefer to remove the wrapper of the volume set object.

Converting to a single volume file system

1 Determine if the first volume in the volume set, which is identified as device number 0, has the capacity to receive the data from the other volumes that will be removed:

```
# df /mnt1
/mnt1  (/dev/vx/dsk/dg1/vol1):16777216 blocks  3443528 files
```

2 If the first volume does not have sufficient capacity, grow the volume to a sufficient size:

```
# fsvoladm resize /mnt1 vol1 150g
```

3 Remove all existing allocation policies:

```
# fsppadm unassign /mnt1
```

4 Remove all volumes except the first volume in the volume set:

```
# fsvoladm remove /mnt1 vol2
# vxvset -g dg1 rmvol vset1 vol2
# fsvoladm remove /mnt1 vol3
# vxvset -g dg1 rmvol vset1 vol3
```

Before removing a volume, the file system attempts to relocate the files on that volume. Successful relocation requires space on another volume, and no allocation policies can be enforced that pin files to that volume. The time for the command to complete is proportional to the amount of data that must be relocated.

5 Unmount the file system:

```
# umount /mnt1
```

6 Remove the volume from the volume set:

```
# vxvset -g dg1 rmvol vset1 vol1
```

7 Edit the `/etc/vfstab` file to replace the volume set name, `vset1`, with the volume device name, `vol1`.

8 Mount the file system:

```
# mount -F vxfs /dev/vx/dsk/dg1/vol1 /mnt1
```

Chapter 10

Dynamic Storage Tiering

This chapter includes the following topics:

About Dynamic Storage Tiering

VxFS uses multi-tier online storage via the Dynamic Storage Tiering (DST) feature, which functions on top of multi-volume file systems. Multi-volume file systems are file systems that occupy two or more virtual volumes. The collection of volumes is known as a volume set, and is made up of disks or disk array LUNs belonging to a single Veritas Volume Manager (VxVM) disk group. A multi-volume file system presents a single name space, making the existence of multiple volumes transparent to users and applications. Each volume retains a separate identity for administrative purposes, making it possible to control the locations to which individual files are directed.

See "About multi-volume support" on page 118.

Note: Some of the commands have changed or removed between the 4.1 release and the 5.0 release to make placement policy management more user-friendly. The following are the commands that have been removed: `fsrpadm`, `fsmove`, and `fssweep`. The output of the `queryfile`, `queryfs`, and `list` options of the `fsapadm` command now print the allocation order by name instead of number.

DST allows administrators of multi-volume VxFS file systems to manage the placement of files on individual volumes in a volume set by defining placement policies that control both initial file location and the circumstances under which existing files are relocated. These placement policies cause the files to which they apply to be created and extended on specific subsets of a file system's volume set, known as placement classes. The files are relocated to volumes in other placement classes when they meet the specified naming, timing, access rate, and storage capacity-related conditions.

You make a VxVM volume part of a placement class by associating a volume tag with it. For file placement purposes, VxFS treats all of the volumes in a placement class as equivalent, and balances space allocation across them. A volume may have more than one tag associated with it. If a volume has multiple tags, the volume belongs to multiple placement classes and is subject to allocation and relocation policies that relate to any of the placement classes. Multiple tagging should be used carefully.

See "Placement classes" on page 139.

VxFS imposes no capacity, performance, availability, or other constraints on placement classes. Any volume may be added to any placement class, no matter what type the volume has nor what types other volumes in the class have. However, a good practice is to place volumes of similar I/O performance and availability in the same placement class.

Note: Dynamic Storage Tiering is a licensed feature. You must purchase a separate license key for DST to operate. See the *Veritas Storage Foundation Release Notes*.

The *Using Dynamic Storage Tiering* Symantec Yellow Book provides additional information regarding the Dynamic Storage Tiering feature, including the value of DST and best practices for using DST. You can download *Using Dynamic Storage Tiering* from the following webpage:

http://www.symantec.com/enterprise/yellowbooks/index.jsp

Placement classes

A placement class is a Dynamic Storage Tiering attribute of a given volume in a volume set of a multi-volume file system. This attribute is a character string, and is known as a volume tag. A volume may have different tags, one of which could be the placment class. The placement class tag makes a volume distinguishable by DST.

Volume tags are organized as hierarchical name spaces in which the levels of the hierarchy are separated by periods. By convention, the uppermost level in the volume tag hierarchy denotes the Storage Foundation component or application that uses a tag, and the second level denotes the tag's purpose. DST recognizes volume tags of the form `vxfs.placement_class.`*`class_name`*. The prefix `vxfs` identifies a tag as being associated with VxFS. `placement_class` identifies the tag as a file placement class used by DST. *`class_name`* represents the name of the file placement class to which the tagged volume belongs. For example, a volume with the tag **vxfs.placement_class.tier1** belongs to placement class tier1. Administrators use the `vxvoladm` command to associate tags with volumes.

See the vxadm(1M) manual page.

VxFS policy rules specify file placement in terms of placement classes rather than in terms of individual volumes. All volumes that belong to a particular placement class are interchangeable with respect to file creation and relocation operations. Specifying file placement in terms of placement classes rather than in terms of specific volumes simplifies the administration of multi-tier storage in the following ways:

- Adding or removing volumes does not require a file placement policy change. If a volume with a tag value of **vxfs.placement_class.tier2** is added to a file system's volume set, all policies that refer to tier2 immediately apply to the newly added volume with no administrative action. Similarly, volumes can be evacuated, that is, have data removed from them, and be removed from a file system without a policy change. The active policy continues to apply to the file system's remaining volumes.
- File placement policies are not specific to individual file systems. A file placement policy can be assigned to any file system whose volume set includes volumes tagged with the tag values (placement classes) named in the policy. This property makes it possible for data centers with large numbers of servers to define standard placement policies and apply them uniformly to all servers with a single administrative action.

Tagging volumes as placement classes

The following example tags the `vsavola` volume as placement class `tier1`, `vsavolb` as placement class `tier2`, `vsavolc` as placement class `tier3`, and `vsavold` as placement class `tier4` using the `vxadm` command.

To tag volumes

- Tag the volumes as placement classes:

```
# vxvoladm -g cfsdg settag vsavola vxfs.placement_class.tier1
# vxvoladm -g cfsdg settag vsavolb vxfs.placement_class.tier2
# vxvoladm -g cfsdg settag vsavolc vxfs.placement_class.tier3
# vxvoladm -g cfsdg settag vsavold vxfs.placement_class.tier4
```

Listing placement classes

Placement classes are listed using the `vxvoladm listtag` command.

See the vxvoladm(1M) manual page.

The following example lists all volume tags, including placment classes, set on a volume vsavola in the diskgroup cfsdg.

To list placement classes

- List the volume tags, including placement classes:

```
# vxvoladm -g cfsdg listtag vsavola
```

Administering placement policies

A VxFS file placement policy document contains rules by which VxFS creates, relocates, and deletes files, but the placement policy does not refer to specific file systems or volumes. You can create a file system's active file placement policy by assigning a placement policy document to the file system via the `fsppadm` command or the GUI.

See the fsppadm(1M) manual page.

At most, one file placement policy can be assigned to a VxFS file system at any time. A file system may have no file placement policy assigned to it, in which case VxFS allocates space for new files according to its own internal algorithms.

In systems with Storage Foundation Management Server (SFMS) software installed, file placement policy information is stored in the SFMS database. The SFMS database contains both XML policy documents and lists of hosts and file systems

for which each document is the current active policy. When a policy document is updated, SFMS can assign the updated document to all file systems whose current active policies are based on that document. By default, SFMS does not update file system active policies that have been created or modified locally, that is by the hosts that control the placement policies' file systems. If a SFMS administrator forces assignment of a placement policy to a file system, the file system's active placement policy is overwritten and any local changes that had been made to the placement policy are lost.

Assigning a placement policy

The following example uses the `fsppadm assign` command to assign the file placement policy represented in the XML policy document `/tmp/policy1.xml` for the file system at mount point `/mnt1`.

To assign a placement policy

- Assign a placement policy to a file system:

```
# fsppadm assign /mnt1 /tmp/policy1.xml
```

Unassigning a placement policy

The following example uses the `fsppadm unassign` command to unassign the active file placement policy from the file system at mount point `/mnt1`.

To unassign a placement policy

- Unassign the placement policy from a file system:

```
# fsppadm unassign /mnt1
```

Analyzing the space impact of enforcing a placement policy

The following example uses the `fsppadm analyze` command to analyze the impact if the enforce operation was performed on the mount point `/mnt1`. The command builds the I/O temperature database if necessary.

To analyze the space impact of enforcing a placement policy

- Analyze the impact:

```
# fsppadm analyze -i /mnt1
```

Querying which files will be affected by enforcing a placement policy

The following example uses the `fsppadm query` command to generate a list of files that will be affected by enforcing a placement policy. The command provides details about where the files currently reside, to where the files will be relocated, and which rule in the placement policy applies to the files.

To query which files will be affected by enforcing a placement policy

- Query the files:

```
# fsppadm query /mnt1/dir1/dir2 /mnt2 /mnt1/dir3
```

Enforcing a placement policy

Enforcing a placement policy for a file system requires that the policy be assigned to the file system. You must assign a placement policy before it can be enforced.

Enforce operations are logged in a hidden file, `.__fsppadm_enforce.log`, in the `lost+found` directory of the mount point. This log file contains details such as files' previous locations, the files' new locations, and the reasons for the files' relocations. The enforce operation creates the `.__fsppadm_enforce.log` file if the file does not exist. The enforce operation appends the file if the file already exists. The `.__fsppadm_enforce.log` file can be backed up or removed as with a normal file.

The following example uses the `fsppadm enforce` command to enforce the file placement policy for the file system at mount point `/mnt1`, and includes the access time, modification time, and file size of the specified paths in the report, `/tmp/report`.

To enforce a placement policy

- Enforce a placement policy to a file system:

```
# fsppadm enforce -a -r /tmp/report /mnt1
Current Current  Relocated Relocated
Class   Volume   Class     Volume    Rule    File
tier3   dstvole  tier3     dstvole   a_to_z  /mnt1/mds1/d1/file1
tier3   dstvole  tier3     dstvole   a_to_z  /mnt1/mds1/d1/file2
tier3   dstvole  tier3     dstvole   a_to_z  /mnt1/mds1/d1/d2/file3
tier3   dstvolf  tier3     dstvolf   a_to_z  /mnt1/mds1/d1/d2/file4
.
.
.
Sweep path    : /mnt1
Files moved   : 42
KB moved      : 1267


Tier Name         Size (KB)   Free Before (KB)    Free After (KB)
tier4               524288             524256             524256
tier3               524288             522968             522968
tier2               524288             524256             524256
tier1               524288             502188             501227
```

Validating a placement policy

The following example uses the `fsppadm validate` command to validate the placement policy `policy.xml` against all mounted file systems.

To validate a placement policy against all mounted file systems

- Validate the placement policy:

```
# fsppadm validate /tmp/policy.xml
```

File placement policy grammar

VxFS allocates and relocates files within a multi-volume file system based on properties in the file system metadata that pertains to the files. Placement decisions may be based on file name, directory of residence, time of last access, access frequency, file size, and ownership. An individual file system's criteria for

allocating and relocating files are expressed in the file system's file placement policy.

A VxFS file placement policy defines the desired placement of sets of files on the volumes of a VxFS multi-volume file system. A file placement policy specifies the placement classes of volumes on which files should be created, and where and under what conditions the files should be relocated to volumes in alternate placement classes or deleted. You can create file placement policy documents, which are XML text files, using either an XML or text editor, or a VxFS graphical interface wizard.

