

VERITAS File System™ 3.5

Administrator's Guide

Solaris

August 2002
N08842F


VERITAS

Disclaimer

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

Copyright

Copyright © 2002 VERITAS Software Corporation. All rights reserved. VERITAS, VERITAS Software, the VERITAS logo, VERITAS File System, and all other VERITAS product names and slogans are trademarks or registered trademarks of VERITAS Software Corporation in the USA and/or other countries. Other product names mentioned herein may be trademarks or registered trademarks of their respective companies.

August 2002.

VERITAS Software Corporation
350 Ellis Street
Mountain View, CA 94043
USA
Phone 650-527-8000
Fax 650-527-2908
<http://www.veritas.com>



Contents

Preface	xiii
Introduction	xiii
Organization	xiv
Conventions	xv
Related Documents	xvi
Getting Help	xvi
 Chapter 1. The VERITAS File System	1
Introduction	1
VxFS Features	2
Disk Layouts	3
File System Performance Enhancements	3
VERITAS Enterprise Administrator Graphical User Interface	4
Extent Based Allocation	5
Typed Extents	6
Extent Attributes	7
Fast File System Recovery	7
Online System Administration	8
Defragmentation	8
Resizing	8
Application Interface	9
Application Transparency	9
Expanded Application Facilities	9
Extended mount Options	10



Enhanced Data Integrity Modes	10
Using blkclear Option for Data Integrity	10
Using closesync Option for Data Integrity	10
Using the log Option for Data Integrity	11
Enhanced Performance Mode	11
Using the delaylog Option for Enhanced Performance	11
Using the qlog Option for Enhanced Performance	11
Temporary File System Modes	12
Using the tmplog option For Temporary File Systems	12
Improved Synchronous Writes	12
Support for Large Files	12
Enhanced I/O Performance	13
Enhanced I/O Clustering	13
VxVM Integration	13
Application-Specific Parameters	13
Access Control Lists	14
Storage Checkpoints	14
Online Backup	14
Quotas	15
Support for Databases	15
VERITAS QuickLog	16
Cluster File Systems	16

Chapter 2. VxFS Performance: Creating, Mounting, and Tuning File Systems ... 17

Introduction	17
Choosing mkfs Command Options	18
Block Size	18
Intent Log Size	19
Choosing mount Command Options	19
log	20



delaylog	20
tmplog	20
logiosize	20
nodatainlog	21
blkclear	21
mincache	21
convosync	23
qlog	24
largefiles nolargefiles	24
Creating a File System with Large Files	24
Mounting a File System with Large Files	25
Managing a File System with Large Files	25
Combining mount Command Options	26
Example 1 - Desktop File System	26
Example 2 - Temporary File System or Restoring from Backup	26
Example 3 - Data Synchronous Writes	26
Kernel Tunables	27
Internal Inode Table Size	27
vx_maxlink	27
VxVM Maximum I/O Size	28
vol_maxio	28
Monitoring Free Space	28
Monitoring Fragmentation	29
I/O Tuning	30
Tuning VxFS I/O Parameters	30
Tunable VxFS I/O Parameters	32
Chapter 3. Extent Attributes	37
Introduction	37
Attribute Specifics	38



Reservation: Preallocating Space to a File	39
Fixed Extent Size	39
Other Controls	40
Alignment	40
Contiguity	40
Write Operations Beyond Reservation	40
Reservation Trimming	40
Reservation Persistence	41
Including Reservation in the File	41
Commands Related to Extent Attributes	41
Failure to Preserve Extent Attributes	42

Chapter 4. Application Interface 43

Introduction	43
Cache Advisories	44
Direct I/O	44
Unbuffered I/O	45
Discovered Direct I/O	45
Data Synchronous I/O	45
Other Advisories	46
Extent Information	46
Space Reservation	47
Fixed Extent Sizes	49
Freeze and Thaw	50
Get I/O Parameters ioctl	50

Chapter 5. Storage Checkpoints 51

What is a Storage Checkpoint?	52
How a Storage Checkpoint Works	53
Types of Storage Checkpoints	56
Data Storage Checkpoints	56