The following output shows the overall structure of a placement policy:

```
<?xml version="1.0"?>
<!DOCTYPE PLACEMENT_POLICY [
<!-- The placement policy document definition file -->

<!-- Specification for PLACEMENT_POLICY element.
     It can contain the following:
       1. 0 or 1 COMMENT element
       2. 1 or more RULE elements
  -->
<!ELEMENT PLACEMENT_POLICY (COMMENT?, RULE+)>
<!-- The attributes of PLACEMENT_POLICY element -->
<!-- XML requires all attributes must be enclosed in double quotes -->
<!ATTLIST PLACEMENT_POLICY
  Name CDATA #REQUIRED
  Version (5.0) #REQUIRED
>


<!-- Specification for COMMENT element -->
<!ELEMENT COMMENT (#PCDATA)>


<!-- Specification for RULE element.
     It can contain the following:
       1. 0 or 1 COMMENT element
       2. 1 or more SELECT elements
       3. 0 or 1 CREATE element
       4. 0 or more DELETE elements
       5. 0 or more RELOCATE elements

     The elements must appear in the above order, particularly,
     DELETE elements, if any, must preceed RELOCATE elements, if any.
```

```
     If any of the DELETE elements triggers an action, subsequent
     elements (DELETE and/or RELOCATE elements, if any) will not be
     processed.
  -->
<!ELEMENT RULE (COMMENT?, SELECT+, CREATE?, DELETE*, RELOCATE*)>
<!-- The attributes of RULE element -->
  <!-- The possible and accepted values for Flags are:
         1. data
   -->
<!-- XML requires all attributes must be enclosed in double quotes -->
<!ATTLIST RULE
  Name CDATA #REQUIRED
  Flags (data) #REQUIRED
>


<!-- Specification for SELECT element. This describes selection criteria.
     It can contain the following:
       1. 0 or 1 COMMENT elements
       2. 0 or more DIRECTORY elements
       3. 0 or more PATTERN elements
       4. 0 or more USER elements
       5. 0 or more GROUP elements

     The elements can appear in any order.
  -->
<!ELEMENT SELECT (COMMENT?, DIRECTORY*, PATTERN*, USER*, GROUP*)>
<!-- The attributes of SELECT element -->
<!-- XML requires all attributes must be enclosed in double quotes -->
<!ATTLIST SELECT
  Name CDATA #IMPLIED
>


<!-- Specification for DIRECTORY element
  The DIRECTORY element takes a path relative to the
  mount point. So if the intention is to sweep from
  /db/finance/data and /db is the mount point,
  DIRECTORY element should contain finance/data

  Only one value can be specified per element.
  -->
<!ELEMENT DIRECTORY (#PCDATA)>
```

```
<!-- The attributes of DIRECTORY element -->
  <!-- The possible and accepted values for Flags are:
         1. recursive
         2. nonrecursive

       If a given directory appears in more than one RULE,
       all such DIRECTORY elements must all be recursive or
       nonrecursive but can not be a combination. If no DIRECTORY
       element is specified, all the files under the mount point
  will be selected.
  -->
<!-- XML requires all attributes must be enclosed in double quotes -->
<!ATTLIST DIRECTORY
  Flags (recursive|nonrecursive) #REQUIRED
>


<!-- Specification for PATTERN element
  The PATTERN can be a full name of a file, i.e., can not contain
  "/" characters. Or it can have a '*' character. The first '*'
  character will be considered as wild character and any other
  character, including a second '*' are treated as literals.

  Only one value can be specified per element.
  -->
<!ELEMENT PATTERN (#PCDATA)>
<!-- The attributes of PATTERN element -->
  <!-- The possible and accepted values for Flags are
         1. recursive

       This is an optional attribute. It is meaningful only
       if the PATTERN is a dirctory. Default is nonrecursive,
       which will be case for file PATTERNs. If this attribute
       is specified, the enclosing SELECTion criteria will
       select all files in any component directory (for example
       dir1, in /mnt/dir0/dir1 if PATTERN is dir1) that is
       anywhere (below the DIRECTORY,
         - if it is specified and has 'recursive flag or
         - anywhere in file system, if DIRECTORY is not
           specified),
       provided the component directory  matches the PATTERN
       (here 'dir1' in the example). If PATTERN has wild
       character '*' in it, wild char based matching is performed.
```

```
  -->
<!-- XML requires all attributes must be enclosed in double quotes -->
<!ATTLIST PATTERN
  Flags (recursive | nonrecursive) "nonrecursive"
>


<!-- Specification for USER element
  The USER is a name string of the unix domain user

  Only one value can be specified per element.
  -->
<!ELEMENT USER (#PCDATA)>


<!-- Specification for GROUP element
  The GROUP is a name string of the unix domain group

  Only one value can be specified per element.
  -->
<!ELEMENT GROUP (#PCDATA)>


<!-- Specification for CREATE element. This describes creation criteria.
     It can contain the following:
       1. 0 or 1 COMMENT element
       2. 1 ON element

     The order of elements may be significant in future
  -->
<!ELEMENT CREATE (COMMENT?, ON)>
<!-- The attributes of CREATE element -->
<!-- XML requires all attributes must be enclosed in double quotes -->
<!ATTLIST CREATE
  Name CDATA #IMPLIED
  Flags CDATA #IMPLIED
>


<!-- Specification for ON element. This describes location criteria.
     It can contain the following:
       1. 0 or more DESTINATION elements
```

```
      Though zero DESTINATION elements is defined in grammar, current
      implementation requires at least on DESTINATION.
  -->
<!ELEMENT ON (DESTINATION*)>
<!-- The attributes of ON element -->
  <!-- The possible and accepted values for Flags is:
         1. any

       If this attribute is set, there may or may not be any CLASS
       elements in the DESTINATION elements under the ON element.
       If any of the DESTINATION elements have CLASS element, such
       CLASSes in the file system would be used first before other
       placement class storage is used.
  -->
<!-- XML requires all attributes must be enclosed in double quotes -->
<!ATTLIST ON
  Name CDATA #IMPLIED
  Flags (any) #IMPLIED
>


<!-- Specification for DESTINATION element. This describes target location.
     It can contain the following:
       1. 0 or 1 CLASS element
       2. 0 or 1 PERCENT element
       3. 0 or 1 BALANCE_SIZE element
  -->
<!ELEMENT DESTINATION (CLASS?, PERCENT?, BALANCE_SIZE?)>
<!-- The attributes of DESTINATION element -->
  <!-- The possible and accepted values for Flags:
       (THIS IS NOT IMPLEMENTED)

         1. disallow

       If this 'disallow' is set, there must not be any PERCENT or
       BALANCE_SIZE elements in such DESTINATION element but there
       must be a CLASS element. There must not be any RELOCATE and
       DELETE statements in the enclosing RULE element either.
  -->
<!-- XML requires all attributes must be enclosed in double quotes -->
<!ATTLIST DESTINATION
  Name CDATA #IMPLIED
  Flags (disallow) #IMPLIED
```

```
>


<!-- Specification for CLASS element
  The file system resides on a multi-component volume set.
  Each volume in the volume set will be in what is called a
  placement class. The placement classes are implemented as tags
  on the volumes. These tags are organized into a hierarchy prefix.
  The placement policy uses the vxfs.placement_class. prefix.
  The CLASS element specifies the placement class of the
  underlying storage, without the prefix. For example, if a
  volume has a placement class of vxfs.placment_class.gold
  then gold would be the value of CLASS element.
  -->
<!ELEMENT CLASS (#PCDATA)>


<!-- Specification for PERCENT element
  (THIS IS NOT IMPLEMENTED)

  If the PERCENT element is in DESTINATION element, it determines
  how much of its CLASS can be filled up with the files selected
  by a given RULE.

  If the PERCENT element is in SOURCE element, it determines
  how much of its CLASS can be emptied when the files are relocated
  our deleted from it.
  -->
<!ELEMENT PERCENT (#PCDATA)>


<!-- Specification for BALANCE_SIZE element
  Multiple volumes may have the same placement class and there can
  be multiple DESTINATIONs (hence CLASSes) in a given ON (and TO)
  element. If a BALANCE_SIZE is specified for a given CLASS,
  the usage of volumes of that given placement class will be used
  evenly by allocating BALANCE_SIZE amount of space for each
  volume for each allocation.
  -->
<!ELEMENT BALANCE_SIZE (#PCDATA)>
<!-- The attributes of BALANCE_SIZE element -->
  <!-- The possible and accepted values for Units are:
         1. bytes
```

```
        2. KB
        3. MB
        4. GB
  -->
<!-- XML requires all attributes must be enclosed in double quotes -->
<!ATTLIST BALANCE_SIZE
  Units (bytes|KB|MB|GB) #REQUIRED
>

<!-- Specification for DELETE element. This describes deletion criteria.
     It can contain the following:
       1. 0 or 1 COMMENT element
       2. 0 or 1 FROM element
       3. 0 or 1 WHEN element
  -->
<!ELEMENT DELETE (COMMENT?, FROM?, WHEN?)>
<!-- The attributes of DELETE element -->
<!-- XML requires all attributes must be enclosed in double quotes -->
<!ATTLIST DELETE
  Name CDATA #IMPLIED
  Flags (none) #IMPLIED
>

<!-- Specification for RELOCATE element. This describes relocation criteria.
     It can contain the following:
       1. 0 or 1 COMMENT element
       2. 0 or 1 FROM element
       3. 1 TO element
       4. 0 or 1 WHEN element

     The order of TO elements is significant. Earlier CLASSes would be
     used before the latter ones.
  -->
<!ELEMENT RELOCATE (COMMENT?, FROM?, TO, WHEN?)>
<!-- The attributes of RELOCATE element -->
<!-- XML requires all attributes must be enclosed in double quotes -->
<!ATTLIST RELOCATE
  Name CDATA #IMPLIED
  Flags (none) #IMPLIED
>


<!-- Specification for FROM element. This describes source criteria.
```

```
     It can contain the following:
       1. 1 or more SOURCE elements
  -->
<!ELEMENT FROM (SOURCE+)>
<!-- The attributes of FROM element -->
<!-- XML requires all attributes must be enclosed in double quotes -->
<!ATTLIST FROM
  Name CDATA #IMPLIED
  Flags (none) #IMPLIED
>


<!-- Specification for SOURCE element. This describes source location.
     It can contain the following:
       1. 1 CLASS element
       2. 0 or 1 PERCENT element
  -->
<!ELEMENT SOURCE (CLASS, PERCENT?)>
<!-- The attributes of SOURCE element -->
<!-- XML requires all attributes must be enclosed in double quotes -->
<!ATTLIST SOURCE
  Name CDATA #IMPLIED
  Flags (none) #IMPLIED
>


<!-- Specification for TO element. This describes destination criteria.
     It can contain the following:
       1. 1 or more DESTINATION elements
  -->
<!ELEMENT TO (DESTINATION+)>
<!-- The attributes of TO element -->
<!-- XML requires all attributes must be enclosed in double quotes -->
<!ATTLIST TO
  Name CDATA #IMPLIED
  Flags (none) #IMPLIED
>


<!-- Specification for WHEN element. This describes relocation specifiers.
     It can contain the following
       1.  0 or 1 SIZE element
       2.  0 or 1 ACCAGE element
```

```
        3.  0 or 1 MODAGE element
        4.  0 or 1 IOTEMP element
        5.  0 or 1 ACCESSTEMP element

     The order of elements is significant.
  -->
<!ELEMENT WHEN (SIZE?, ACCAGE?, MODAGE?, IOTEMP?, ACCESSTEMP?)>
<!-- The attributes of WHEN element -->
<!-- XML requires all attributes must be enclosed in double quotes -->
<!ATTLIST WHEN
  Name CDATA #IMPLIED
  Flags (none) #IMPLIED
>


<!-- Specification for SIZE element
     It can contain the following:
       1.  0 or 1 MIN element
       2.  0 or 1 MAX element
  -->
<!ELEMENT SIZE (MIN?, MAX?)>
<!-- The attributes of SIZE element -->
  <!-- The possible and accepted values for Prefer are:
       (THIS IS NOT IMPLEMENTED)

         1. low
         2. high

       The possible and accepted values for Units are:
         1. bytes
         2. KB
         3. MB
         4. GB
  -->
<!-- XML requires all attributes must be enclosed in double quotes -->
<!ATTLIST SIZE
  Prefer (low|high) #IMPLIED
  Units (bytes|KB|MB|GB) #REQUIRED
>


<!-- Specification for ACCAGE element
     It can contain the following
```

```
        1.  0 or 1 MIN element
        2.  0 or 1 MAX element
  -->
<!ELEMENT ACCAGE (MIN?, MAX?)>
<!-- The attributes of ACCAGE element -->
  <!-- The possible and accepted values for Prefer are:
        (THIS IS NOT IMPLEMENTED)

          1. low
          2. high

        The possible and accepted values for Units are:
          1. hours
          2. days
  -->
<!-- XML requires all attributes must be enclosed in double quotes -->
<!ATTLIST ACCAGE
  Prefer (low|high) #IMPLIED
  Units (hours|days) #REQUIRED
>


<!-- Specification for MODAGE element
     It can contain the following:
       1.  0 or 1 MIN element
       2.  0 or 1 MAX element
  -->
<!ELEMENT MODAGE (MIN?, MAX?)>
<!-- The attributes of MODAGE element -->
  <!-- The possible and accepted values for Prefer are:
        (THIS IS NOT IMPLEMENTED)

          1. low
          2. high

        The possible and accepted values for Units are:
          1. hours
          2. days
  -->
<!-- XML requires all attributes must be enclosed in double quotes -->
<!ATTLIST MODAGE
  Prefer (low|high) #IMPLIED
  Units (hours|days) #REQUIRED
```

```
>


<!-- Specification for IOTEMP element
     The value of IOTEMP represents bytes read (nrbytes),
     bytes written (nwbytes) or bytes transferred, i.e.,
     read and written (nrwbytes), divided by the size of the
     file, over a specified PERIOD (in days).

     It can contain the following:
       1. 0 or 1 MIN element
       2. 0 or 1 MAX element
       3. 1 PERIOD element
  -->
<!ELEMENT IOTEMP (MIN?, MAX?, PERIOD)>
<!-- The attributes of IOTEMP element -->
  <!-- The possible and accepted values for Prefer are:
       (THIS IS NOT IMPLEMENTED)

         1. low
         2. high
  -->
  <!-- The possible and accepted values for Type are:
         1. nrbytes
         2. nwbytes
         3. nrwbytes
  -->
<!-- XML requires all attributes must be enclosed in double quotes -->
<!ATTLIST IOTEMP
  Prefer (low|high) #IMPLIED
  Type (nrbytes|nwbytes|nrwbytes) #REQUIRED
>


<!-- Specification for ACCESSTEMP element
     The value of ACCESSTEMP represents times read (nrbytes),
     times written (nwbytes) or times access i.e.,
     read and written (nrws) over a specified PERIOD (in days).

     It can contain the following:
       1. 0 or 1 MIN element
       2. 0 or 1 MAX element
       3. 1 PERIOD element
```

```
  -->
<!ELEMENT ACCESSTEMP (MIN?, MAX?, PERIOD)>
<!-- The attributes of ACCESSTEMP element -->
  <!-- The possible and accepted values for Prefer are:
       (THIS IS NOT IMPLEMENTED)

         1. low
         2. high
  -->
  <!-- The possible and accepted values for Type are:
         1. nreads
         2. nwrites
         3. nrws
  -->
<!-- XML requires all attributes must be enclosed in double quotes -->
<!ATTLIST ACCESSTEMP
  Prefer (low|high) #IMPLIED
  Type (nreads|nwrites|nrws) #REQUIRED
>


<!-- Specification for MIN element -->
<!ELEMENT MIN (#PCDATA)>
<!-- The attributes of MIN element -->
  <!-- The possible and accepted values for Flags are:
         1. gt for greater than
         2. eq for equal to
         3. gteq for greater than or equal to
  -->
<!-- XML requires all attributes must be enclosed in double quotes -->
<!ATTLIST MIN
  Flags (gt|eq|gteq) #REQUIRED
>


<!-- Specification for MAX element -->
<!ELEMENT MAX (#PCDATA)>
<!-- The attributes of MAX element -->
  <!-- The possible and accepted values for Flags are:
         1. lt for less than
         2. lteq for less than or equal to
  -->
<!-- XML requires all attributes must be enclosed in double quotes -->
```

```
<!ATTLIST MAX
  Flags (lt|lteq) #REQUIRED
>


<!-- Specification for PERIOD element -->
<!ELEMENT PERIOD (#PCDATA)>
<!-- The attributes of PERIOD element -->
<!-- XML requires all attributes must be enclosed in double quotes -->
<!ATTLIST PERIOD
  Units (days) #REQUIRED
>
]>
```

File placement policy rules

A VxFS file placement policy consists of one or more rules. Each rule applies to one or more files. The files to which a rule applies are designated in one or more SELECT statements. A SELECT statement designates files according to one or more of four properties: their names or naming patterns, the directories in which they reside, their owners' user names, and their owners' group names.

A file may be designated by more than one rule. For example, if one rule designates files in directory `/dir`, and another designates files owned by user1, a file in `/dir` that is owned by user1 is designated by both rules. Only the rule that appears first in the placement policy applies to the file; subsequent rules are ignored.

You can define placement policies that do not encompass the entire file system name space. When a file that is not designated by any rule in its file system's active placement policy is created, VxFS places the file according to its own internal algorithms. To maintain full control over file placement, include a catchall rule at the end of each placement policy document with a SELECT statement that designates files by the naming pattern *. Such a rule designates all files that have not been designated by the rules appearing earlier in the placement policy document.

SELECT statement

The VxFS placement policy rule SELECT statement designates the collection of files to which a rule applies.

The following XML snippet illustrates the general form of the SELECT statement:

```
<SELECT>
  <DIRECTORY Flags="directory_flag_value">...value...
  </DIRECTORY>
  <PATTERN Flags="pattern_flag_value">...value...</PATTERN>
  <USER>...value...</USER>
  <GROUP>...value...</GROUP>
</SELECT>
```

A SELECT statement may designate files by using the following selection criteria:

<DIRECTORY> A full path name relative to the file system mount point. The `Flags="directory_flag_value"` XML attribute must have a value of `nonrecursive`, denoting that only files in the specified directory are designated, or a value of `recursive`, denoting that files in all subdirectories of the specified directory are designated. The `Flags` attribute is mandatory.

The <DIRECTORY> criterion is optional, and may be specified more than once.

<PATTERN>

Either an exact file name or a pattern using a single wildcard character (*). For example, the pattern “abc*” denotes all files whose names begin with “abc”. The pattern “abc.*” denotes all files whose names are exactly "abc" followed by a period and any extension. The pattern “*abc” denotes all files whose names end in “abc”, even if the name is all or part of an extension. The pattern “*.abc” denotes files of any name whose name extension (following the period) is “abc”. The pattern “ab*c” denotes all files whose names start with “ab” and end with “c”. The first "*" character is treated as a wildcard, while any subsequent "*" characters are treated as literal text. The pattern cannot contain "/".

The wildcard character matches any character, including ".", "?", and "[", unlike using the wildcard in a shell.

The `Flags=”pattern_flag_value”` XML attribute is optional, and if specified can only have a value of `recursive`. Specify `Flags="recursive"` only if the pattern is a directory. If `Flags` is not specified, the default attribute value is nonrecursive. If `Flags="recursive"` is specified, the enclosing selection criteria selects all files in any component directory that is anywhere below the directory specified by <DIRECTORY> if the component directory matches the pattern and either of the following is true:

- <DIRECTORY> is specified and has the recursive flag.
- <DIRECTORY> is not specified and the directory is anywhere in the file system.

If the pattern contains the wildcard character (*), wildcard character matching is performed.

The <PATTERN> criterion is optional, and may be specified more than once. Only one value can be specified per <PATTERN> element.

<USER>

User name of the file's owner. The user number cannot be specified in place of the name.

The <USER> criterion is optional, and may be specified more than once.

<GROUP>

Group name of the file's owner. The group number cannot be specified in place of the group name.

The <GROUP> criterion is optional, and may be specified more than once.

One or more instances of any or all of the file selection criteria may be specified within a single SELECT statement. If two or more selection criteria of different types are specified in a single statement, a file must satisfy one criterion of each type to be selected.

In the following example, only files that reside in either the ora/db or the crash/dump directory, and whose owner is either user1 or user2 are selected for possible action:

```
<SELECT>
  <DIRECTORY Flags="nonrecursive">ora/db</DIRECTORY>
  <DIRECTORY Flags="nonrecursive">crash/dump</DIRECTORY>
  <USER>user1</USER>
  <USER>user2</USER>
</SELECT>
```

A rule may include multiple SELECT statements. If a file satisfies the selection criteria of one of the SELECT statements, it is eligible for action.

In the following example, any files owned by either user1 or user2, no matter in which directories they reside, as well as all files in the `ora/db` or `crash/dump` directories, no matter which users own them, are eligible for action:

```
<SELECT>
  <DIRECTORY Flags="nonrecursive">ora/db</DIRECTORY>
  <DIRECTORY Flags="nonrecursive">crash/dump</DIRECTORY>
</SELECT>
<SELECT>
  <USER>user1</USER>
  <USER>user2</USER>
</SELECT>
```

When VxFS creates new files, VxFS applies active placement policy rules in the order of appearance in the active placement policy's XML source file. The first rule in which a SELECT statement designates the file to be created determines the file's placement; no later rules apply. Similarly, VxFS scans the active policy rules on behalf of each file when relocating files, stopping the rules scan when it reaches the first rule containing a SELECT statement that designates the file. This behavior holds true even if the applicable rule results in no action. Take for example a policy rule that indicates that `.dat` files inactive for 30 days should be relocated, and a later rule indicates that `.dat` files larger than 10 megabytes should be relocated. A 20 megabyte .dat file that has been inactive for 10 days will not be relocated because the earlier rule applied. The later rule is never scanned.

A placement policy rule's action statements apply to all files designated by any of the rule's SELECT statements. If an existing file is not designated by a SELECT statement in any rule of a file system's active placement policy, then DST does not relocate or delete the file. If an application creates a file that is not designated by a SELECT statement in a rule of the file system's active policy, then VxFS places the file according to its own internal algorithms. If this behavior is inappropriate,

the last rule in the policy document on which the file system's active placement policy is based should specify <PATTERN>*</PATTERN> as the only selection criterion in its SELECT statement, and a CREATE statement naming the desired placement class for files not selected by other rules.

CREATE statement

A CREATE statement in a file placement policy rule specifies one or more placement classes of volumes on which VxFS should allocate space for new files to which the rule applies at the time the files are created. You can specify only placement classes, not individual volume names, in a CREATE statement.

A file placement policy rule may contain at most one CREATE statement. If a rule does not contain a CREATE statement, VxFS places files designated by the rule's SELECT statements according to its internal algorithms. However, rules without CREATE statements can be used to relocate or delete existing files that the rules' SELECT statements designate.

The following XML snippet illustrates the general form of the CREATE statement:

```
<CREATE>
  <ON Flags="...flag_value...">
    <DESTINATION>
      <CLASS>...placement_class_name...</CLASS>
      <BALANCE_SIZE Units="units_specifier">...chunk_size...
      </BALANCE_SIZE>
    </DESTINATION>
    <DESTINATION>...additional placement class specifications...
    </DESTINATION>
  </ON>
</CREATE>
```

A CREATE statement includes a single <ON> clause, in which one or more <DESTINATION> XML elements specify placement classes for initial file allocation in order of decreasing preference. VxFS allocates space for new files to which a rule applies on a volume in the first class specified, if available space permits. If space cannot be allocated on any volume in the first class, VxFS allocates space on a volume in the second class specified if available space permits, and so forth.

If space cannot be allocated on any volume in any of the placement classes specified, file creation fails with an ENOSPC error, even if adequate space is available elsewhere in the file system's volume set. This situation can be circumvented by specifying a `Flags` attribute with a value of "`any`" in the <ON> clause. If <ON Flags="any"> is specified in a CREATE statement, VxFS first attempts to allocate space for new files to which the rule applies on the specified

placement classes. Failing that, VxFS resorts to its internal space allocation algorithms, so file allocation does not fail unless there is no available space any-where in the file system's volume set.

The `Flags="any"` attribute differs from the catchall rule in that this attribute applies only to files designated by the SELECT statement in the rule, which may be less inclusive than the <PATTERN>*</PATTERN> file selection specification of the catchall rule.

In addition to the placement class name specified in the <CLASS> sub-element, a <DESTINATION> XML element may contain a <BALANCE_SIZE> sub-element. Presence of a <BALANCE_SIZE> element indicates that space allocation should be distributed across the volumes of the placement class in chunks of the indicated size. For example, if a balance size of one megabyte is specified for a placement class containing three volumes, VxFS allocates the first megabyte of space for a new or extending file on the first (lowest indexed) volume in the class, the second megabyte on the second volume, the third megabyte on the third volume, the fourth megabyte on the first volume, and so forth. Using the `Units` attribute in the <BALANCE_SIZE> XML tag, the balance size value may be specified in the following units:

bytes	Bytes
KB	Kilobytes
MB	Megabytes
GB	Gigabytes

The <BALANCE_SIZE> element distributes the allocation of database files across the volumes in a placement class. In principle, distributing the data in each file across multiple volumes distributes the I/O load across the volumes as well.

The CREATE statement in the following example specifies that files to which the rule applies should be created on the tier1 volume if space is available, and on one of the tier2 volumes if not. If space allocation on tier1 and tier2 volumes is not possible, file creation fails, even if space is available on tier3 volumes.

```
<CREATE>
  <ON>
    <DESTINATION>
      <CLASS>tier1</CLASS>
    </DESTINATION>
    <DESTINATION>
      <CLASS>tier2</CLASS>
      <BALANCE_SIZE Units="MB">1</BALANCE_SIZE>
```

```
    </DESTINATION>
  </ON>
</CREATE>
```

The <BALANCE_SIZE> element with a value of one megabyte is specified for allocations on tier2 volumes. For files allocated on tier2 volumes, the first megabyte would be allocated on the first volume, the second on the second volume, and so forth.

RELOCATE statement

The RELOCATE action statement of file placement policy rules specifies an action that VxFS takes on designated files during periodic scans of the file system, and the circumstances under which the actions should be taken. The `fsppadm enforce` command is used to scan all or part of a file system for files that should be relocated based on rules in the active placement policy at the time of the scan.

See the fsppadm(1M) manual page.

The `fsppadm enforce` scans file systems in path name order. For each file, VxFS identifies the first applicable rule in the active placement policy, as determined by the rules' SELECT statements. If the file resides on a volume specified in the <FROM> clause of one of the rule's RELOCATE statements, and if the file meets the criteria for relocation specified in the statement's <WHEN> clause, the file is scheduled for relocation to a volume in the first placement class listed in the <TO> clause that has space available for the file. The scan that results from issuing the `fsppadm enforce` command runs to completion before any files are relocated.

The following XML snippet illustrates the general form of the RELOCATE statement:

```
<RELOCATE>
  <FROM>
    <SOURCE>
      <CLASS>...placement_class_name...</CLASS>
    </SOURCE>
    <SOURCE>...additional placement class specifications...
    </SOURCE>
  </FROM>
  <TO>
    <DESTINATION>
      <CLASS>...placement_class_name...</CLASS>
      <BALANCE_SIZE Units="units_specifier">
        ...chunk_size...
      </BALANCE_SIZE>
```

```
    </DESTINATION>
    <DESTINATION>
      ...additional placement class specifications...
    </DESTINATION>
  </TO>
  <WHEN>...relocation conditions...</WHEN>
</RELOCATE>
```

A RELOCATE statement contains the following clauses:

<FROM> An optional clause that contains a list of placement classes from whose volumes designated files should be relocated if the files meet the conditions specified in the <WHEN> clause. No priority is associated with the ordering of placement classes listed in a <FROM> clause. If a file to which the rule applies is located on a volume in any specified placement class, the file is considered for relocation.

If a RELOCATE statement contains a <FROM> clause, VxFS only considers files that reside on volumes in placement classes specified in the clause for relocation. If no <FROM> clause is present, qualifying files are relocated regardless of where the files reside.

<TO>

Indicates the placement classes to which qualifying files should be relocated. Unlike the source placement class list in a FROM clause, placement classes in a <TO> clause are specified in priority order. Files are relocated to volumes in the first specified placement class if possible, to the second if not, and so forth.

The <TO> clause of the RELOCATE statement contains a list of <DESTINATION> XML elements specifying placement classes to whose volumes VxFS relocates qualifying files. Placement classes are specified in priority order. VxFS relocates qualifying files to volumes in the first placement class specified as long as space is available. A <DESTINATION> element may contain an optional <BALANCE_SIZE> modifier sub-element. The <BALANCE_SIZE> modifier indicates that relocated files should be distributed across the volumes of the destination placement class in chunks of the indicated size. For example, if a balance size of one megabyte is specified for a placement class containing three volumes, VxFS relocates the first megabyte the file to the first (lowest indexed) volume in the class, the second megabyte to the second volume, the third megabyte to the third volume, the fourth megabyte to the first volume, and so forth. Using the Units attribute in the <BALANCE_SIZE> XML tag, the chunk value may be specified in the balance size value may be specified in bytes (Units="bytes"), kilobytes (Units="KB"), megabytes (Units="MB"), or gigabytes (Units="GB").

The <BALANCE_SIZE> element distributes the allocation of database files across the volumes in a placement class. In principle, distributing the data in each file across multiple volumes distributes the I/O load across the volumes as well.

<WHEN>

An optional clause that indicates the conditions under which files to which the rule applies should be relocated. Files that have been unaccessed or unmodified for a specified period, reached a certain size, or reached a specific I/O temperature or access temperature level may be relocated. If a RELOCATE statement does not contain a <WHEN> clause, files to which the rule applies are relocated unconditionally.

A <WHEN> clause may be included in a RELOCATE statement to specify that files should be relocated only if any or all of four types of criteria are met. Files can be specified for relocation if they satisfy one or more criteria.

The following are the criteria that can be specified for the <WHEN> clause:

<ACCAGE>

This criterion is met when files are inactive for a designated period or during a designated period relative to the time at which the `fsppadm enforce` command was issued.

<MODAGE> This criterion is met when files are unmodified for a designated period or during a designated period relative to the time at which the `fsppadm enforce` command was issued.

<SIZE> This criterion is met when files exceed or drop below a designated size or fall within a designated size range.

<IOTEMP> This criterion is met when files exceed or drop below a designated I/O temperature, or fall with in a designated I/O temperature range. A file's I/O temperature is a measure of the I/O activity against it during the period designated by the <PERIOD> element prior to the time at which the `fsppadm enforce` command was issued.

See "Calculating I/O temperature and access temperature" on page 174.

<ACCESSTEMP> This criterion is met when files exceed or drop below a specified average access temperature, or fall within a specified access temperature range. A file's access temperature is similar to its I/O temperature, except that access temperature is computed using the number of I/O requests to the file, rather than the number of bytes transferred.

The following XML snippet illustrates the general form of the <WHEN> clause in a RELOCATE statement:

```
<WHEN>
  <ACCAGE Units="...units_value...">
    <MIN Flags="...comparison_operator...">
      ...min_access_age...</MIN>
    <MAX Flags="...comparison_operator...">
      ...max_access_age...</MAX>
  </ACCAGE>
  <MODAGE Units="...units_value...">
    <MIN Flags="...comparison_operator...">
      ...min_modification_age...</MIN>
    <MAX Flags="...comparison_operator...">
      ...max_modification_age...</MAX>
  </MODAGE>
  <SIZE " Units="...units_value...">
    <MIN Flags="...comparison_operator...">
      ...min_size...</MIN>
    <MAX Flags="...comparison_operator...">
      ...max_size...</MAX>
  </SIZE>
  <IOTEMP Type="...read_write_preference...">
```

```
      <MIN Flags="...comparison_operator...">
        ...min_I/O_temperature...</MIN>
      <MAX Flags="...comparison_operator...">
        ...max_I/O_temperature...</MAX>
      <PERIOD>...days_of_interest...</PERIOD>
    </IOTEMP>
    <ACCESSTEMP Type="...read_write_preference...">
      <MIN Flags="...comparison_operator...">
        ...min_access_temperature...</MIN>
      <MAX Flags="...comparison_operator...">
        ...max_access_temperature...</MAX>
      <PERIOD>...days_of_interest...</PERIOD>
    </ACCESSTEMP>
  </WHEN>
```

The access age (<ACCAGE>) element refers to the amount of time since a file was last accessed. VxFS computes access age by subtracting a file's time of last access, atime, from the time when the `fsppadm enforce` command was issued. The <MIN> and <MAX> XML elements in an <ACCAGE> clause, denote the minimum and maximum access age thresholds for relocation, respectively. These elements are optional, but at least one must be included. Using the `Units` XML attribute, the <MIN> and <MAX> elements may be specified in the following units:

hours	Hours
days	Days. A day is considered to be 24 hours prior to the time that the `fsppadm enforce` command was issued.

Both the <MIN> and <MAX> elements require `Flags` attributes to direct their operation.

For <MIN>, the following `Flags` attributes values may be specified:

gt	The time of last access must be greater than the specified interval.
eq	The time of last access must be equal to the specified interval.
gteq	The time of last access must be greater than or equal to the specified interval.

For <MAX>, the following Flags attributes values may be specified.

lt	The time of last access must be less than the specified interval.

lteq	The time of last access must be less than or equal to the specified interval.

Including a <MIN> element in a <WHEN> clause causes VxFS to relocate files to which the rule applies that have been inactive for longer than the specified interval. Such a rule would typically be used to relocate inactive files to less expensive storage tiers. Conversely, including <MAX> causes files accessed within the specified interval to be relocated. It would typically be used to move inactive files against which activity had recommenced to higher performance or more reliable storage. Including both <MIN> and <MAX> causes VxFS to relocate files whose access age lies between the two.

The modification age relocation criterion, <MODAGE>, is similar to access age, except that files' POSIX mtime values are used in computations. You would typically specify the <MODAGE> criterion to cause relocation of recently modified files to higher performance or more reliable storage tiers in anticipation that the files would be accessed recurrently in the near future.

The file size relocation criterion, <SIZE>, causes files to be relocated if the files are larger or smaller than the values specified in the <MIN> and <MAX> relocation criteria, respectively, at the time that the `fsppadm enforce` command was issued. Specifying both criteria causes VxFS to schedule relocation for files whose sizes lie between the two. Using the Units attribute, threshold file sizes may be specified in the following units:

bytes	Bytes
KB	Kilobytes
MB	Megabytes
GB	Gigabytes

Specifying the I/O temperature relocation criterion

The I/O temperature relocation criterion, <IOTEMP>, causes files to be relocated if their I/O temperatures rise above or drop below specified values over a specified period immediately prior to the time at which the `fsppadm enforce` command was issued. A file's I/O temperature is a measure of the read, write, or total I/O activity against it normalized to the file's size. Higher I/O temperatures indicate higher levels of application activity; lower temperatures indicate lower levels. VxFS computes a file's I/O temperature by dividing the number of bytes transferred to or from it (read, written, or both) during the specified period by its size at the time that the `fsppadm enforce` command was issued.

See "Calculating I/O temperature and access temperature" on page 174.

As with the other file relocation criteria, <IOTEMP> may be specified with a lower threshold by using the <MIN> element, an upper threshold by using the <MAX> element, or as a range by using both. However, I/O temperature is dimensionless and therefore has no specification for units.

VxFS computes files' I/O temperatures over the period between the time when the `fsppadm enforce` command was issued and the number of days in the past specified in the <PERIOD> element, where a day is a 24 hour period. For example, if the `fsppadm enforce` command was issued at 2 PM on Wednesday, and a <PERIOD> value of 2 was specified, VxFS looks at file I/O activity for the period between 2 PM on Monday and 2 PM on Wednesday. The number of days specified in the <PERIOD> element should not exceed one or two weeks due to the disk space used by the File Change Log (FCL) file.

See "About the File Change Log file" on page 110.

I/O temperature is a softer measure of I/O activity than access age. With access age, a single access to a file resets the file's atime to the current time. In contrast, a file's I/O temperature decreases gradually as time passes without the file being accessed, and increases gradually as the file is accessed periodically. For example, if a new 10 megabyte file is read completely five times on Monday and `fsppadm enforce` runs at midnight, the file's two-day I/O temperature will be five and its access age in days will be zero. If the file is read once on Tuesday, the file's access age in days at midnight will be zero, and its two-day I/O temperature will have dropped to three. If the file is read once on Wednesday, the file's access age at midnight will still be zero, but its two-day I/O temperature will have dropped to one, as the influence of Monday's I/O will have disappeared.

If the intention of a file placement policy is to keep files in place, such as on top-tier storage devices, as long as the files are being accessed at all, then access age is the more appropriate relocation criterion. However, if the intention is to relocate files as the I/O load on them decreases, then I/O temperature is more appropriate.

The case for upward relocation is similar. If files that have been relocated to lower-tier storage devices due to infrequent access experience renewed application activity, then it may be appropriate to relocate those files to top-tier devices. A policy rule that uses access age with a low <MAX> value, that is, the interval between `fsppadm enforce` runs, as a relocation criterion will cause files to be relocated that have been accessed even once during the interval. Conversely, a policy that uses I/O temperature with a <MIN> value will only relocate files that have experienced a sustained level of activity over the period of interest.

RELOCATE statement examples

The following example illustrates an unconditional relocation statement, which is the simplest form of the RELOCATE policy rule statement:

```
<RELOCATE>
  <FROM>
    <SOURCE>
      <CLASS>tier1</CLASS>
    </SOURCE>
  </FROM>
  <TO>
    <DESTINATION>
      <CLASS>tier2</CLASS>
    </DESTINATION>
  </TO>
</RELOCATE>
```

The files designated by the rule's SELECT statement that reside on volumes in placement class tier1 at the time the `fsppadm enforce` command executes would be unconditionally relocated to volumes in placement class tier2 as long as space permitted. This type of rule might be used, for example, with applications that create and access new files but seldom access existing files once they have been processed. A CREATE statement would specify creation on tier1 volumes, which are presumably high performance or high availability, or both. Each instantiation of `fsppadm enforce` would relocate files created since the last run to tier2 volumes.

The following example illustrates a more comprehensive form of the RELOCATE statement that uses access age as the criterion for relocating files from tier1 volumes to tier2 volumes. This rule is designed to maintain free space on tier1 volumes by relocating inactive files to tier2 volumes:

```
<RELOCATE>
  <FROM>
    <SOURCE>
      <CLASS>tier1</CLASS>
    </SOURCE>
  </FROM>
  <TO>
    <DESTINATION>
      <CLASS>tier2</CLASS>
     </DESTINATION>
  </TO>
  <WHEN>
    <SIZE Units="MB">
      <MIN Flags="gt">1</MIN>
      <MAX Flags="lt">1000</MAX>
    </SIZE>
    <ACCAGE Units="days">
```

```
      <MIN Flags="gt">30</MIN>
    </ACCAGE>
  </WHEN>
</RELOCATE>
```

Files designated by the rule's SELECT statement are relocated from tier1 volumes to tier2 volumes if they are between 1 MB and 1000 MB in size and have not been accessed for 30 days. VxFS relocates qualifying files in the order in which it encounters them as it scans the file system's directory tree. VxFS stops scheduling qualifying files for relocation when when it calculates that already-scheduled relocations would result in tier2 volumes being fully occupied.

The following example illustrates a possible companion rule that relocates files from tier2 volumes to tier1 ones based on their I/O temperatures. This rule might be used to return files that had been relocated to tier2 volumes due to inactivity to tier1 volumes when application activity against them increases. Using I/O temperature rather than access age as the relocation criterion reduces the chance of relocating files that are not actually being used frequently by applications. This rule does not cause files to be relocated unless there is sustained activity against them over the most recent two-day period.

```
<RELOCATE>
  <FROM>
    <SOURCE>
      <CLASS>tier2</CLASS>
    </SOURCE>
  </FROM>
  <TO>
    <DESTINATION>
      <CLASS>tier1</CLASS>
    </DESTINATION>
  </TO>
  <WHEN>
    <IOTEMP Type="nrbytes">
      <MIN Flags="gt">5</MIN>
      <PERIOD>2</PERIOD>
    </IOTEMP>
  </WHEN>
</RELOCATE>
```

This rule relocates files that reside on tier2 volumes to tier1 volumes if their I/O temperatures are above 5 for the two day period immediately preceding the issuing of the `fsppadm enforce` command. VxFS relocates qualifying files in the order in which it encounters them during its file system directory tree scan. When tier1 volumes are fully occupied, VxFS stops scheduling qualifying files for relocation.

VxFS file placement policies are able to control file placement across any number of placement classes. The following example illustrates a rule for relocating files with low I/O temperatures from tier1 volumes to tier2 volumes, and to tier3 volumes when tier2 volumes are fully occupied:

```
<RELOCATE>
  <FROM>
    <SOURCE>
      <CLASS>tier1</CLASS>
    </SOURCE>
  </FROM>
  <TO>
    <DESTINATION>
      <CLASS>tier2</CLASS>
    </DESTINATION>
    <DESTINATION>
      <CLASS>tier3</CLASS>
    </DESTINATION>
  </TO>
  <WHEN>
    <IOTEMP Type="nrbytes">
      <MAX Flags="lt">4</MAX>
      <PERIOD>3</PERIOD>
    </IOTEMP>
  </WHEN>
</RELOCATE>
```

This rule relocates files whose 3-day I/O temperatures are less than 4 and which reside on tier1 volumes. When VxFS calculates that already-relocated files would result in tier2 volumes being fully occupied, VxFS relocates qualifying files to tier3 volumes instead. VxFS relocates qualifying files as it encounters them in its scan of the file system directory tree.

The <FROM> clause in the RELOCATE statement is optional. If the clause is not present, VxFS evaluates files designated by the rule's SELECT statement for relocation no matter which volumes they reside on when the `fsppadm enforce` command is issued. The following example illustrates a fragment of a policy rule that relocates files according to their sizes, no matter where they reside when the `fsppadm enforce` command is issued:

```
<RELOCATE>
  <TO>
    <DESTINATION>
      <CLASS>tier1</CLASS>
```

```
      </DESTINATION>
    </TO>
    <WHEN>
      <SIZE Units="MB">
        <MAX Flags="lt">10</MAX>
      </SIZE>
    </WHEN>
  </RELOCATE>
  <RELOCATE>
    <TO>
      <DESTINATION>
        <CLASS>tier2</CLASS>
      </DESTINATION>
    </TO>
    <WHEN>
      <SIZE Units="MB">
        <MIN Flags="gteq">10</MIN>
        <MAX Flags="lt">100</MAX>
      </SIZE>
    </WHEN>
  </RELOCATE>
  <RELOCATE>
    <TO>
      <DESTINATION>
        <CLASS>tier3</CLASS>
      </DESTINATION>
    </TO>
    <WHEN>
      <SIZE Units="MB">
        <MIN Flags="gteq">100</MIN>
      </SIZE>
    </WHEN>
  </RELOCATE>
```

This rule relocates files smaller than 10 megabytes to tier1 volumes, files between 10 and 100 megabytes to tier2 volumes, and files larger than 100 megabytes to tier3 volumes. VxFS relocates all qualifying files that do not already reside on volumes in their DESTINATION placement classes when the `fsppadm enforce` command is issued.

DELETE statement

The DELETE file placement policy rule statement is very similar to the RELOCATE statement in both form and function, lacking only the <TO> clause. File placement policy-based deletion may be thought of as relocation with a fixed destination.

Note: Use DELETE statements with caution.

The following XML snippet illustrates the general form of the DELETE statement:

```
<DELETE>
  <FROM>
    <SOURCE>
      <CLASS>...placement_class_name...</CLASS>
    </SOURCE>
    <SOURCE>
      ...additional placement class specifications...
    </SOURCE>
  </FROM>
  <WHEN>...relocation conditions...</WHEN>
</DELETE>
```

A DELETE statement contains the following clauses:

<FROM> An optional clause that contains a list of placement classes from whose volumes designated files should be deleted if the files meet the conditions specified in the <WHEN> clause. No priority is associated with the ordering of placement classes in a <FROM> clause. If a file to which the rule applies is located on a volume in any specified placement class, the file is deleted. If a DELETE statement does not contain a <FROM> clause, VxFS deletes qualifying files no matter on which of a file system's volumes the files reside.

<WHEN> An optional clause specifying the conditions under which files to which the rule applies should be deleted. The form of the <WHEN> clause in a DELETE statement is identical to that of the <WHEN> clause in a RELOCATE statement. If a DELETE statement does not contain a <WHEN> clause, files designated by the rule's SELECT statement, and the <FROM> clause if it is present, are deleted unconditionally.

DELETE statement examples

The following example illustrates the use of the DELETE statement:

```
<DELETE>
  <FROM>
    <SOURCE>
      <CLASS>tier3</CLASS>
    </SOURCE>
  </FROM>
</DELETE>
<DELETE>
  <FROM>
    <SOURCE>
      <CLASS>tier2</CLASS>
    </SOURCE>
  </FROM>
  <WHEN>
    <ACCAGE Units="days">
      <MIN Flags="gt">120</MIN>
    </ACCAGE>
  </WHEN>
</DELETE>
```

The first DELETE statement unconditionally deletes files designated by the rule's SELECT statement that reside on tier3 volumes when the `fsppadm enforce` command is issued. The absence of a <WHEN> clause in the DELETE statement indicates that deletion of designated files is unconditional.