Nodata Storage Checkpoints	56
Removable Storage Checkpoints	56
Non-mountable Storage Checkpoints	57
Storage Checkpoint Administration	57
Creating a Storage Checkpoint	57
Removing a Storage Checkpoint	58
Accessing a Storage Checkpoint	59
Converting a Data Storage Checkpoint to a Nodata Storage Checkpoint	61
Difference Between a Data and a Nodata Storage Checkpoint	62
Conversion with Multiple Storage Checkpoints	64
Space Management Considerations	68
Chapter 6. Online Backup	69
Introduction	69
Snapshot File Systems	70
Using a Snapshot File System for Backup	70
Creating a Snapshot File System	71
Making a Backup	72
Performance of Snapshot File Systems	72
Differences Between Snapshots and Storage Checkpoints	73
Snapshot File System Internals	74
Snapshot File System Disk Structure	74
How a Snapshot File System Works	75
Chapter 7. Quotas	77
Introduction	77
Quota Limits	78
Quota Files on VxFS	78
Quota Commands	79
Quota Checking With VxFS	80
Using Quotas	80



vxquotaon	80
mount	81
vxedquota	81
vxquota	82
vxquot	82
vxquotaoff	82
Chapter 8. Quick I/O for Databases	83
Introduction	83
Quick I/O Functionality and Performance	84
Supporting Kernel Asynchronous I/O	84
Supporting Direct I/O	84
Avoiding Kernel Write Locks	84
Avoiding Double Buffering	85
Using VxFS Files as Raw Character Devices	85
Quick I/O Naming Convention	85
Use Restrictions	86
Creating a Quick I/O File Using qiomkfile	86
Accessing Regular VxFS Files Through Symbolic Links	88
Using Absolute or Relative Path Names	88
Preallocating Files Using the setext Command	89
Using Quick I/O with Oracle Databases	89
Using Quick I/O with Sybase Databases	90
Enabling and Disabling Quick I/O	91
Cached Quick I/O For Databases	91
Enabling Cached Quick I/O	92
Enabling Cached Quick I/O for File Systems	92
Enabling Cached Quick I/O for Individual Files	93
Tuning Cached Quick I/O	94
Quick I/O Statistics	94

Quick I/O Summary	94
Chapter 9. VERITAS QuickLog	95
Introduction	95
VERITAS QuickLog Overview	96
QuickLog Setup	96
Creating a QuickLog Device	98
Removing a QuickLog Device	99
VxFS Administration Using QuickLog	99
Enabling a QuickLog Device	99
Disabling a QuickLog Device	100
QuickLog Administration and Troubleshooting	100
QuickLog Load Balancing	100
QuickLog Statistics	101
QuickLog Recovery	102
Cluster QuickLog Devices	102
Appendix A. VERITAS File System Quick Reference	103
Introduction	103
Creating a File System	104
How to Create a File System	104
Mounting a File System	106
How to Mount a File System	106
Mount Options	107
How to Edit the vfstab File	108
Unmounting a File System	110
How to Unmount a File System	110
Displaying Information on Mounted File Systems	111
How to Display File System Information	111
Identifying File System Types	112
How to Identify a File System	112



Resizing a File System	113
How to Extend a File System Using fsadm	113
How to Shrink a File System	114
How to Reorganize a File System	115
Backing Up and Restoring a File System	116
How to Create and Mount a Snapshot File System	116
How to Back Up a File System	117
How to Restore a File System	117
Using Quotas	118
How to Turn On Quotas	118
How to Set Up User Quotas	119
How to View Quotas	120
How to Turn Off Quotas	120
Appendix B. Kernel Messages	121
Introduction	121
File System Response to Problems	122
Marking an Inode Bad	122
Disabling Transactions	122
Disabling a File System	122
Recovering a Disabled File System	123
Kernel Messages	123
Global Message IDs	123
Appendix C. Disk Layout	157
Introduction	157
Disk Space Allocation	158
The VxFS Version 4 Disk Layout	158
The VxFS Version 5 Disk Layout	162
Using UNIX Commands on File Systems Larger than One TB	162

Glossary163

Index171





Preface

Introduction

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

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



Organization

Chapter 1, “[The VERITAS File System](#),” introduces the major features and characteristics of VxFS.

Chapter 2, “[VxFS Performance: Creating, Mounting, and Tuning File Systems](#),” describes VxFS tools that optimize system performance. This section includes information on mount options.

Chapter 3, “[Extent Attributes](#),” describes the policies associated with allocation of disk space.

Chapter 4, “[Application Interface](#),” describes ways to optimize an application for use with VxFS. This chapter includes details on cache advisories, extent sizes, and reservation of file space.