The second DELETE statement deletes files to which the rule applies that reside on tier2 volumes when the fsppadm enforce command is issued and that have not been accessed for the past 120 days.

Calculating I/O temperature and access temperature

An important application of VxFS Dynamic Storage Tiering is automating the relocation of inactive files to lower cost storage. If a file has not been accessed for the period of time specified in the <ACCAGE> element, a scan of the file system should schedule the file for relocation to a lower tier of storage. But, time since last access is inadequate as the only criterion for activity-based relocation for the following reasons:

- Access age is a binary measure. The time since last access of a file is computed by subtracting the time at which the `fsppadm enforce` command is issued from the POSIX atime in the file's metadata. If a file is opened the day before the `fsppadm enforce` command, its time since last access is one day, even though it may have been inactive for the month preceding. If the intent of a policy rule is to relocate inactive files to lower tier volumes, it will perform

badly against files that happen to be accessed, however casually, within the interval defined by the value of the <ACCAGE> pa-rameter.

- Access age is a poor indicator of resumption of significant activity. Using ACCAGE, the time since last access, as a criterion for relocating inactive files to lower tier volumes may fail to schedule some relocations that should be performed, but at least this method results in less relocat--ion activity than necessary. Using ACCAGE as a criterion for relocating previously inactive files that have become active is worse, because this method is likely to schedule relocation activity that is not warranted. If a policy rule's intent is to cause files that have experienced I/O activity in the recent past to be relocated to higher performing, perhaps more failure tolerant storage, ACCAGE is too coarse a filter. For example, in a rule specifying that files on tier2 volumes that have been accessed within the last three days should be relocated to tier1 volumes, no distinction is made between a file that was browsed by a single user and a file that actually was used intensively by applications.

DST implements the concept of I/O temperature and access temperature to overcome these deficiencies. A file's I/O temperature is equal to the number of bytes transferred to or from it over a specified period of time divided by the size of the file. For example, if a file occupies one megabyte of storage at the time of an `fsppadm enforce` operation and the data in the file has been completely read or written 15 times within the last three days, VxFS calculates its 3-day average I/O temperature to be 5 (15 MB of I/O ÷ 1 MB file size ÷ 3 days).

Similarly, a file's average access temperature is the number of read or write requests made to it over a specified number of 24-hour periods divided by the number of periods. Unlike I/O temperature, access temperature is unrelated to file size. A large file to which 20 I/O requests are made over a 2-day period has the same average access temperature as a small file accessed 20 times over a 2-day period.

If a file system's active placement policy includes any <IOTEMP> or <ACCESSTEMP> clauses, VxFS begins policy enforcement by using information in the file system's FCL file to calculate average I/O activity against all files in the file system during the longest <PERIOD> specified in the policy. Shorter specified periods are ignored. VxFS uses these calculations to qualify files for I/O temperature-based relocation and deletion.

See "About the File Change Log file" on page 110.

Note: If FCL is turned off, I/O temperature-based relocation will not be accurate. When you invoke the `fsppadm enforce` command, the command displays a warning if the FCL is turned off.

As its name implies, the File Change Log records information about changes made to files in a VxFS file system. In addition to recording creations, deletions, extensions, the FCL periodically captures the cumulative amount of I/O activity (number of bytes read and written) on a file-by-file basis. File I/O activity is recorded in the FCL each time a file is opened or closed, as well as at timed intervals to capture information about files that remain open for long periods.

If a file system's active file placement policy contains <IOTEMP> clauses, execution of the `fsppadm enforce` command begins with a scan of the FCL to extract I/O activity information over the period of interest for the policy. The period of interest is the interval between the time at which the `fsppadm enforce` command was issued and that time minus the largest interval value specified in any <PERIOD> element in the active policy.

For files with I/O activity during the largest interval, VxFS computes an approximation of the amount of read, write, and total data transfer (the sum of the two) activity by subtracting the I/O levels in the oldest FCL record that pertains to the file from those in the newest. It then computes each file's I/O temperature by dividing its I/O activity by its size at Tscan. Dividing by file size is an implicit acknowledgement that relocating larger files consumes more I/O resources than relocating smaller ones. Using this algorithm requires that larger files must have more activity against them in order to reach a given I/O temperature, and thereby justify the resource cost of relocation.

While this computation is an approximation in several ways, it represents an easy to compute, and more importantly, unbiased estimate of relative recent I/O activity upon which reasonable relocation decisions can be based.

File relocation and deletion decisions can be based on read, write, or total I/O activity.

The following XML snippet illustrates the use of IOTEMP in a policy rule to specify relocation of low activity files from tier1 volumes to tier2 volumes:

```
<RELOCATE>
  <FROM>
    <SOURCE>
      <CLASS>tier1</CLASS>
    </SOURCE>
  </FROM>
  <TO>
    <DESTINATION>
      <CLASS>tier2</CLASS>
    </DESTINATION>
  </TO>
  <WHEN>
```

```
    <IOTEMP Type="nrbytes">
      <MAX Flags="lt">3</MAX>
      <PERIOD Units="days">4</PERIOD>
    </IOTEMP>
  </WHEN>
</RELOCATE>
```

This snippet specifies that files to which the rule applies should be relocated from tier1 volumes to tier2 volumes if their I/O temperatures fall below 3 over a period of 4 days. The `Type="nrbytes"` XML attribute specifies that total data transfer activity, which is the the sum of bytes read and bytes written, should be used in the computation. For example, a 50 megabyte file that experienced less than 150 megabytes of data transfer over the 4-day period immediately preceding the `fsppadm enforce` scan would be a candidate for relocation. VxFS considers files that experience no activity over the period of interest to have an I/O temperature of zero. VxFS relocates qualifying files in the order in which it encounters the files in its scan of the file system directory tree.

Using I/O temperature or access temperature rather than a binary indication of activity, such as the POSIX atime or mtime, minimizes the chance of not relocating files that were only accessed occasionally during the period of interest. A large file that has had only a few bytes transferred to or from it would have a low I/O temperature, and would therefore be a candidate for relocation to tier2 volumes, even if the activity was very recent.

But, the greater value of I/O temperature or access temperature as a file relocation criterion lies in upward relocation: detecting increasing levels of I/O activity against files that had previously been relocated to lower tiers in a storage hierarchy due to inactivity or low temperatures, and relocating them to higher tiers in the storage hierarchy.

The following XML snippet illustrates relocating files from tier2 volumes to tier1 when the activity level against them increases.

```
<RELOCATE>
  <FROM>
    <SOURCE>
      <CLASS>tier2</CLASS>
    </SOURCE>
  </FROM>
  <TO>
    <DESTINATION>
      <CLASS>tier1</CLASS>
    </DESTINATION>
  </TO>
```

```
    <WHEN>
      <IOTEMP Type="nrbytes">
        <MAX Flags="gt">5</MAX>
        <PERIOD Units="days">2</PERIOD>
      </IOTEMP>
    </WHEN>
  </RELOCATE>
```

The <RELOCATE> statement specifies that files on tier2 volumes whose I/O temperature as calculated using the number of bytes read is above 5 over a 2-day period are to be relocated to tier1 volumes. Bytes written to the file during the period of interest are not part of this calculation.

Using I/O temperature rather than a binary indicator of activity as a criterion for file relocation gives administrators a granular level of control over automated file relocation that can be used to attune policies to application requirements. For example, specifying a large value in the <PERIOD> element of an upward relocation statement prevents files from being relocated unless I/O activity against them is sustained. Alternatively, specifying a high temperature and a short period tends to relocate files based on short-term intensity of I/O activity against them.

I/O temperature and access temperature utilize the sqlite3 database for building a temporary table indexed on an inode. This temporary table is used to filter files based on I/O temperature and access temperature. The temporary table is stored in the database file `.__fsppadm_fcliotemp.db`, which resides in the `lost+found` directory of the mount point.

Multiple criteria in file placement policy rule statements

In certain cases, file placement policy rule statements may contain multiple clauses that affect their behavior. In general, when a rule statement contains multiple clauses of a given type, all clauses must be satisfied in order for the statement to be effective. There are four cases of note in which multiple clauses may be used.

Multiple file selection criteria in SELECT statement clauses

Within a single SELECT statement, all the selection criteria clauses of a single type are treated as a selection list. A file need only satisfy a single criterion of a given type to be designated.

In the following example, files in any of the `db/datafiles`, `db/indexes`, and `db/logs` directories, all relative to the file system mount point, would be selected:

```
<SELECT>
  <DIRECTORY Flags="nonrecursive">db/datafiles</DIRECTORY>
  <DIRECTORY Flags="nonrecursive">db/indexes</DIRECTORY>
  <DIRECTORY Flags="nonrecursive">db/logs</DIRECTORY>
</SELECT>
```

This example is in direct contrast to the treatment of selection criteria clauses of different types. When a SELECT statement includes multiple types of file selection criteria, a file must satisfy one criterion of each type in order for the rule's action statements to apply.

In the following example, a file must reside in one of db/datafiles, db/indexes, or db/logs and be owned by one of DBA_Manager, MFG_DBA, or HR_DBA to be designated for possible action:

```
<SELECT>
  <DIRECTORY Flags="nonrecursive">db/datafiles</DIRECTORY>
  <DIRECTORY Flags="nonrecursive">db/indexes</DIRECTORY>
  <DIRECTORY Flags="nonrecursive">db/logs</DIRECTORY>
  <USER>DBA_Manager</USER>
  <USER>MFG_DBA</USER>
  <USER>HR_DBA</USER>
</SELECT>
```

If a rule includes multiple SELECT statements, a file need only satisfy one of them to be selected for action. This property can be used to specify alternative conditions for file selection.

In the following example, a file need only reside in one of `db/datafiles`, `db/indexes`, or `db/logs` or be owned by one of DBA_Manager, MFG_DBA, or HR_DBA to be designated for possible action:

```
<SELECT>
  <DIRECTORY Flags="nonrecursive">db/datafiles</DIRECTORY>
  <DIRECTORY Flags="nonrecursive">db/indexes</DIRECTORY>
  <DIRECTORY Flags="nonrecursive">db/logs</DIRECTORY>
</SELECT>
<SELECT>
  <USER>DBA_Manager</USER>
  <USER>MFG_DBA</USER>
  <USER>HR_DBA</USER>
</SELECT>
```

Multiple placement classes in <ON> clauses of CREATE statements and in <TO> clauses of RELOCATE statements

Both the <ON> clause of the CREATE statement and the <TO> clause of the RELOCATE statement can specify priority ordered lists of placement classes using multiple <DESTINATION> XML elements. VxFS uses a volume in the first placement class in a list for the designated purpose of file creation or relocation, if possible. If no volume in the first listed class has sufficient free space or if the file system's volume set does not contain any volumes with that placement class, VxFS uses a volume in the second listed class if possible. If no volume in the second listed class can be used, a volume in the third listed class is used if possible, and so forth.

The following example illustrates of three placement classes specified in the <ON> clause of a CREATE statement:

```
<CREATE>
  <ON>
    <DESTINATION>
      <CLASS>tier1</CLASS>
    </DESTINATION>
    <DESTINATION>
      <CLASS>tier2</CLASS>
    </DESTINATION>
    <DESTINATION>
      <CLASS>tier3</CLASS>
    </DESTINATION>
  </ON>
</CREATE>
```

In this statement, VxFS would allocate space for newly created files designated by the rule's SELECT statement on tier1 volumes if space was available. If no tier1 volume had sufficient free space, VxFS would attempt to allocate space on a tier2 volume. If no tier2 volume had sufficient free space, VxFS would attempt allocation on a tier3 volume. If sufficient space could not be allocated on a volume in any of the three specified placement classes, allocation would fail with an ENOSPC error, even if the file system's volume set included volumes in other placement classes that did have sufficient space.

The <TO> clause in the RELOCATE statement behaves similarly. VxFS relocates qualifying files to volumes in the first placement class specified if possible, to volumes in the second specified class if not, and so forth. If none of the destination criteria can be met, such as if all specified classes are fully occupied, qualifying files are not relocated, but no error is signaled in this case.

Multiple placement classes in <FROM> clauses of RELOCATE and DELETE statements

The <FROM> clause in RELOCATE and DELETE statements can include multiple source placement classes. However, unlike the <ON> and <TO> clauses, no order or priority is implied in <FROM> clauses. If a qualifying file resides on a volume in any of the placement classes specified in a <FROM> clause, it is relocated or deleted regardless of the position of its placement class in the <FROM> clause list of classes.

Multiple conditions in <WHEN> clauses of RELOCATE and DELETE statements

The <WHEN> clause in RELOCATE and DELETE statements may include multiple relocation criteria. Any or all of <ACCAGE>, <MODAGE>, <SIZE>, and <IOTEMP> can be specified. When multiple conditions are specified, all must be satisfied in order for a selected file to qualify for relocation or deletion.

In the following example, a selected file would have to be both inactive, that is, not accessed, for more than 30 days and larger than 100 megabytes to be eligible for relocation or deletion:

```
<WHEN>
  <ACCAGE Units="days">
    <MIN Flags="gt">30</MIN>
  </ACCAGE>
  <SIZE Units="MB">
    <MIN Flags="gt">100</MIN>
  </SIZE>
</WHEN>
```

You cannot write rules to relocate or delete a single designated set of files if the files meet one of two or more relocation or deletion criteria.

File placement policy rule and statement ordering

You can use the Dynamic Storage Tiering graphical user interface (GUI) to create any of four types of file placement policy documents. Alternatively, you can use a text editor or XML editor to create XML policy documents directly. The GUI places policy rule statements in the correct order to achieve the desired behavior. If you use a text editor, it is your responsibility to order policy rules and the statements in them so that the desired behavior results.

The rules that comprise a placement policy may occur in any order, but during both file allocation and `fsppadm enforce` relocation scans, the first rule in which a file is designated by a SELECT statement is the only rule against which that file is evaluated. Thus, rules whose purpose is to supersede a generally applicable behavior for a special class of files should precede the general rules in a file placement policy document.

The following XML snippet illustrates faulty rule placement with potentially unintended consequences:

```
<?xml version="1.0"?>
<!DOCTYPE FILE_PLACEMENT_POLICY SYSTEM "placement.dtd">
<FILE_PLACEMENT_POLICY Version="5.0">
  <RULE Name="GeneralRule">
    <SELECT>
      <PATTERN>*</PATTERN>
    </SELECT>
    <CREATE>
      <ON>
        <DESTINATION>
          <CLASS>tier2</CLASS>
        </DESTINATION>
      </ON>
    </CREATE>
    ...other statements...
  </RULE>
  <RULE Name="DatabaseRule">
    <SELECT>
      <PATTERN>*.db</PATTERN>
    </SELECT>
    <CREATE>
      <ON>
        <DESTINATION>
          <CLASS>tier1</CLASS>
        </DESTINATION>
      </ON>
    </CREATE>
    ...other statements...
  </RULE>
</FILE_PLACEMENT_POLICY>
```

The GeneralRule rule specifies that all files created in the file system, designated by <PATTERN>*</PATTERN>, should be created on tier2 volumes. The DatabaseRule rule specifies that files whose names include an extension of .db

should be created on tier1 volumes. The GeneralRule rule applies to any file created in the file system, including those with a naming pattern of *.db, so the DatabaseRule rule will never apply to any file. This fault can be remedied by exchanging the order of the two rules. If the DatabaseRule rule occurs first in the policy document, VxFS encounters it first when determining where to new place files whose names follow the pattern *.db, and correctly allocates space for them on tier1 volumes. For files to which the DatabaseRule rule does not apply, VxFS continues scanning the policy and allocates space according to the specification in the CREATE statement of the GeneralRule rule.

A similar consideration applies to statements within a placement policy rule. VxFS processes these statements in order, and stops processing on behalf of a file when it encounters a statement that pertains to the file. This can result in unintended behavior.

The following XML snippet illustrates a RELOCATE statement and a DELETE statement in a rule that is intended to relocate if the files have not been accessed in 30 days, and delete the files if they have not been accessed in 90 days:

```
<RELOCATE>
  <TO>
    <DESTINATION>
      <CLASS>tier2</CLASS>
    </DESTINATION>
  </TO>
  <WHEN>
    <ACCAGE Units="days">
      <MIN Flags="gt">30</MIN>
    </ACCAGE>
  </WHEN>
</RELOCATE>
<DELETE>
  <WHEN>
    <ACCAGE Units="days">
      <MIN Flags="gt">90</MIN>
    </ACCAGE>
  </WHEN>
</DELETE>
```

As written with the RELOCATE statement preceding the DELETE statement, files will never be deleted, because the <WHEN> clause in the RELOCATE statement applies to all selected files that have not been accessed for at least 30 days. This includes those that have not been accessed for 90 days. VxFS ceases to process a file against a placement policy when it identifies a statement that applies to that file, so the DELETE statement would never occur. This example illustrates the

general point that RELOCATE and DELETE statements that specify less inclusive criteria should precede statements that specify more inclusive criteria in a file placement policy document. The GUI automatically produce the correct statement order for the policies it creates.

File placement policies and extending files

In a VxFS file system with an active file placement policy, the placement class on whose volume a file resides is part of its metadata, and is attached when it is created and updated when it is relocated. When an application extends a file, VxFS allocates the incremental space on the volume occupied by the file if possible. If not possible, VxFS allocates the space on another volume in the same placement class. For example, if a file is created on a tier1 volume and later relocated to a tier2 volume, extensions to the file that occur before the relocation have space allocated on a tier1 volume, while those occurring after to the relocation have their space allocated on tier2 volumes. When a file is relocated, all of its allocated space, including the space acquired by extension, is relocated to tier2 volumes in this case.

Chapter 11

Quick I/O for Databases

This chapter includes the following topics:

About Quick I/O

Quick I/O for Databases (referred to as Quick I/O) allows applications to 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 part of the VRTSvxfs package, but is available for use only with other Symantec products.

See the Veritas Storage Foundation Release Notes.

About Quick I/O functionality and performance

Many database administrators (DBAs) create databases on file systems because file systems make common administrative tasks, such as moving, copying, and backing up, much simpler. However, putting databases on file systems significantly reduces database performance. By using 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.

Quick I/O provides higher database performance in the following ways:

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

About asynchronous I/O kernel support

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 allows the database server to 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.

About direct I/O support

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.

About Kernel write locks avoidance

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.

About double buffering avoidance

Most database servers implement their own buffer cache and do not need the system buffer cache. Thus, 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.