Chapter 5, “[Storage Checkpoints](#),” describes the VxFS replication technology that allows the quick and easy creation of resource-efficient file system backups.

Chapter 6, “[Online Backup](#),” describes the snapshot backup feature of VxFS.

Chapter 7, “[Quotas](#),” describes VxFS methods to limit user access to file and data resources.

Chapter 8, “[Quick I/O for Databases](#),” describes the VERITAS Quick I/O™ feature that treats preallocated files as raw character devices to increase performance.

Chapter 9, “[VERITAS QuickLog](#),” describes the optional VERITAS QuickLog™ product that improves the performance of intent log writes.

Appendix A, “[VERITAS File System Quick Reference](#),” provides information on common file system tasks and examples of typical VxFS operations.

Appendix B, “[Kernel Messages](#),” lists VxFS kernel error messages in numerical order and provides explanations and suggestions for dealing with these problems.

Appendix C, “[Disk Layout](#),” describes and illustrates the major components of VxFS disk layouts.

The “[Glossary](#)” contains a list of terms and definitions relevant to VxFS.

Conventions

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

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



Related Documents

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

The *VERITAS SANPoint Foundation Suite Installation and Configuration Guide* provides information on configuring a cluster and using cluster file systems.

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

Getting Help

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

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

For license information:

- ◆ Phone: 1-925-931-2464
- ◆ Email: license@veritas.com
- ◆ Fax: 1-925-931-2487

For software updates:

- ◆ Email: swupdate@veritas.com

For information on purchasing VERITAS products:

- ◆ Phone: 1-800-258-UNIX (1-800-258-8649) or 1-650-527-8000
- ◆ Email: vx-sales@veritas.com

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

<http://www.veritas.com>

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

<http://support.veritas.com>

Introduction

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

This chapter provides an overview of major VxFS features that are described in detail in later chapters. The following topics are introduced in this chapter:

- ◆ [VxFS Features](#)
- ◆ [Disk Layouts](#)
- ◆ [File System Performance Enhancements](#)
- ◆ [VERITAS Enterprise Administrator Graphical User Interface](#)
- ◆ [Extent Based Allocation](#)
- ◆ [Extent Attributes](#)
- ◆ [Fast File System Recovery](#)
- ◆ [Online System Administration](#)
- ◆ [Application Interface](#)
- ◆ [Extended mount Options](#)
- ◆ [Enhanced I/O Performance](#)
- ◆ [Access Control Lists](#)
- ◆ [Storage Checkpoints](#)
- ◆ [Online Backup](#)
- ◆ [Quotas](#)
- ◆ [Support for Databases](#)
- ◆ [VERITAS QuickLog](#)
- ◆ [Cluster File Systems](#)



VxFS Features

Basic features include:

- ◆ Extent based allocation
- ◆ Extent attributes
- ◆ Fast file system recovery
- ◆ Access control lists (ACLs)
- ◆ Online administration
- ◆ Online backup
- ◆ Enhanced application interface
- ◆ Enhanced mount options
- ◆ Improved synchronous write performance
- ◆ Support for file systems up to 32 terabytes in size
- ◆ Support for files up to 1 terabyte in size (up to two terabytes for sparse files)
- ◆ Enhanced I/O performance
- ◆ Quotas
- ◆ Improved database performance
- ◆ Storage Checkpoints
- ◆ Cluster file systems
- ◆ Support for improved network file server (NFS) performance through use of VERITAS QuickLog™
- ◆ VxFS supports all UFS file system features and facilities except for the linking, removing, or renaming of “.” and “..” directory entries. Such operations may disrupt file system sanity.

Disk Layouts

The disk layout is the way file system information is stored on disk. On VxFS, five disk layout versions, numbered 1 through 5, were created to support various new features and specific UNIX environments. Currently, only the Version 4 and 5 disk layouts can be created, but file systems with Version 1 and Version 2 disk layouts can be mounted.

See [“Disk Layout”](#) on page 157 for a description of the disk layouts.

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. An overview of the VxFS allocation policy is provided in the section [“Extent Based Allocation”](#) on page 5.

VxFS provides the following performance enhancements:

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

The rest of this chapter, as well as [“VxFS Performance: Creating, Mounting, and Tuning File Systems”](#) on page 17 and [“Application Interface”](#) on page 43 provide details on many of these features.



VERITAS Enterprise Administrator Graphical User Interface

The VERITAS Enterprise Administrator™ (VEA) is a Java-based GUI that consists of a server and a client. The server runs on a UNIX system that is running the VERITAS Volume Manager and VxFS. The client runs on any platform that supports the Java Runtime Environment. You can use VEA to perform a subset of VxFS administrative functions on a local or remote system. These functions include:

- ◆ Adding a VxFS File System to a VERITAS Volume Manager Volume
- ◆ Removing a File System from the File System Table
- ◆ Mounting/Unmounting a File System
- ◆ Defragmenting a File System
- ◆ Resizing a File System
- ◆ Creating a Snapshot Copy of a File System
- ◆ Checking a File System on a Volume
- ◆ Viewing File System Properties
- ◆ Using the QuickLog Feature
- ◆ Unmounting a File System from a Cluster Node
- ◆ Removing Resource Information for a Cluster File System

For instructions on how to use VEA, see the *VERITAS Volume Manager User's Guide – VERITAS Enterprise Administrator*. This guide is available in the `/opt/VRTSvmdoc` directory after you install the VRTSvmdoc package.



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 8 TB, 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 also points to two indirect address extents, which contain the addresses of other extents:

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

Each indirect address extent is 8K long and contains 2048 entries. All indirect data extents for a file must be the same size; this size is set when the first indirect data extent is allocated and stored in the inode. Directory inodes always use an 8K indirect data extent size. By default, regular file inodes also use an 8K indirect data extent size that can be altered with `vxtunefs` (see “[Tuning VxFS I/O Parameters](#)” on page 30); these inodes allocate the indirect data extents in clusters to simulate larger extents.



Typed Extents

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

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

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

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

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

Extent Attributes

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

The `setext` and `getext` commands allow the administrator to set or view extent attributes associated with a file, as well as to preallocate space for a file. Refer to “[Extent Attributes](#)” on page 37, “[Application Interface](#)” on page 43, and the `setext(1)` and `getext(1)` manual pages for discussions on how to use extent attributes.

The `vxtunefs` command allows the administrator to set or view the default indirect data extent size. Refer to “[VxFS Performance: Creating, Mounting, and Tuning File Systems](#)” on page 17 and the `vxtunefs(1M)` manual page for discussions on how to use the indirect data extent size feature.

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

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

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



Online System Administration

A VxFS file system can be defragmented and resized while it remains online and accessible to users. The following sections provide an overview of these features.

Defragmentation

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

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

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

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

Resizing

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

Most large file systems with too much space try to reclaim the unused space by off-loading the contents of the file system and rebuilding it to a preferable size. The VxFS utility `fsadm` can expand or shrink a file system without unmounting the file system or interrupting user productivity. However, to expand a file system, the underlying device on which it is mounted must be expandable.

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

Application Interface

VxFS conforms to the System V Interface Definition (SVID) requirements and supports user access through the Network File System (NFS). Applications that require performance features not available with other file systems can take advantage of VxFS enhancements that are introduced in this section and covered in detail in [“Application Interface”](#) on page 43.

Application Transparency

In most cases, any application designed to run on native file systems will run transparently on VxFS.

Expanded Application Facilities

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

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

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



Extended mount Options

The VxFS file system supports extended mount options to specify:

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

See “[VxFS Performance: Creating, Mounting, and Tuning File Systems](#)” on page 17 and the `mount_vxfs(1M)` manual page for details on the VxFS mount options.

Enhanced Data Integrity Modes

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

Most file systems are “buffered” in that resources are allocated to files and data is written asynchronously to files. In general, the buffering schemes provide better performance without compromising data integrity.

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

Using `blkclear` Option for Data Integrity

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

Using `closesync` Option for Data Integrity

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

Using the log Option for 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. If a system failure occurs, `fsck` replays any recent changes to preserve all metadata. Recent file data may be lost unless a request was made to `sync` it to disk.

Enhanced Performance Mode

VxFS has several mount options that improve performance such as `delaylog` and `qlog`.

Using the delaylog Option for Enhanced Performance

The default VxFS logging mode, `mount -o delaylog`, increases performance by delaying the logging of some structural changes, but does not provide the equivalent data integrity as the previously described modes. That is 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. `delaylog` is the default mount option.

Using the qlog Option for Enhanced Performance

VxFS provides the `mount -o qlog=` option to activate QuickLog for a file system. QuickLog increases VxFS performance by exporting the file system log to a separate physical volume. This eliminates the disk seek time between the VxFS data and log areas on disk and increases the performance of synchronous log writes. See “[VERITAS QuickLog](#)” on page 95 for details.