About using Veritas File System files as raw character devices

When VxFS with Quick I/O is installed, files may be accessed by the following ways:

- 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

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

About the 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:
```

The cdev portion is an acronym for character device. Whenever an application opens an existing VxFS file with the suffix ::cdev:vxfs, 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. Symantec advises you to reserve the extension only for Quick I/O files.

About use restrictions

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

- The name `xxx::cdev:vxfs:` is recognized as a special name by VxFS only when the following conditions are met:
  - The qio module is loaded
  - Quick I/O has a valid license
  - The regular file `xxx` is physically present on the VxFS file system
  - There is no regular file named `xxx::cdev:vxfs:` on the system
- If the file `xxx` is being used for memory mapped I/O, it cannot be accessed as a Quick I/O file.
- 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:`.
- The size of the file cannot be extended by writes through the Quick I/O interface.

About 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 the 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.

The `qiomkfile` command creates two files: a regular file with preallocated, contiguous space; and a symbolic link pointing to the Quick I/O name extension.

Creating a Quick I/O file using qiomkfile

The following example shows how to create a Quick I/O file using the `qiomkfile` command.

See the qiomkfile(1) manual page.

To create a Quick I/O file using qiomkfile

1 Create a 100 MB file named dbfile in /database:

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

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.

- If you specify the `-a` option with `qiomkfile`, an absolute path name is used, such as the following:

```
/database/dbfile points to /database/.dbfile::cdev:vxfs:
```

See "About absolute and relative path names" on page 190.

2 Check the results:

```
$ 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:
```

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 specified the `-a` option with `qiomkfile`, the results are as follows:

```
$ 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:
```

Accessing regular VxFS files through symbolic links

One 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. Any database or application can then access the file as a raw character device.

See the Veritas Editions product documentation.

The following example creates a 100 MB Quick I/O file named dbfile on the VxFS file system `/database` that can be accessed through a symbolic link.

To access a file through a symbolic link

1. Go to the `/database` file system:

```
$ cd /database
```

2. Create a 100 MB Quick I/O file named `dbfile`:

```
$ dd if=/dev/zero of=/database/.dbfile bs=128k count=800
```

   The `dd` command preallocates the file space.

3. Create a symbolic link to `dbfile`:

```
$ ln -s .dbfile::cdev:vxfs: /database/dbfile
```

About absolute and 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, 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 can 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, but 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.

See the setext(1) manual page.

The following example shows how to use `setext` to create a 100 MB database file for an Oracle database.

To preallocate files using setext

1 Go to the `/database` file system:

```
# cd /database
```

2 Create the `.dbfile` file:

```
# touch .dbfile
```

3 Reserve 100 MB for the `.dbfile` file using `setext`:

```
# setext -r 102400 -f noreserve -f chgsize .dbfile
```

4 Create a symbolic link to `.dbfile`:

```
# ln -s .dbfile::cdev:vxfs: dbfile
```

5 Change the owner of `dbfile` to `oracle`:

```
# chown oracle dbfile
```

6 Change the group of `dbfile` to `dba`:

```
# chgrp dba dbfile
```

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. Oracle requires additional space for one Oracle header size. In the following 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`*.

To create a tablespace with an Oracle database

1 Create the file `dbfile` and preallocate 100 MB for the file:

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

2 Start the Oracle database:

```
$ sqlplus /nolog
```

3 Create the tablespace:

```
SQL> connect / as sysdba
SQL> create tablespace ts1 datafile '/database/dbfile' size 99M;
SQL> exit;
```

Using Quick I/O with Sybase databases

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.

The following example creates a 100 megabyte master device masterdev on the file system /sybmaster.

To create a new Sybase database device

1 Go to the `/sybmaster` file system:

```
$ cd /sybmaster
```

2 Create the `masterdev` file and preallocate 100 MB for the file:

```
$ qiomkfile -s 100m masterdev
```

You can use this master device while running the `sybsetup` program or `sybinit` script.

3 To create the master device directly, enter:

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

4 Add a new 500 megabyte database device `datadev` to the file system `/sybdata` on your dataserver:

```
$ cd /sybdata
$ qiomkfile -s 500m datadev
...
```

5 Start the Sybase database:

```
$ isql -U sa -P sa_password -S dataserver_name
```

6 Set up the datadev database device:

```
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. The `-o qio` and `-o noqio` mount options enable and disable, respectively, Quick I/O when a file system is mounted.

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
```

To enable or disable Quick I/O

1 Specify the `-o qio` mount option to enable Quick I/O:

```
# mount -F vxfs -o qio MyFS
```

2 Specify the `-o noqio` mount option to disable Quick I/O:

```
# mount -F vxfs -o noqio MyFS
```

About Cached Quick I/O for databases

A 32-bit application (such as a 32-bit database) 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.

With 64-bit applications, for which limited memory is not a critical problem, using the file system cache still provides performance benefits by using the 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.

Enabling Cached Quick I/O

Caching for Quick I/O files can be enabled online when the database is running by using the `vxtunefs` utility and the `qioadmin` command.

See the vxtunefs(1M) and qioadmin(1) manual pages.

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

To enable caching

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.

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.

Note: The `vxtunefs` command enables caching for all the Quick I/O files on the file system.

The following example enables Cached Quick I/O for the file system `/database01`.

To enable Cached Quick I/O for a file system

1. Enable Cached Quick I/O:

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

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

2. If desired, make this setting persistent across mounts by adding a file system entry in the file `/etc/vx/tunefstab`:

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

   See the tunefstab(4) manual page.

Manipulating Cached Quick I/O settings for individual files

A Quick I/O file's Cached Quick I/O settings are manipulated with the `vxtunefs` utility and the `qioadmin` command.

See the vxtunefs(1M) and qioadmin(1) manual pages.

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 enable caching on a file

- Enable caching on a file:

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

To disable caching on a file

- Disable caching on a file:

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

To make the caching 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
```

To check on the current cache advisory settings for a file

- Check the current cache advisory settings:

```
$ qioadmin -P filename mount_point
filename,OFF
```

To check the setting of the qio_cache_enable flag for a file system

- Check the setting of the *`qio_cache_enable`* flag:

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

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

About 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.

Increasing database performance using Quick I/O

Perform the following steps to increase database performance on a VxFS file system using Quick I/O.

See the Veritas Editions product documentation.

See the qioadmin(1) and vxtunefs(1M) manual pages.

To increase database performance

1. Verify 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, or use the `qiomkfile` command. 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:`.

Appendix A

Quick Reference

This appendix includes the following topics:

Command summary

Symbolic links to all VxFS command executables are installed in the `/opt/VRTS/bin` directory. Add this directory to the end of your *PATH* environment variable to access the commands.

Table A-1 describes the VxFS-specific commands.

Table A-1 VxFS commands

Command	Description
cfscluster	CFS cluster configuration command. This functionality is available only with the Veritas Cluster File System product.
cfsdgadm	Adds or deletes shared disk groups to or from a cluster configuration. This functionality is available only with the Veritas Cluster File System product.
cfsmntadm	Adds, deletes, modifies, and sets policy on cluster mounted file systems. This functionality is available only with the Veritas Cluster File System product.
cfsmount, cfsumount	Mounts or unmounts a cluster file system. This functionality is available only with the Veritas Cluster File System product.
cp	Copies files and directories on VxFS file systems.
df	Reports the number of free disk blocks and inodes for a VxFS file system.
fcladm	Administers VxFS File Change Logs.
ff	Lists file names and inode information for a VxFS file system.
fiostat	Administers file I/O statistics
fsadm	Resizes or defragments a VxFS file system.
fsapadm	Administers VxFS allocation policies.
fscat	Cats a VxFS file system.
fscdsadm	Performs online CDS operations.
fscdsconv	Performs offline CDS migration tasks on VxFS file systems.
fscdstask	Performs various CDS operations.
fsck	Checks and repairs a VxFS file system.
fsckpt_restore	Restores file systems from VxFS Storage Checkpoints.
fsckptadm	Administers VxFS Storage Checkpoints.
fsclustadm	Manages cluster-mounted VxFS file systems. This functionality is available only with the Veritas Cluster File System product.
fsdb	Debugs VxFS file systems.
fsdbencap	Encapsulates databases.
fsmap	Displays VxFS file system extent information.

Table A-1 VxFS commands *(continued)*

Command	Description
fsppadm	Administers VxFS placement policies.
fstyp	Returns the type of file system on a specified disk partition.
fsvmap	Maps volumes of VxFS file systems to files.
fsvoladm	Administers VxFS volumes.
glmconfig	Configures Group Lock Managers (GLM). This functionality is available only with the Veritas Cluster File System product.
ls	Lists files and directories on a VxFS file system.
mkfs	Constructs a VxFS file system.
mount	Mounts a VxFS file system.
mv	Moves files and directories on a VxFS file system.
ncheck	Generates path names from inode numbers for a VxFS file system.
qioadmin	Administers VxFS Quick I/O for Databases cache.
qiomkfile	Creates a VxFS Quick I/O device file. This functionality is available only with the Veritas Quick I/O for Databases feature.
qiostat	Displays statistics for VxFS Quick I/O for Databases. This functionality is available only with the Veritas Quick I/O for Databases feature.
setext	Sets extent attributes on a file in a VxFS file system.
umount_vxfs	Unmounts a VxFS file system.
vxdump	Incrementally dumps file systems.
vxedquota	Edits user quotas for a VxFS file system.
vxenablef	Enables specific VxFS features.
vxfsconvert	Converts an unmounted file system to VxFS or upgrades a VxFS disk layout version.
vxfsstat	Displays file system statistics.
vxlsino	Looks up VxFS reverse path names.
vxquot	Displays file system ownership summaries for a VxFS file system.
vxquota	Displays user disk quotas and usage on a VxFS file system.

Table A-1 VxFS commands *(continued)*

Command	Description
vxquotaoff vxquotaon	Turns quotas on and off for a VxFS file system.
vxrepquota	Summarizes quotas for a VxFS file system.
vxrestore	Restores a file system incrementally.
vxtunefs	Tunes a VxFS file system.
vxupgrade	Upgrades the disk layout of a mounted VxFS file system.

Online manual pages

This release includes the following online manual pages as part of the `VRTSvxfs` package. These are installed in the appropriate directories under `/opt/VRTS/man` (add this to your *`MANPATH`* environment variable), but does not update the windex database. To ensure that new VxFS manual pages display correctly, update the windex database after installing `VRTSvxfs`.

See the catman(1M) manual page.

Table A-2 describes the VxFS-specific section 1 manual pages.

Table A-2 Section 1 manual pages

Section 1	Description
cp_vxfs	Copies files and directories on a VxFS file system.
cpio_vxfs	Copies files and directories on a VxFS file system.
fiostat	Administers file I/O statistics.
getext	Gets extent attributes for a VxFS file system.
ls_vxfs	Lists files and directories on a VxFS file system.
mv_vxfs	Moves files and directories on a VxFS file system.
qioadmin	Administers VxFS Quick I/O for Databases cache. This functionality is available only with the Veritas Quick I/O for Databases feature.
qiomkfile	Creates a VxFS Quick I/O device file. This functionality is available only with the Veritas Quick I/O for Databases feature.

Table A-2 Section 1 manual pages *(continued)*

Section 1	Description
qiostat	Displays statistics for VxFS Quick I/O for Databases. This functionality is available only with the Veritas Quick I/O for Databases feature.
setext	Sets extent attributes on a file in a VxFS file system.

Table A-3 describes the VxFS-specific section 1M manual pages.

Table A-3 Section 1M manual pages

Section 1M	Description
cfscluster	Configures CFS clusters. This functionality is available only with the Veritas Cluster File System product.
cfsdgadm	Adds or deletes shared disk groups to/from a cluster configuration. This functionality is available only with the Veritas Cluster File System product.
cfsmntadm	Adds, deletes, modifies, and sets policy on cluster mounted file systems. This functionality is available only with the Veritas Cluster File System product.
cfsmount, cfsumount	Mounts or unmounts a cluster file system. This functionality is available only with the Veritas Cluster File System product.
df_vxfs	Reports the number of free disk blocks and inodes for a VxFS file system.
fcladm	Administers VxFS File Change Logs.
ff_vxfs	Lists file names and inode information for a VxFS file system.
fsadm_vxfs	Resizes or reorganizes a VxFS file system.
fsapadm	Administers VxFS allocation policies.
fscat_vxfs	Cats a VxFS file system.
fscdsadm	Performs online CDS operations.
fscdsconv	Performs offline CDS migration tasks on VxFS file systems.
fscdstask	Performs various CDS operations.
fsck_vxfs	Checks and repairs a VxFS file system.
fsckptadm	Administers VxFS Storage Checkpoints.
fsckpt_restore	Restores file systems from VxFS Storage Checkpoints.

Table A-3 Section 1M manual pages *(continued)*

Section 1M	Description
fsclustadm	Manages cluster-mounted VxFS file systems. This functionality is available only with the Veritas Cluster File System product.
fsdbencap	Encapsulates databases.
fsdb_vxfs	Debugs VxFS file systems.
fsmap	Displays VxFS file system extent information.
fsppadm	Administers VxFS placement policies.
fstyp_vxfs	Returns the type of file system on a specified disk partition.
fsvmap	Maps volumes of VxFS file systems to files.
fsvoladm	Administers VxFS volumes.
glmconfig	Configures Group Lock Managers (GLM). This functionality is available only with the Veritas Cluster File System product.
mkfs_vxfs	Constructs a VxFS file system.
mount_vxfs	Mounts a VxFS file system.
ncheck_vxfs	Generates path names from inode numbers for a VxFS file system.
quot	Summarizes ownership on a VxFS file system.
quotacheck_vxfs	Checks VxFS file system quota consistency.
umount_vxfs	Unmounts a VxFS file system.
vxdiskusg	Generates VxFS disk accounting data by user ID.
vxdump	Incrementally dumps file systems.
vxedquota	Edits user quotas for a VxFS file system.
vxenablef	Enables specific VxFS features.
vxfsconvert	Converts an unmounted file system to VxFS or upgrades a VxFS disk layout version.
vxfsstat	Displays file system statistics.
vxlsino	Looks up VxFS reverse path names.
vxquot	Displays file system ownership summaries for a VxFS file system.
vxquota	Displays user disk quotas and usage on a VxFS file system.

Table A-3 Section 1M manual pages *(continued)*

Section 1M	Description
vxquotaoff vxquotaon	Turns quotas on and off for a VxFS file system.
vxrepquota	Summarizes quotas for a VxFS file system.
vxrestore	Restores a file system incrementally.
vxtunefs	Tunes a VxFS file system.
vxupgrade	Upgrades the disk layout of a mounted VxFS file system.

Table A-4 describes the VxFS-specific section 3 manual pages.

Table A-4 Section 3 manual pages

Section 3	Description
vxfs_ap_alloc2	Allocates an fsap_info2 structure.
vxfs_ap_assign_ckpt	Assigns an allocation policy to file data and metadata in a Storage Checkpoint.
vxfs_ap_assign_file	Assigns an allocation policy for file data and metadata.
vxfs_ap_assign_file_pat	Assigns a pattern-based allocation policy for a directory.
vxfs_ap_assign_fs	Assigns an allocation policy for all file data and metadata within a specified file system.
vxfs_ap_assign_fs_pat	Assigns an pattern-based allocation policy for a file system.
vxfs_ap_define	Defines a new allocation policy.
vxfs_ap_define2	Defines a new allocation policy.
vxfs_ap_enforce_file	Ensures that all blocks in a specified file match the file allocation policy.
vxfs_ap_enforce_file2	Reallocates blocks in a file to match allocation policies.
vxfs_ap_enumerate	Returns information about all allocation policies.
vxfs_ap_enumerate2	Returns information about all allocation policies.
vxf_ap_free2	Frees one or more fsap_info2 structures.
vxfs_ap_query	Returns information about a specific allocation policy.
vxfs_ap_query2	Returns information about a specific allocation policy.

Table A-4 Section 3 manual pages *(continued)*

Section 3	Description
vxfs_ap_query_ckpt	Returns information about allocation policies for each Storage Checkpoint.
vxfs_ap_query_file	Returns information about allocation policies assigned to a specified file.
vxfs_ap_query_file_pat	Returns information about the pattern-based allocation policy assigned to a directory.
vxfs_ap_query_fs	Retrieves allocation policies assigned to a specified file system.
vxfs_ap_query_fs_pat	Returns information about the pattern-based allocation policy assigned to a file system.
vxfs_ap_remove	Deletes a specified allocation policy.
vxfs_fcl_sync	Sets a synchronization point in the VxFS File Change Log.
vxfs_fiostats_dump	Returns file and sub-file I/O statistics.
vxfs_fiostats_getconfig	Gets sub-file I/O statistics configuration values.
vxfs_fiostats_set	Turns on and off sub-file I/O statistics and resets statistics counters.
vxfs_get_ioffsets	Obtains VxFS inode field offsets.
vxfs_inotopath	Returns path names for a given inode number.
vxfs_nattr_check vxfs_nattr_fcheck	Checks for the existence of named data streams.
vxfs_nattr_link	Links to a named data stream.
vxfs_nattr_open	Opens a named data stream.
vxfs_nattr_rename	Renames a named data stream.
vxfs_nattr_unlink	Removes a named data stream.
vxfs_nattr_utimes	Sets access and modification times for named data streams.
vxfs_vol_add	Adds a volume to a multi-volume file system.
vxfs_vol_clearflags	Clears specified flags on volumes in a multi-volume file system.
vxfs_vol_deencapsulate	De-encapsulates a volume from a multi-volume file system.
vxfs_vol_encapsulate	Encapsulates a volume within a multi-volume file system.

Table A-4 Section 3 manual pages *(continued)*

Section 3	Description
vxfs_vol_encapsulate_bias	Encapsulates a volume within a multi-volume file system.
vxfs_vol_enumerate	Returns information about the volumes within a multi-volume file system.
vxfs_vol_queryflags	Queries flags on volumes in a multi-volume file system.
vxfs_vol_remove	Removes a volume from a multi-volume file system.
vxfs_vol_resize	Resizes a specific volume within a multi-volume file system.
vxfs_vol_setflags	Sets specified flags on volumes in a multi-volume file system.
vxfs_vol_stat	Returns free space information about a component volume within a multi-volume file system.

Table A-5 describes the VxFS-specific section 4 manual pages.

Table A-5 Section 4 manual pages

Section 4	Description
fs_vxfs	Provides the format of a VxFS file system volume.
inode_vxfs	Provides the format of a VxFS file system inode.
tunefstab	Describes the VxFS file system tuning parameters table.

Table A-6 describes the VxFS-specific section 7 manual pages.

Table A-6 Section 7 manual pages

Section 7	Description
vxfsio	Describes the VxFS file system control functions.

Creating a VxFS file system

The `mkfs` command creates a VxFS file system by writing to a special character device file. The special character device is a location or character device node of a particular storage device. `mkfs` builds a file system with a root directory and a `lost+found` directory.

Before running `mkfs`, you must create the target device. Refer to your operating system documentation for more information. If you are using a logical device (such as a VxVM volume), see the VxVM documentation for instructions on device initialization.

See the mkfs(1M), and mkfs_vxfs(1M) manual pages.

To create a file system

- Use the `mkfs` command to create a file system:

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

-F vxfs	Specifies the VxFS file system type.
-m	Displays the command line that was used to create the file system. The file system must already exist. This option enables you to determine the parameters used to construct the file system.
generic_options	Options common to most other file system types.
-o 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 users to create files larger than two gigabytes. The default option is largefiles.
special	Specifies the special device file location or character device node of a particular storage device.
size	Specifies the number of 512-byte sectors in the file system. If *`size`* is not specified, `mkfs` determines the size of the special device.

Example of creating a file system

The following example creates a VxFS file system of 12288 sectors in size on a VxVM volume.

To create a VxFS file system

1 Create the file system:

```
# mkfs -F vxfs /dev/vx/rdsk/diskgroup/volume 12288
version 7 layout
12288 sectors, 6144 blocks of size 1024, log size 512 blocks
largefiles supported
```

2 Mount the newly created file system.

Converting a file system to VxFS

The `vxfsconvert` command can be used to convert a UFS file system to a VxFS file system.

See the vxfsconvert(1M) manual page.

To convert a UFS file system to a VxFS file system

- Use the `vxfsconvert` command to convert a UFS file system to VxFS:

```
vxfsconvert [-l logsize] [-s size] [-efnNvyY] special
```

-e	Estimates the amount of space required to complete the conversion.
-f	Displays the list of supported file system types.
-l logsize	Specifies the size of the file system intent log.
-n|N	Assumes a no response to all questions asked by `vxfsconvert`.
-s size	Directs `vxfsconvert` to use free disk space past the current end of the file system to store VxFS metadata.
-v	Specifies verbose mode.
-y|Y	Assumes a yes response to all questions asked by `vxfsconvert`.
special	Specifies the name of the character (raw) device that contains the file system to convert.

Example of converting a file system

The following example converts a UFS file system to a VxFS file system with an intent log size of 4096 blocks.

To convert a UFS file system to a VxFS file system

- Convert the file system:

```
# vxfsconvert -l 4096 /dev/vx/rdsk/diskgroup/volume
```

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.

To mount a file system

- Use the `mount` command to mount a 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 Storage Checkpoint.
-o cluster	Mounts a file system in shared mode. Available only with the VxFS cluster file system feature.
special	A VxFS 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.

The following table lists some of the specific_options:

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. The default option is largefiles.
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 `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 "About Quick I/O" on page 185.
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.
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 command options" on page 30.

See the fsckptadm(1M), mount(1M), mount_vxfs(1M), and vfstab(4) manual pages.

Example of mounting a file system

The following example mounts the file system `/dev/vx/dsk/fsvol/vol1` on the `/ext` directory with read/write access and delayed logging.

To mount the file system

- Mount the file system:

```
# mount -F vxfs -o delaylog /dev/vx/dsk/fsvol/vol1 /ext
```

Editing the vfstab file

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

You must specify the following:

- 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.

The following 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/vx/dsk/fsvol/vol1	/dev/vx/rdsk/fsvol/vol1	/ext	vxfs	1	yes	–

Unmounting a file system

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

See the umount_vxfs(1M) manual page.

To unmount a file system

- Use the `umount` command to unmount a file system:

  ```
  umount [-F vxfs] [generic_options] [-o [force]] {special|mount_point}
  ```

  Specify the file system to be unmounted as a *mount_point* or *special*. *special* is the VxFS block special device on which the file system resides.

Example of unmounting a file system

The following are examples of unmounting file systems.

To unmount the file system /dev/vx/dsk/fsvol/vol1

- Unmount the file system:

  ```
  # umount /dev/vx/dsk/fsvol/vol1
  ```

To unmount all file systems not required by the system

- Unmount the file systems:

  ```
  # umount -a
  ```

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

Displaying information on mounted file systems

Use the `mount` command to display a list of currently mounted file systems.

See the mount(1M) and mount_vxfs(1M) manual pages.

To view the status of mounted file systems

- Use the `mount` command to view the status of mounted file systems:

  ```
  mount -v
  ```

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

Example of displaying information on mounted file systems

The following example shows the result of invoking the `mount` command without options.

To display information on mounted file systems

- Invoke the `mount` command without options:

```
# mount
/ on /dev/root read/write/setuid on Thu May 26 16:58:24 2004
/proc on /proc read/write on Thu May 26 16:58:25 2004
/dev/fd on /dev/fd read/write on Thu May 26 16:58:26 2004
/tmp on /tmp read/write on Thu May 26 16:59:33 2004
/var/tmp on /var/tmp read/write on Thu May 26 16:59:34 2004
```

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.

See the fstyp(1M) and fstyp_vxfs(1M) manual pages.

To determine a file system's type

- Use the `fstyp` command to determine a file system's type:

```
fstyp -v special
```

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

Example of determining a file system's type

The following example uses the `fstyp` command to determine a the file system type of the `/dev/vx/dsk/fsvol/vol1` device.

To determine the file system's type

- Use the `fstyp` command to determine the file system type of the device `/dev/vx/dsk/fsvol/vol1`:

```
# fstyp -v /dev/vx/dsk/fsvol/vol1
```

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

```
vxfs
magic a501fcf5  version 7 ctime Tue Jun 25 18:29:39 2003
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
```

Resizing a file system

You can extend or shrink mounted VxFS file systems using the `fsadm` command. Use the `extendfs` command to extend the size of an unmounted file system. A file system using the Version 4 disk layout can be up to two terabytes in size. A file system using the Version 5 disk layout can be up to 32 terabytes in size. A file system using the Version 6 or 7 disk layout can be up to 8 exabytes in size. The size to which a Version 5, 6, or 7 disk layout file system can be increased depends on the file system block size.

See "About disk layouts" on page 275.

See the format(1M) and fsadm_vxfs(1M) manual pages.

Extending 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).

Note: If a file system is full, busy, or too fragmented, the resize operation may fail.

The device must have enough space to contain the larger file system.

See the format(1M) manual page.

See the *Veritas Volume Manager Administrator's Guide*.

To extend a VxFS file system

- Use the `fsadm` command to extend a VxFS file system:

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

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.

Example of extending a file system

The following is an example of extending a file system with the `fsadm` command.

To extend a file system

- Extend the VxFS file system mounted on `/ext` to 22528 sectors:

```
# /usr/lib/fs/vxfs/fsadm -b 22528 /ext
```

Shrinking 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 toward the end of the file system, shrinking may not be possible. If a file system is full, busy, or too fragmented, the resize operation may fail.

To decrease the size of a VxFS file system

- Use the `fsadm` command to decrease the size of a VxFS file system:

```
fsadm [-F vxfs] [-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 of shrinking a file system

The following example shrinks a VxFS file system mounted at `/ext` to 20480 sectors.

To shrink a VxFS file system

- Shrink a VxFS file system mounted at `/ext` to 20480 sectors:

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

Warning: After this operation, there is unused space at the end of the device. You can then resize the device, but be careful not to make the device smaller than the new size of the file system.

Reorganizing 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.

Note: If a file system is full or busy, the reorg operation may fail.

To reorganize a VxFS file system

- Use the `fsadm` command to reorganize a VxFS file system:

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

vxfs	The file system type.
-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 of reorganizing a file system

The following example reorganizes the file system mounted at `/ext`.

To reorganize a VxFS file system

- Reorganize the VxFS file system mounted at `/ext`:

```
# fsadm -F vxfs -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 "About snapshot file systems" on page 93.

See the mount(1M), mount_vxfs(1M), vxdump(1M), and vxrestore(1M) manual pages.

Creating and mounting 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 `mount` command to create and mount a snapshot of a VxFS file system:

```
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 of creating and mounting a snapshot of a VxFS file system

The following example creates a snapshot file system of the file system at `/home` on `/dev/vx/dsk/fsvol/vol1`, and mounts it at `/snapmount`.

To create and mount a snapshot file system of a file system

- Create a snapshot file system of the file system at `/home` on `/dev/vx/dsk/fsvol/vol1` and mount it at `/snapmount`:

```
# mount -F vxfs -o snapof=/home, \
snapsize=32768 /dev/vx/dsk/fsvol/vol1 /snapmount
```

  You can now back up the file system.

Backing up a file system

After creating a snapshot file system, you can use `vxdump` to back it up.

To back up a VxFS snapshot file system

- Use the `vxdump` command to back up a VxFS snapshot file system:

```
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 of backing up a file system

The following example backs up the VxFS snapshot file system mounted at `/snapmount` to the tape drive with device name `/dev/rmt/00m`.

To back up a VxFS snapshot file system

- Back up the VxFS snapshot file system mounted at `/snapmount` to the tape drive with device name `/dev/rmt/00m`:

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

Restoring 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 `vxrestore` command to restore a VxFS snapshot file system:

```
vxrestore [-v] [-x] [filename]
```

-v	Specifies verbose mode.
-x	Extracts the named files from the tape.
filename	The file or directory to restore. If filename is omitted, the root directory, and thus the entire tape, is extracted.

Example of restoring a file system

The following example restores a VxFS snapshot file system from the tape:

`/dev/st1` into the mount point `/restore`

To restore a VxFS snapshot file system

- Restore a VxFS snapshot file system from the tape `/dev/st1` into the mount point `/restore`:

```
# cd /restore
# vxrestore -v -x -f /dev/st1
```

Using quotas

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

See "Using quotas" on page 104.

See the vxquota(1M), vxquotaon(1M), vxquotaoff(1M), and vxedquota(1M) manual pages.

Turning 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

1 Turn on quotas for a mounted file system:

```
vxquotaon mount_point
```

2 Mount a file system and turn on quotas at the same time:

```
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 of turning on quotas for a mounted file system

The following example creates a quoatas file and turns on quotas for a VxFS file system mounted at `/mnt`.

To turn on quotas for a mounted file system

- Create a quotas file if it does not already exist and turn on quotas for a VxFS file system mounted at `/mnt`:

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

Example of turning on quotas at mount time

The following example turns on quotas when the `/dev/vx/dsk/fsvol/vol1` file system is mounted.

To turn on quotas for a file system at mount time

- Turn on quotas at mount time by specifying the `-o quota` option:

```
# mount -F vxfs -o quota /dev/vx/dsk/fsvol/vol1 /mnt
```

Setting 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 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 the following:

```
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 set up user quotas

1 Invoke the quota editor:

```
vxedquota username
```

2 Modify the time limit:

```
vxedquota -t
```

Viewing quotas

The superuser or individual user can view disk quotas and usage on VxFS file systems using the vxquota command. 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.

To view quotas for a specific user

- Use the vxquota command to view quotas for a specific user:

```
vxquota -v username
```

Turning off quotas

You can turn off quotas for a mounted file system using the vxquotaoff command.

To turn off quotas for a file system

- Turn off quotas for a file system:

```
vxquotaoff mount_point
```

Example of turning off quotas

The following example turns off quotas for a VxFS file system mounted at `/mnt`.

To turn off quotas

- Turn off quotas for a VxFS file system mounted at `/mnt`:

```
# vxquotaoff /mnt
```

Appendix B

Diagnostic messages

This appendix includes the following topics:

File system response to problems

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

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.

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 the fsck_vxfs(1M) manual page.

To execute a full structural check

- Use the `fsck` command to execute a full structural check:

```
# fsck -F vxfs -o full -y /dev/vx/rdsk/diskgroup/volume
```

Warning: Be careful when running this command. By specifying the `-y` option, all `fsck` user prompts are answered with a "yes", which can make irreversible changes if it performs a full file system check.

About kernel messages

Kernel messages are 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.

About global message IDs

When a VxFS kernel message displays on the system console, it is preceded by a numerical ID shown in the msgcnt field. This ID number increases with each

instance of the message to guarantee that the sequence of events is known when analyzing 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.

Kernel messages

Some commonly encountered kernel messages are described on the following table:

Table B-1 Kernel messages

Message Number	Message and Definition
001	NOTICE: msgcnt x: mesg 001: V-2-1: vx_nospace - *`mount_point`* file system full (n block extent) ■ Description 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 the `fsadm` command. See the fsadm_vxfs(1M) manual page.

Table B-1 Kernel messages *(continued)*

Message Number	Message and Definition
002	WARNING: msgcnt x: mesg 002: V-2-2: vx_snap_strategy - *mount_point* file system write attempt to read-only file system WARNING: msgcnt x: mesg 002: V-2-2: 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.
003, 004, 005	WARNING: msgcnt x: mesg 003: V-2-3: vx_mapbad - *mount_point* file system free extent bitmap in au aun marked bad WARNING: msgcnt x: mesg 004: V-2-4: vx_mapbad - *mount_point* file system free inode bitmap in au aun marked bad WARNING: msgcnt x: mesg 005: V-2-5: 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 cannot 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.

Table B-1 Kernel messages *(continued)*

Message Number	Message and Definition
006, 007	WARNING: msgcnt x: mesg 006: V-2-6: vx_sumupd - *mount_point* file system summary update in au aun failed WARNING: msgcnt x: mesg 007: V-2-7: 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 cannot 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.
008, 009	WARNING: msgcnt x: mesg 008: V-2-8: vx_direrr: function - *mount_point* file system dir inode *dir_inumber* dev/block device_ID/block dirent inode *dirent_inumber* error *errno* WARNING: msgcnt x: mesg 009: V-2-9: vx_direrr: function - *mount_point* file system dir inode *dir_inumber* dirent inode *dirent_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.

Table B-1 Kernel messages *(continued)*

Message Number	Message and Definition
010	WARNING: msgcnt x: mesg 010: V-2-10: 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: mesg 011: V-2-11: 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. See the fsadm_vxfs(1M) online manual page.
012	WARNING: msgcnt x: mesg 012: V-2-12: 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.

Table B-1 Kernel messages *(continued)*

Message Number	Message and Definition
013	WARNING: msgcnt x: mesg 013: V-2-13: 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 cannot 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: mesg 014: V-2-14: 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 "Using kernel tunables" on page 40.

Table B-1 Kernel messages *(continued)*

Message Number	Message and Definition
015	WARNING: msgcnt x: mesg 015: V-2-15: vx_ibadinactive - *mount_point* file system cannot mark inode *inumber* bad WARNING: msgcnt x: mesg 015: V-2-15: vx_ilisterr - *mount_point* file system cannot 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: mesg 016: V-2-16: 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.

Table B-1 Kernel messages *(continued)*

Message Number	Message and Definition
017	

Table B-1 Kernel messages *(continued)*

Message Number	Message and Definition
	WARNING: msgcnt x: mesg 017: V-2-17: vx_attr_getblk - *mount_point* file system inode *inumber* marked bad in core
	WARNING: msgcnt x: mesg 017: V-2-17: vx_attr_iget - *mount_point* file system inode *inumber* marked bad in core
	WARNING: msgcnt x: mesg 017: V-2-17: vx_attr_indadd - *mount_point* file system inode *inumber* marked bad in core
	WARNING: msgcnt x: mesg 017: V-2-17: vx_attr_indtrunc - *mount_point* file system inode *inumber* marked bad in core
	WARNING: msgcnt x: mesg 017: V-2-17: vx_attr_iremove - *mount_point* file system inode *inumber* marked bad in core
	WARNING: msgcnt x: mesg 017: V-2-17: vx_bmap - *mount_point* file system inode *inumber* marked bad in core
	WARNING: msgcnt x: mesg 017: V-2-17: vx_bmap_indirect_ext4 - *mount_point* file system inode *inumber* marked bad in core
	WARNING: msgcnt x: mesg 017: V-2-17: vx_delbuf_flush - *mount_point* file system inode *inumber* marked bad in core
	WARNING: msgcnt x: mesg 017: V-2-17: vx_dio_iovec - *mount_point* file system inode *inumber* marked bad in core
	WARNING: msgcnt x: mesg 017: V-2-17: vx_dirbread - *mount_point* file system inode *inumber* marked bad in core
	WARNING: msgcnt x: mesg 017: V-2-17: vx_dircreate - *mount_point* file system inode *inumber* marked bad in core
	WARNING: msgcnt x: mesg 017: V-2-17: vx_dirlook - *mount_point* file system inode *inumber* marked bad in core
	WARNING: msgcnt x: mesg 017: V-2-17: vx_doextop_iau - *mount_point* file system inode *inumber* marked bad in core
	WARNING: msgcnt x: mesg 017: V-2-17: vx_doextop_now - *mount_point* file system inode *inumber* marked bad in core
	WARNING: msgcnt x: mesg 017: V-2-17: vx_do_getpage - *mount_point* file system inode *inumber* marked bad in core
	WARNING: msgcnt x: mesg 017: V-2-17: vx_enter_ext4 - *mount_point* file system inode *inumber* marked bad in core
	WARNING: msgcnt x: mesg 017: V-2-17: vx_exttrunc - *mount_point* file system inode *inumber* marked bad in core

Table B-1 Kernel messages *(continued)*

Message Number	Message and Definition
	WARNING: msgcnt x: mesg 017: V-2-17: vx_get_alloc - *mount_point* file system inode *inumber* marked bad in core
017 (continued)	WARNING: msgcnt x: mesg 017: V-2-17: vx_ilisterr - *mount_point* file system inode *inumber* marked bad in core WARNING: msgcnt x: mesg 017: V-2-17: vx_indtrunc - *mount_point* file system inode *inumber* marked bad in core WARNING: msgcnt x: mesg 017: V-2-17: vx_iread - *mount_point* file system inode *inumber* marked bad in core WARNING: msgcnt x: mesg 017: V-2-17: vx_iremove - *mount_point* file system inode *inumber* marked bad in core WARNING: msgcnt x: mesg 017: V-2-17: vx_iremove_attr - *mount_point* file system inode *inumber* marked bad in core WARNING: msgcnt x: mesg 017: V-2-17: vx_logwrite_flush - *mount_point* file system inode *inumber* marked bad in core WARNING: msgcnt x: mesg 017: V-2-17: vx_oltmount_iget - *mount_point* file system inode *inumber* marked bad in core WARNING: msgcnt x: mesg 017: V-2-17: vx_overlay_bmap - *mount_point* file system inode *inumber* marked bad in core WARNING: msgcnt x: mesg 017: V-2-17: vx_readnomap - *mount_point* file system inode *inumber* marked bad in core WARNING: msgcnt x: mesg 017: V-2-17: vx_reorg_trunc - *mount_point* file system inode *inumber* marked bad in core WARNING: msgcnt x: mesg 017: V-2-17: vx_stablestore - *mount_point* file system inode *inumber* marked bad in core WARNING: msgcnt x: mesg 017: V-2-17: vx_tranitimes - *mount_point* file system inode *inumber* marked bad in core WARNING: msgcnt x: mesg 017: V-2-17: vx_trunc - *mount_point* file system inode *inumber* marked bad in core WARNING: msgcnt x: mesg 017: V-2-17: vx_write_alloc2 - *mount_point* file system inode *inumber* marked bad in core WARNING: msgcnt x: mesg 017: V-2-17: vx_write_default - *mount_point* file system inode *inumber* marked bad in core WARNING: msgcnt x: mesg 017: V-2-17: vx_zero_alloc - *mount_point* file system inode *inumber* marked bad in core

Table B-1 Kernel messages *(continued)*

Message Number	Message and Definition
017 (continued)	■ Description When inode information is no longer dependable, the kernel marks it bad in memory. This is followed by a message to mark it bad on disk as well unless the mount command ioerror option is set to disable, or there is subsequent I/O failure when updating the inode on disk. No further operations can be performed on the inode. 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. The file system can be remounted without a full `fsck` unless the `VX_FULLFSCK` flag is set for the file system.