Temporary File System Modes

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

Using the `tmplog` option For Temporary File Systems

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

Improved Synchronous Writes

VxFS provides superior performance for synchronous write applications. The default `mount datainlog` option greatly improves the performance of small synchronous writes.

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

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

Support for Large Files

VxFS can support files up to two terabytes in size. See “[largefiles](#) | [nolargefiles](#)” on page 24 for information on how to create, mount, and manage file systems containing large files.

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

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.

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.

For a discussion on VxVM integration and performance benefits, refer to [“VxFS Performance: Creating, Mounting, and Tuning File Systems”](#) on page 17, [“Application Interface”](#) on page 43, and the `vxtunefs(1M)` and `tunefstab(1M)` manual pages.



Access Control Lists

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

Storage Checkpoints

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

See [“Storage Checkpoints”](#) on page 51 for information on using the Storage Checkpoint feature.

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 [“Online Backup”](#) on page 69 for information on doing backups 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. You can assign quotas for each of these resources. Each quota consists of two limits for each resource:

- ◆ The *hard limit* represents an absolute limit on data blocks or files. 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 temporarily exceed limits as long as they fall under those limits before the allotted time expires.

See “[Quota Limits](#)” on page 78 for details on using VxFS quotas.

Support for Databases

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

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

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

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

See “[Quick I/O for Databases](#)” on page 83 for details on VxFS database support.



VERITAS QuickLog

VERITAS QuickLog moves the VxFS intent log from the physical volume containing the file system onto a separate physical volume. Without QuickLog, the intent log information for VxFS is usually stored near the beginning of the file system volume. Because other disk operations (inode, data) are issued on the same volume, the disk heads seek between the log and file system data areas. Using QuickLog eliminates the seek time between the log and file system data on the same volume, thereby improving system performance.

See “[VERITAS QuickLog](#)” on page 95 for details on the VERITAS QuickLog feature.

Cluster File Systems

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

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 SANPoint Foundation Suite HA to provide a complete clustering environment.

See *The VERITAS SANPoint Foundation Suite Installation and Configuration Guide*, included in the VERITAS SANPoint Foundation Suite product, for more information.

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*.

VxFS Performance: Creating, Mounting, and Tuning File Systems

2

Introduction

For any file system, the ability to provide peak performance is important. Adjusting the available VERITAS File System (VxFS) options provides a way to increase system performance. This chapter describes the commands and practices you can use to optimize VxFS. For information on optimizing an application for use with VxFS, see “[Application Interface](#)” on page 43.

The following topics are covered in this chapter:

- ◆ [Choosing mkfs Command Options](#)
 - [Block Size](#)
 - [Intent Log Size](#)
- ◆ [Choosing mount Command Options](#)
 - [log](#)
 - [delaylog](#)
 - [tmplog](#)
 - [logiosize](#)
 - [nodatainlog](#)
 - [blkclear](#)
 - [mincache](#)
 - [convosync](#)
 - [qlog](#)
 - [largefiles](#) | [nolargefiles](#)
 - [Combining mount Command Options](#)



- ◆ **Kernel Tunables**
 - Internal Inode Table Size
 - vx_maxlink
 - VxVM Maximum I/O Size
- ◆ **Monitoring Free Space**
 - Monitoring Fragmentation
- ◆ **I/O Tuning**
 - Tuning VxFS I/O Parameters
 - Tunable VxFS I/O Parameters

Choosing mkfs Command Options

There are several characteristics that you can select when you create a file system. The most important options pertaining to system performance are the block size and intent log size.

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 (see “[The VxFS Version 5 Disk Layout](#)” on page 162 for more information).

Intent Log Size

You specify the intent log size when creating a file system by using the `mkfs -o logsize` option. The intent log size cannot be altered after the file system is created. The `mkfs` utility uses a default intent log size of 16 megabytes. The default size is sufficient for most workloads. If the system is used as an NFS server or for intensive synchronous write workloads, performance may be improved using a larger log size.

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

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

Note When using QuickLog, you choose the log at creation time and can easily change it at any time during use. For more information on log creation, log manipulation, and load balancing, see “[VERITAS QuickLog](#)” on page 95.