Table B-1 Kernel messages *(continued)*

Message Number	Message and Definition
019	WARNING: msgcnt x: mesg 019: V-2-19: 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.
020	WARNING: msgcnt x: mesg 020: V-2-20: 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 cannot 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.

Table B-1 Kernel messages *(continued)*

Message Number	Message and Definition
021	WARNING: msgcnt x: mesg 021: V-2-21: 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 is not 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.

Table B-1 Kernel messages *(continued)*

Message Number	Message and Definition
022	WARNING: msgcnt x: mesg 022: V-2-22: 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 cannot be remounted for read/write access. When a root Veritas 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: mesg 023: V-2-23: vx_unmountroot - root file system is busy and cannot 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 cannot 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.

Table B-1 Kernel messages *(continued)*

Message Number	Message and Definition
024	WARNING: msgcnt x: mesg 024: V-2-24: 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 cannot 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: mesg 025: V-2-25: vx_wsuper - *mount_point* file system super-block 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: mesg 026: V-2-26: 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.

Table B-1 Kernel messages *(continued)*

Message Number	Message and Definition
027	WARNING: msgcnt x: mesg 027: V-2-27: 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: mesg 028: V-2-28: 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.

Table B-1 Kernel messages *(continued)*

Message Number	Message and Definition
029, 030	WARNING: msgcnt x: mesg 029: V-2-29: vx_snap_getbp - *mount_point* snapshot file system block map write error WARNING: msgcnt x: mesg 030: V-2-30: 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.
031	WARNING: msgcnt x: mesg 031: V-2-31: 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: mesg 032: V-2-32: 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.

Table B-1 Kernel messages *(continued)*

Message Number	Message and Definition
033	WARNING: msgcnt x: mesg 033: V-2-33: 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: mesg 034: V-2-34: vx_resetlog - *mount_point* file system cannot 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.
035	WARNING: msgcnt x: mesg 035: V-2-35: 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.

Table B-1 Kernel messages *(continued)*

Message Number	Message and Definition
036	WARNING: msgcnt x: mesg 036: V-2-36: 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 cannot 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: mesg 037: V-2-37: vx_metaioerr - function - *volume_name* file system meta data [read|write] error in dev/block device_ID/block ■ 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.

Table B-1 Kernel messages *(continued)*

Message Number	Message and Definition
038	WARNING: msgcnt x: mesg 038: V-2-38: vx_dataioerr - *volume_name* file system file data [read|write] error in dev/block device_ID/block ■ 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. 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.

Table B-1 Kernel messages *(continued)*

Message Number	Message and Definition
039	WARNING: msgcnt x: mesg 039: V-2-39: 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.
040	WARNING: msgcnt x: mesg 040: V-2-40: vx_dqbad - *`mount_point`* file system user|group 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.

Table B-1 Kernel messages *(continued)*

Message Number	Message and Definition
041	WARNING: msgcnt x: mesg 041: V-2-41: vx_dqget - *mount_point* file system user|group quota file cannot 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: mesg 042: V-2-42: vx_bsdquotaupdate - *mount_point* file system user|group_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: mesg 043: V-2-43: vx_bsdquotaupdate - *mount_point* file system user|group_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.
044	WARNING: msgcnt x: mesg 044: V-2-44: vx_bsdquotaupdate - *mount_point* file system user|group_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. ■ Action Remove some files to free up space.

Table B-1 Kernel messages *(continued)*

Message Number	Message and Definition
045	WARNING: msgcnt x: mesg 045: V-2-45: vx_bsdiquotaupdate - *mount_point* file system user|group_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: mesg 046: V-2-46: vx_bsdiquotaupdate - *mount_point* file system user|group_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: mesg 047: V-2-47: vx_bsdiquotaupdate - warning: *mount_point* file system user|group_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.

Table B-1 Kernel messages *(continued)*

Message Number	Message and Definition
048, 049	WARNING: msgcnt x: mesg 048: V-2-48: vx_dqread - warning: *mount_point* file system external user|group quota file read failed WARNING: msgcnt x: mesg 049: V-2-49: vx_dqwrite - warning: *mount_point* file system external user|group 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.
055	WARNING: msgcnt x: mesg 055: V-2-55: vx_force_unmount - *mount_point* file system disabled by forced unmount ■ Description blah ■ Action blah

Table B-1 Kernel messages *(continued)*

Message Number	Message and Definition
056	WARNING: msgcnt x: mesg 056: V-2-56: 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 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.
057	WARNING: msgcnt x: mesg 057: V-2-57: 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 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.

Table B-1 Kernel messages *(continued)*

Message Number	Message and Definition
058	WARNING: msgcnt x: mesg 058: V-2-58: 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: mesg 059: V-2-59: 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.

Table B-1 Kernel messages *(continued)*

Message Number	Message and Definition
060	WARNING: msgcnt x: mesg 060: V-2-60: 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.
061	WARNING: msgcnt x: mesg 061: V-2-61: 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: mesg 062: V-2-62: 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.

Table B-1 Kernel messages *(continued)*

Message Number	Message and Definition
063	WARNING: msgcnt x: mesg 063: V-2-63: 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 cannot 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: mesg 064: V-2-64: 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 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 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.
066	NOTICE: msgcnt x: mesg 066: V-2-66: 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.

Table B-1 Kernel messages *(continued)*

Message Number	Message and Definition
067	WARNING: msgcnt x: mesg 067: V-2-67: 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: mesg 068: V-2-68: *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. See "Using kernel tunables" on page 40.
069	WARNING: msgcnt x: mesg 069: V-2-69: 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. See "Using kernel tunables" on page 40.

Table B-1 Kernel messages *(continued)*

Message Number	Message and Definition
070	WARNING: msgcnt x: mesg 070: V-2-70: 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.
071	NOTICE: msgcnt x: mesg 071: V-2-71: 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 while the file system was previously mounted. See Message Number 038. ■ Action Informational only, no action required.
072	WARNING: msgcnt x: vxfs: mesg 072: could not failover for *volume_name* file system ■ Description This message is specific to the cluster file system. The message indicates a problem in a scenario where a node failure has occurred in the cluster and the newly selected primary node encounters a failure. ■ Action Save the system logs and core dump of the node along with the disk image (metasave) and contact your customer support organization. The node can be rebooted to join the cluster.

Table B-1 Kernel messages *(continued)*

Message Number	Message and Definition
075	WARNING: msgcnt x: mesg 075: V-2-75: 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: mesg 076: V-2-76: 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, or simply wait for the operation to complete so file system can be unmounted cleanly. See the umount_vxfs(1M) manual page.
077	WARNING: msgcnt x: mesg 077: V-2-77: 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.

Table B-1 Kernel messages *(continued)*

Message Number	Message and Definition
078	WARNING: msgcnt x: mesg 078: V-2-78: 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.

Table B-1 Kernel messages *(continued)*

Message Number	Message and Definition
079	

Table B-1 Kernel messages *(continued)*

Message Number	Message and Definition
	WARNING: msgcnt x: mesg 017: V-2-79: vx_attr_getblk - *mount_point* file system inode *inumber* marked bad on disk
	WARNING: msgcnt x: mesg 017: V-2-79: vx_attr_iget - *mount_point* file system inode *inumber* marked bad on disk
	WARNING: msgcnt x: mesg 017: V-2-79: vx_attr_indadd - *mount_point* file system inode *inumber* marked bad on disk
	WARNING: msgcnt x: mesg 017: V-2-79: vx_attr_indtrunc - *mount_point* file system inode *inumber* marked bad on disk
	WARNING: msgcnt x: mesg 017: V-2-79: vx_attr_iremove - *mount_point* file system inode *inumber* marked bad on disk
	WARNING: msgcnt x: mesg 017: V-2-79: vx_bmap - *mount_point* file system inode *inumber* marked bad on disk
	WARNING: msgcnt x: mesg 017: V-2-79: vx_bmap_indirect_ext4 - *mount_point* file system inode *inumber* marked bad on disk
	WARNING: msgcnt x: mesg 017: V-2-79: vx_delbuf_flush - *mount_point* file system inode *inumber* marked bad on disk
	WARNING: msgcnt x: mesg 017: V-2-79: vx_dio_iovec - *mount_point* file system inode *inumber* marked bad on disk
	WARNING: msgcnt x: mesg 017: V-2-79: vx_dirbread - *mount_point* file system inode *inumber* marked bad on disk
	WARNING: msgcnt x: mesg 017: V-2-79: vx_dircreate - *mount_point* file system inode *inumber* marked bad on disk
	WARNING: msgcnt x: mesg 017: V-2-79: vx_dirlook - *mount_point* file system inode *inumber* marked bad on disk
	WARNING: msgcnt x: mesg 017: V-2-79: vx_doextop_iau - *mount_point* file system inode *inumber* marked bad on disk
	WARNING: msgcnt x: mesg 017: V-2-79: vx_doextop_now - *mount_point* file system inode *inumber* marked bad on disk
	WARNING: msgcnt x: mesg 017: V-2-79: vx_do_getpage - *mount_point* file system inode *inumber* marked bad on disk
	WARNING: msgcnt x: mesg 017: V-2-79: vx_enter_ext4 - *mount_point* file system inode *inumber* marked bad on disk
	WARNING: msgcnt x: mesg 017: V-2-79: vx_exttrunc - *mount_point* file system inode *inumber* marked bad on disk

Table B-1 Kernel messages *(continued)*

Message Number	Message and Definition
	WARNING: msgcnt x: mesg 017: V-2-79: vx_get_alloc - *mount_point* file system inode *inumber* marked bad on disk
079 (continued)	WARNING: msgcnt x: mesg 017: V-2-79: vx_ilisterr - *mount_point* file system inode *inumber* marked bad on disk WARNING: msgcnt x: mesg 017: V-2-79: vx_indtrunc - *mount_point* file system inode *inumber* marked bad on disk WARNING: msgcnt x: mesg 017: V-2-79: vx_iread - *mount_point* file system inode *inumber* marked bad on disk WARNING: msgcnt x: mesg 017: V-2-79: vx_iremove - *mount_point* file system inode *inumber* marked bad on disk WARNING: msgcnt x: mesg 017: V-2-79: vx_iremove_attr - *mount_point* file system inode *inumber* marked bad on disk WARNING: msgcnt x: mesg 017: V-2-79: vx_logwrite_flush - *mount_point* file system inode *inumber* marked bad on disk WARNING: msgcnt x: mesg 017: V-2-79: vx_oltmount_iget - *mount_point* file system inode *inumber* marked bad on disk WARNING: msgcnt x: mesg 017: V-2-79: vx_overlay_bmap - *mount_point* file system inode *inumber* marked bad on disk WARNING: msgcnt x: mesg 017: V-2-79: vx_readnomap - *mount_point* file system inode *inumber* marked bad on disk WARNING: msgcnt x: mesg 017: V-2-79: vx_reorg_trunc - *mount_point* file system inode *inumber* marked bad on disk WARNING: msgcnt x: mesg 017: V-2-79: vx_stablestore - *mount_point* file system inode *inumber* marked bad on disk WARNING: msgcnt x: mesg 017: V-2-79: vx_tranitimes - *mount_point* file system inode *inumber* marked bad on disk WARNING: msgcnt x: mesg 017: V-2-79: vx_trunc - *mount_point* file system inode *inumber* marked bad on disk WARNING: msgcnt x: mesg 017: V-2-79: vx_write_alloc2 - *mount_point* file system inode *inumber* marked bad on disk WARNING: msgcnt x: mesg 017: V-2-79: vx_write_default - *mount_point* file system inode *inumber* marked bad on disk WARNING: msgcnt x: mesg 017: V-2-79: vx_zero_alloc - *mount_point* file system inode *inumber* marked bad on disk

Table B-1 Kernel messages *(continued)*

Message Number	Message and Definition
079 (continued)	■ 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.

Table B-1 Kernel messages *(continued)*

Message Number	Message and Definition
080	WARNING: msgcnt x: mesg 080: V-2-80: Disk layout versions older than Version 4 will not be supported in the next release. It is advisable to upgrade to the latest disk layout version now. See the vxupgrade(1M) manual page. See the *Veritas Storage Foundation Release Notes*. ■ Action Use the `vxupgrade` command to upgrade file systems using older disk layouts to Version 5, then 6, then 7. Consider the following when planning disk layout upgrades: ■ Version 1 disk layout file systems can support more than 8 million inodes, while Version 2 disk layout file systems have an 8 million inode limit. The Version 1 disk layout provides finer control of disk geometry than subsequent disk layouts. This finer control is not relevant on disks employing newer technologies, but can still be applicable on older hardware. If you are using Version 1 disk layout file systems on older hardware that needs fine control of disk geometry, a disk layout upgrade may be problematic. Images of Version 1 or Version 2 disk layout file systems created by copy utilities, such as `dd` or `volcopy`, will become unusable after a disk layout upgrade.
081	WARNING: msgcnt x: mesg 081: V-2-81: possible network partition detected ■ Description This message displays when CFS detects a possible network partition and disables the file system locally, that is, on the node where the message appears. ■ Action There are one or more private network links for communication between the nodes in a cluster. At least one link must be active to maintain the integrity of the cluster. If all the links go down, after the last network link is broken, the node can no longer communicate with other nodes in the cluster. Check the network connections. After verifying that the network connections is operating correctly, unmount the disabled file system and mount it again.

Table B-1 Kernel messages *(continued)*

Message Number	Message and Definition
082	WARNING: msgcnt x: mesg 082: V-2-82: *volume_name* file system is on shared volume. It may get damaged if cluster is in partitioned state. ■ Description If a cluster node is in a partitioned state, and if the file system is on a shared VxVM volume, this volume may become corrupted by accidental access from another node in the cluster. ■ Action These shared disks can also be seen by nodes in a different partition, so they can inadvertently be corrupted. So the second message 082 tells that the device mentioned is on shared volume and damage can happen only if it is a real partition problem. Do not use it on any other node until the file system is unmounted from the mounted nodes.
083	WARNING: msgcnt x: mesg 083: V-2-83: *mount_point* file system log is not compatible with the specified intent log I/O size ■ Description Either the specified `mount` logiosize size is not compatible with the file system layout, or the file system is corrupted. ■ Action Mount the file system again without specifying the logiosize option, or use a logiosize value compatible with the intent log specified when the file system was created. If the error persists, unmount the file system and use `fsck` to run a full structural check.
084	WARNING: msgcnt x: mesg 084: V-2-84: in *volume_name* quota on failed during assumption. (stage *stage_number*) ■ Description In a cluster file system, when the primary of the file system fails, a secondary file system is chosen to assume the role of the primary. The assuming node will be able to enforce quotas after becoming the primary. If the new primary is unable to enforce quotas this message will be displayed. ■ Action Issue the quotaon command from any of the nodes that have the file system mounted.

Table B-1 Kernel messages *(continued)*

Message Number	Message and Definition
085	WARNING: msgcnt x: mesg 085: V-2-85: Checkpoint quota - warning: *file_system* file system fileset quota hard limit exceeded ■ Description The system administrator sets the quotas for Storage Checkpoints in the form of a soft limit and hard limit. This message displays when the hard limit is exceeded. ■ Action Delete Storage Checkpoints or increase the hard limit.
086	WARNING: msgcnt x: mesg 086: V-2-86: Checkpoint quota - warning: *file_system* file system fileset quota soft limit exceeded ■ Description The system administrator sets the quotas for Storage Checkpoints in the form of a soft limit and hard limit. This message displays when the soft limit is exceeded. ■ Action Delete Storage Checkpoints or increase the soft limit. This is not a mandatory action, but is recommended.
087	WARNING: msgcnt x: mesg 087: V-2-87: vx_dotdot_manipulate: *file_system* file system *inumber* inode ddnumber dotdot inode error ■ Description When performing an operation that changes an inode entry, if the inode is incorrect, this message will display. ■ Action Run a full file system check using `fsck` to correct the errors.
088	WARNING: msgcnt x: mesg 088: V-2-88: quotaon on *file_system* failed; limits exceed limit ■ Description The external quota file, quotas, contains the quota values, which range from 0 up to 2147483647. When quotas are turned on by the `quotaon` command, this message displays when a user exceeds the quota limit. ■ Action Correct the quota values in the quotas file.

Table B-1 Kernel messages *(continued)*

Message Number	Message and Definition
089	WARNING: msgcnt x: mesg 089: V-2-89: quotaon on *file_system* invalid; disk usage for group/user id *uid* exceeds sectors sectors ■ Description The supported quota limit is up to 2147483647 sectors. When quotas are turned on by the `quotaon` command, this message displays when a user exceeds the supported quota limit. ■ Action Ask the user to delete files to lower the quota below the limit.
090	WARNING: msgcnt x: mesg 090: V-2-90: quota on *file_system* failed; soft limits greater than hard limits ■ Description One or more users or groups has a soft limit set greater than the hard limit, preventing the BSD quota from being turned on. ■ Action Check the soft limit and hard limit for every user and group and confirm that the soft limit is not set greater than the hard limit.
091	WARNING: msgcnt x: mesg 091: V-2-91: vx_fcl_truncate - failure to punch hole at offset *offset* for *bytes* bytes in File Change Log file; error *error_number* ■ Description The vxfs kernel has experienced an error while trying to manage the space consumed by the File Change Log file. Because the space cannot be actively managed at this time, the FCL has been deactivated and has been truncated to 1 file system block, which contains the FCL superblock. ■ Action Re-activate the FCL.
092	WARNING: msgcnt x: mesg 092: V-2-92: vx_mkfcltran - failure to map offset *offset* in File Change Log file ■ Description The vxfs kernel was unable to map actual storage to the next offset in the File Change Log file. This is mostly likely caused by a problem with allocating to the FCL file. Because no new FCL records can be written to the FCL file, the FCL has been deactivated. ■ Action Re-activate the FCL.

Table B-1 Kernel messages *(continued)*

Message Number	Message and Definition
096	WARNING: msgcnt x: mesg 096: V-2-96: *file_system* file system fullfsck flag set - *function_name*. ■ Description The next time the file system is mounted, a full `fsck` must be performed. ■ Action No immediate action required. When the file system is unmounted, run a full file system check using `fsck` before mounting it again.
097	WARNING: msgcnt x: mesg 097: V-2-97: VxFS failed to create new thread (*error_number*, *function_address*:*argument_address*) ■ Description VxFS failed to create a kernel thread due to resource constraints, which is often a memory shortage. ■ Action VxFS will retry the thread creation until it succeeds; no immediate action is required. Kernel resources, such as kernel memory, might be overcommitted. If so, reconfigure the system accordingly.
098	WARNING: msgcnt x: mesg 098: V-2-98: VxFS failed to initialize File Change Log for fileset fileset (index number) of *mount_point* file system ■ Description VxFS mount failed to initialize FCL structures for the current fileset mount. As a result, FCL could not be turned on. The FCL file will have no logging records. ■ Action Reactivate the FCL.
099	WARNING: msgcnt x: mesg 099: V-2-99: The specified value for *vx_ninode* is less than the recommended minimum value of *min_value* ■ Description Auto-tuning or the value specified by the system administrator resulted in a value lower than the recommended minimum for the total number of inodes that can be present in the inode cache. VxFS will ignore the newly tuned value and will keep the value specified in the message (`VX_MINNINODE`). ■ Action Informational only; no action required.

Table B-1 Kernel messages *(continued)*

Message Number	Message and Definition
101	WARNING: msgcnt x: mesg 101: V-2-101: File Change Log on *mount_point* for file set *index* approaching max file size supported. File Change Log will be reactivated when its size hits max file size supported. ■ Description The size of the FCL file is approching the maximum file size supported. This size is platform specific. When the FCL file is reaches the maximum file size, the FCL will be deactivated and reactivated. All logging information gathered so far will be lost. ■ Action Take any corrective action possible to restrict the loss due to the FCL being deactivated and reactivated.
102	WARNING: msgcnt x: mesg 102: V-2-102: File Change Log of *mount_point* for file set *index* has been reactivated. ■ Description The size of FCL file reached the maximum supported file size and the FCL has been reactivated. All records stored in the FCL file, starting from the current *fc_loff* up to the maximum file size, have been purged. New records will be recorded in the FCL file starting from offset *fs_bsize*. The activation time in the FCL is reset to the time of reactivation. The impact is equivalent to File Change Log being deactivated and activated. ■ Action Informational only; no action required.
103	WARNING: msgcnt x: mesg 103: V-2-103: File Change Log merge on *mount_point* for file set *index* failed. ■ Description The VxFS kernel has experienced an error while merging internal per-node File Change Log files into the external File Change Log file. Since the File Change Log cannot be maintained correctly without this, the File Change Log has been deactivated. ■ Action Re-activate the File Change Log.

Table B-1 Kernel messages *(continued)*

Message Number	Message and Definition
104	WARNING: msgcnt x: mesg 104: V-2-104: File System *mount_point* device *volume_name* disabled ■ Description The volume manager detected that the specified volume has failed, and the volume manager has disabled the volume. No further I/O requests are sent to the disabled volume. ■ Action The volume must be repaired.
105	WARNING: msgcnt x: mesg 105: V-2-105: File System *mount_point* device *volume_name* re-enabled ■ Description The volume manager detected that a previously disabled volume is now operational, and the volume manager has re-enabled the volume. ■ Action Informational only; no action required.
106	WARNING: msgcnt x: mesg 106: V-2-106: File System *mount_point* device *volume_name* has BAD label ■ Description A file system's label does not match the label that the multi-volume support feature expects the file system to have. The file system's volume is effectively disabled. ■ Action If the label is bad because the volume does not match the assigned label, use the `vxvset` command to fix the label. Otherwise, the label might have been overwritten and the volume's contents may be lost. Call technical support so that the issue can be investigated.
107	WARNING: msgcnt x: mesg 107: V-2-107: File System *mount_point* device *volume_name* valid label found ■ Description The label of a file system that had a bad label was somehow restored. The underlying volume is functional. ■ Action Informational only; no action required.

Table B-1 Kernel messages *(continued)*

Message Number	Message and Definition
108	WARNING: msgcnt x: mesg 108: V-2-108: vx_dexh_error - *error*: fileset *fileset*, directory inode number *dir_inumber*, bad hash inode *hash_inode*, seg *segment* bno *block_number* ■ Description The supplemental hash for a directory is corrupt. ■ Action If the file system is mounted read/write, the hash for the directory will be automatically removed and recreated. If the removal or recreation fails, subsequent messages indicate the type of prolem. If there are no further messages, the removal and recreation of the hash succeeded.
109	WARNING: msgcnt x: mesg 109: V-2-109: failed to tune down *tuneable* to *value* possibly due to *object* in use, could free up only up to *number_of_inodes* ■ Description The number of inodes in the inode table could not be reduced to the decreased value of the tuneable. This may have occurred because objects are in use, in which case the tuneable was not changed. ■ Action To decrease the tuneable value and the number of inodes, specify the `-h` option of the `kctune` command so that the new tuneable value takes effect after a system reboot.
110	WARNING: msgcnt x: mesg 110: V-2-110: The specified value for vx_bc_buffhwm is less than the recommended minimum value of *value*. ■ Description The *vx_bc_buffhwm* dynamic tuneable should not have its value set below a minimum value. ■ Action Set the value of *vx_bc_buffhwm* to greater than the recommended minimum.

Table B-1 Kernel messages *(continued)*

Message Number	Message and Definition
111	WARNING: msgcnt x: mesg 111: V-2-111: You have exceeded the authorized usage (maximum `maxfs` unique mounted user-data file systems) for this product and are out of compliance with your License Agreement. Please email **sales_mail@symantec.com** or contact your Symantec sales representative for information on how to obtain additional licenses for this product. ■ Description As per your Storage Foundation Basic license agreement, you are allowed to have only a limited number of VxFS file systems, and you have exceeded this number. ■ Action Email **sales_mail@symantec.com** or contact your Symantec sales representative for information on how to obtain additional licenses for this product.

About unique message identifiers

VxFS generates diagnostic or error messages for issues not related to the kernel, which are displayed along with a unique message identifier (UMI). Each message has a description and a suggestion on how to handle or correct the underlying problem. The UMI is used to identify the issue should you need to call Technical Support for assistance.

Unique message identifiers

Some commonly encountered UMIs and the associated messages are described on the following table:

Table B-2 Unique message identifiers and messages

Message Number	Message and Definition
20002	UX:vxfs command: ERROR: V-3-20002: *message* ■ Description The command attempted to call `stat`() on a device path to ensure that the path refers to a character device before opening the device, but the `stat`() call failed. The error message will include the platform-specific message for the particular error that was encountered, such as "Access denied" or "No such file or directory". ■ Action The corrective action depends on the particular error.
20003	UX:vxfs command: ERROR: V-3-20003: *message* ■ Description The command attempted to open a disk device, but the `open`() call failed. The error message includes the platform-specific message for the particular error that was encountered, such as "Access denied" or "No such file or directory". ■ Action The corrective action depends on the particular error.
20005	UX:vxfs command: ERROR: V-3-20005: *message* ■ Description The command attempted to read the superblock from a device, but the `read`() call failed. The error message will include the platform-specific message for the particular error that was encountered, such as "Access denied" or "No such file or directory". ■ Action The corrective action depends on the particular error.
20012	UX:vxfs command: ERROR: V-3-20012: *message* ■ Description The command was invoked on a device that did not contain a valid VxFS file system. ■ Action Check that the path specified is what was intended.

Table B-2 Unique message identifiers and messages *(continued)*

Message Number	Message and Definition
20076	UX:vxfs command: ERROR: V-3-20076: *message* ■ Description The command called `stat`() on a file, which is usually a file system mount point, but the call failed. ■ Action Check that the path specified is what was intended and that the user has permission to access that path.
21256	UX:vxfs command: ERROR: V-3-21256: *message* ■ Description The attempt to mount the file system failed because either the request was to mount a particular Storage Checkpoint that does not exist, or the file system is managed by an HSM and the HSM is not running. ■ Action In the first case, use the `fsckptadm list` command to see which Storage Checkpoints exist and mount the appropriate Storage Checkpoint. In the second case, make sure the HSM is running. If the HSM is not running, start and mount the file system again.

Table B-2 Unique message identifiers and messages *(continued)*

Message Number	Message and Definition
21264	UX:vxfs command: ERROR: V-3-21264: *message* ■ Description The attempt to mount a VxFS file system has failed because either the volume being mounted or the directory which is to be the mount point is busy. The reason that a VxVM volume could be busy is if the volume is in a shared disk group and the volume is currently being accessed by a VxFS command, such as `fsck`, on a node in the cluster. One reason that the mount point could be busy is if a process has the directory open or has the directory as its current directory. Another reason that the mount point could be busy is if the directory is NFS-exported. ■ Action For a busy mount point, if a process has the directory open or has the directory as its current directory, use the `fuser` command to locate the processes and either get them to release their references to the directory or kill the processes. Afterward, attempt to mount the file system again. If the directory is NFS-exported, unexport the directory, such as by using `unshare` mntpt on the Solaris operating system. Afterward, attempt to mount the file system again.
21268	UX:vxfs command: ERROR: V-3-21268: *message* ■ Description This message is printed by two different commands: `fsckpt_restore` and `mount`. In both cases, the kernel's attempt to mount the file system failed because of I/O errors or corruption of the VxFS metadata. ■ Action Check the console log for I/O errors and fix any problems reported there. Run a full `fsck`.

Table B-2 Unique message identifiers and messages *(continued)*

Message Number	Message and Definition
21272	UX:vxfs command: ERROR: V-3-21272: *message* ■ Description The mount options specified contain mutually-exclusive options, or in the case of a remount, the new mount options differed from the existing mount options in a way that is not allowed to change in a remount. ■ Action Change the requested mount options so that they are all mutually compatible and retry the mount.
23729	UX:vxfs command: ERROR: V-3-23729: *message* ■ Description Cluster mounts require the vxfsckd daemon to be running, which is controlled by VCS. ■ Action Check the VCS status to see why this service is not running. After starting the daemon via VCS, try the mount again.
24996	UX:vxfs command: ERROR: V-3-24996: *message* ■ Description In some releases of VxFS, before the VxFS `mount` command attempts to mount a file system, `mount` tries to read the VxFS superblock to determine the disk layout version of the file system being mounted so that `mount` can check if that disk layout version is supported by the installed release of VxFS. If the attempt to read the superblock fails for any reason, this message is displayed. This message will usually be preceded by another error message that gives more information as to why the superblock could not be read. ■ Action The corrective action depends on the preceding error, if any.

Appendix C

Disk layout

This appendix includes the following topics:

About disk layouts

The disk layout is the way file system information is stored on disk. On VxFS, seven different disk layout versions were created to take advantage of evolving technological developments.

The disk layout versions used on VxFS are:

Version 1	Version 1 disk layout is the original VxFS disk layout provided with pre-2.0 versions of VxFS.	Not Supported
Version 2	Version 2 disk layout supports features such as filesets, dynamic inode allocation, and enhanced security. The Version 2 layout is available with and without quotas support.	Not Supported

Version 3	Version 3 disk layout encompasses all file system structural information in files, rather than at fixed locations on disk, allowing for greater scalability. Version 3 supports files and file systems up to one terabyte in size.	Not Supported
Version 4	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 and file systems up to one terabyte in size.	Supported
Version 5	Version 5 enables the creation of file system sizes up to 32 terabytes. Files can be a maximum of one terabyte. File systems larger than 1TB must be created on a Veritas Volume Manager volume.	Supported
Version 6	Version 6 disk layout enables features such as multi-volume support, cross-platform data sharing, named data streams, and File Change Log.	Supported
Version 7	Version 7 disk layout enables support for variable and large size history log records, more than 2048 volumes, large directory hash, and Dynamic Storage Tiering.	Supported

Some of the disk layout versions were not supported on all UNIX operating systems. Currently, only the Version 4, 5, 6, and 7 disk layouts can be created and mounted. Version 1 and 2 file systems cannot be created nor mounted. Version 7 is the default disk layout version.

The `vxupgrade` command is provided to upgrade an existing VxFS file system to the Version 5, 6, or 7 layout while the file system remains online. You must upgrade in steps from older to newer layouts.

See the vxupgrade(1M) manual page.

The `vxfsconvert` command is provided to upgrade Version 1 and 2 disk layouts to the Version 5 disk layout while the file system is not mounted.

See the vxfsconvert(1M) manual page.

About 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 for file systems less than one terabyte 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.

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 original 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.

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 previous layouts.

All Version 4 structural files reside in the structural fileset.

The structural files in the Version 4 disk layout are:

object location table file	Contains the object location table (OLT). 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 files	Contains quota information in records. Each record contains resources allocated either per user or per group.

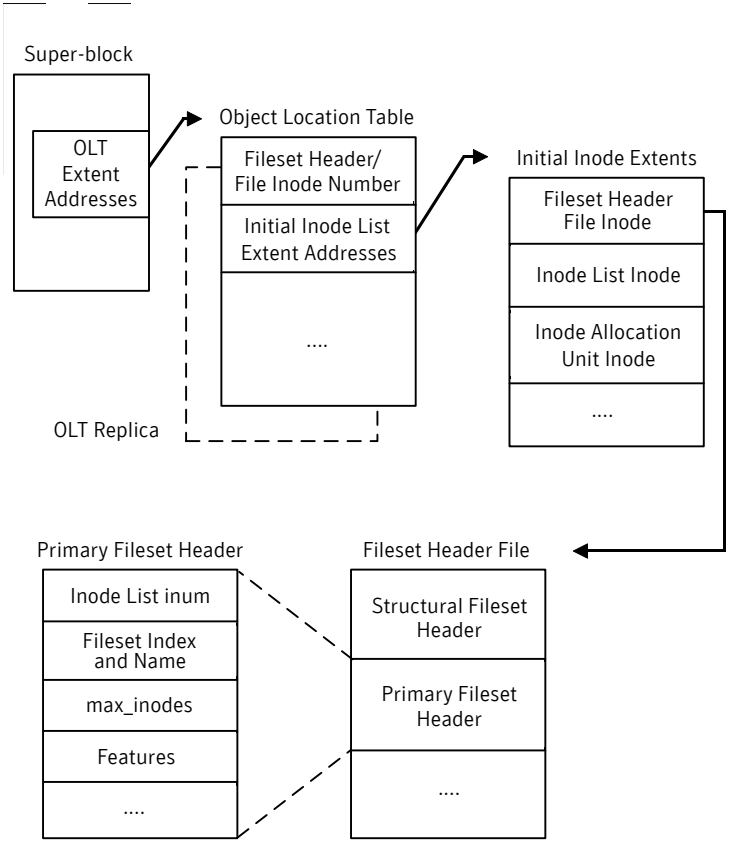
The Version 4 disk layout supports Access Control Lists and Block-Level Incremental (BLI) Backup. BLI Backup is a backup method that stores and retrieves only the data blocks changed since the previous backup, not entire files. This saves times, storage space, and computing resources required to backup large databases.

Figure C-1 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 C-1 VxFS Version 4 disk layout



VxFS Version 5 disk layout

VxFS disk layout Version 5 is similar to Version 4. Structural files in Version 5 are the same in Version 4. However, the Version 5 disk layout supports file systems up to 32 terabytes. For a file system to take advantage of VxFS 32-terabyte support, it must be created on a Veritas Volume Manager volume, and only on a 64-bit kernel operating system. The maximum file system size on a 32-bit kernel is still

one terabyte. Files cannot exceed two terabytes in size. For 64-bit kernels, the maximum size of the file system you can create depends on the block size:

Block Size	Maximum File System Size
1024 bytes	4,294,967,039 sectors (≈ 4 TB)
2048 bytes	8,589,934,078 sectors (≈ 8 TB)
4096 bytes	17,179,868,156 sectors (≈ 16 TB)
8192 bytes	34,359,736,312 sectors (≈ 32 TB)

If you specify the file system size when creating a file system, the block size defaults to the appropriate value as shown above.

See the mkfs(1M) manual page.

The Version 5 disk layout also supports group quotas. Quota limits cannot exceed one terabyte.

See "About quota files on Veritas File System" on page 102.

Some UNIX commands may not work correctly on file systems larger than one terabyte.

See "Using UNIX Commands on File Systems Larger than One TB" on page 281.

VxFS Version 6 disk layout

VxFS disk layout Version 6 is similar to Version 5. Structural files in Version 6 are the same in Version 5. The Version 6 disk layout can theoretically support files and file systems up to 8 exabytes (2^{63}). The maximum file system size that can be created is currently restricted to 2^{35} blocks. For a file system to take advantage of greater than 1 terabyte support, it must be created on a Veritas Volume Manager volume. For 64-bit kernels, the maximum size of the file system you can create depends on the block size:

Block Size	Currently-Supported Maximum File System Size
1024 bytes	68,719,472,624 sectors (≈ 32 TB)
2048 bytes	137,438,945,248 sectors (≈ 64 TB)
4096 bytes	274,877,890,496 sectors (≈ 128 TB)
8192 bytes	549,755,780,992 sectors (≈ 256 TB)

The Version 6 disk layout also supports group quotas.

See "About quota files on Veritas File System" on page 102.

Some UNIX commands may not work correctly on file systems larger than one terabyte.

See "Using UNIX Commands on File Systems Larger than One TB" on page 281.

VxFS Version 7 disk layout

VxFS disk layout Version 7 is similar to Version 6. The Version 7 disk layout can theoretically support files and file systems up to 8 exabytes (2^{63}). The maximum file system size that can be created is currently restricted to 2^{35} blocks. For a file system to take advantage of greater than 1 terabyte support, it must be created on a Veritas Volume Manager volume. For 64-bit kernels, the maximum size of the file system you can create depends on the block size:

The Version 7 disk layout supports group quotas.

See "About quota files on Veritas File System" on page 102.

Some UNIX commands may not work correctly on file systems larger than one terabyte.

See "Using UNIX Commands on File Systems Larger than One TB" on page 281.

Using UNIX Commands on File Systems Larger than One TB

Some UNIX commands may not work correctly on file systems larger than one terabyte.

The `ustat` command returns an `EOVERFLOW` error for VxFS files systems larger than one terabyte because the variable used to store file system size overflows.

See the ustat(2) manual page.

System administration utilities such as backup may not operate correctly if they are not large file aware (files larger than two gigabytes). Similarly, utilities that operate at the file system level must be large file aware to operate correctly on large file systems (file systems that are larger than one terabyte). Note also that you can have a large file system without creating the file system with the `mkfs -o largefiles` option.

See the lfcompile(5) manual page.

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.
API	Application Programming Interface.
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 Symantec 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.
contiguous file	A file in which data blocks are physically adjacent on the underlying media.
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/filesystems 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 fragment size on some UNIX file systems.
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.
fragmentation	The on-going process on an active file system in which the file system is spread further and further along the disk, leaving unused gaps or fragments between areas that are in use. This leads to degraded performance because the file system has fewer options when assigning a file to an extent.
GB	Gigabyte (230 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 indices used by each user. See quotas and external quotas file.
K	Kilobyte (210 bytes or 1024 bytes).
large file	A file larger than two one terabyte. VxFS supports files up to 8 exabytes in size.
large file system	A file system larger than one terabytes. VxFS supports file systems up to 8 exabytes 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 (220 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.
multi-volume file system	A single file system that has been created over multiple volumes, with each volume having its own properties.
MVS	Multi-volume support.
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.
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 quotas.
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 Storage 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 (240 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.
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.

volume A virtual disk which represents an addressable range of disk blocks used by applications such as file systems or databases.

volume set A container for multiple different volumes. Each volume can have its own geometry.

vxfs The Veritas File System type. Used as a parameter in some commands.

VxFS Veritas File System.

VxVM Veritas Volume Manager.

Index

E

F

G

H

I

K

L

M

N

O

P

Q

R

S

T

U

V

W