Choosing mount Command Options

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

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

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

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



log

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

delaylog

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

tmplog

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

logiosize

The `logiosize=size` option is provided to enhance 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 at least *size* bytes to obtain the maximum performance from such devices. The values for *size* can be 512, 1024, 2048, 4096, or 8192.

nodatainlog

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

blkclear

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

mincache

The `mincache` mode has five suboptions:

- ◆ `mincache=closesync`
- ◆ `mincache=direct`
- ◆ `mincache=dsync`
- ◆ `mincache=unbuffered`
- ◆ `mincache=tmppcache`

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



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

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

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

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

convosync

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

The `convosync` (convert `osync`) mode has five suboptions:

- ◆ `convosync=closesync`
- ◆ `convosync=delay.`
- ◆ `convosync=direct`
- ◆ `convosync=dsync`
- ◆ `convosync=unbuffered`

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

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

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

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

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

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



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

qlog

The `qlog` option can be used in conjunction with the name of a QuickLog device. For example, to set the QuickLog device `vxlog1` to log the file system, use `qlog=vxlog1`. If `qlog=` is specified with no QuickLog device, the QuickLog driver chooses an appropriate log device automatically. For more information, see “[VERITAS QuickLog](#)” on page 95.

largefiles | nolargefiles

VxFS supports files up to one terabyte in size.

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

Creating a File System with Large Files

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

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

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

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

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

Mounting a File System with Large Files

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

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

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

Managing a File System with Large Files

You can determine the current status of the `largefiles` flag using the `fsadm` or `mkfs` command:

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

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

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

You can also switch capabilities on an unmounted file system:

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

You cannot 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.



Combining mount Command Options

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

Example 1 - Desktop File System

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

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

Example 2 - Temporary File System or Restoring from Backup

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

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

Example 3 - Data Synchronous Writes

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

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

Kernel Tunables

This section describes the kernel tunable parameters in VxFS.

Internal Inode Table Size

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 value, make the following change in `/etc/system` and reboot:

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

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

```
set ncsiz = 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 (FFFF hexadecimal). Changes to `vx_maxlink` take effect after rebooting.



VxVM Maximum I/O Size

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

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

`vol_maxio`

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

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

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

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

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

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

Monitoring Free Space

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

Monitoring Fragmentation

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

The easiest way to ensure that fragmentation does not become a problem is to schedule regular defragmentation runs 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` or the `df -o s` commands. There are three factors which can be used to determine the degree of fragmentation:

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

An unfragmented file system will have the following characteristics:

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

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

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

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

The “before” result is the degree of fragmentation prior to the reorganization. If the degree of fragmentation is approaching the figures for bad fragmentation, 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 will improve performance. It is advisable to schedule directory reorganization for file systems when the extent reorganization is scheduled. The following is a sample script that is run periodically at 3:00 A.M. from `cron` for a number of file systems:

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

I/O Tuning

Note The tunables and the techniques described in this section 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.

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

Tuning VxFS I/O Parameters

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

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

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

```
# vxtunefs -p mount_point
```

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 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/opt
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 VxFS I/O Parameters

<code>read_pref_io</code>	The preferred read request size. The file system uses this in conjunction with the <code>read_nstream</code> value to determine how much data to read ahead. The default value is 64K.
<code>write_pref_io</code>	The preferred write request size. The file system uses this in conjunction with the <code>write_nstream</code> value to determine how to do flush behind on writes. The default value is 64K.
<code>read_nstream</code>	The number of parallel read requests of size <code>read_pref_io</code> to have outstanding at one time. The file system uses the product of <code>read_nstream</code> multiplied by <code>read_pref_io</code> to determine its read ahead size. The default value for <code>read_nstream</code> is 1.
<code>write_nstream</code>	The number of parallel write requests of size <code>write_pref_io</code> to have outstanding at one time. The file system uses the product of <code>write_nstream</code> multiplied by <code>write_pref_io</code> to determine when to do flush behind on writes. The default value for <code>write_nstream</code> is 1.
<code>default_indir_size</code>	On VxFS, files can have up to ten direct extents of variable size stored in the inode. Once these extents are used up, the file must use indirect extents which are a fixed size that is set when the file first uses indirect extents. These indirect extents are 8K by default. The file system does not use larger indirect extents because it must fail a write and return <code>ENOSPC</code> 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 <code>default_indir_size</code> should be used carefully. If it is set too large, then writes will fail when they are unable to allocate extents of the indirect extent size to a file. In general, the fewer and the larger the files on a file system, the larger the <code>default_indir_size</code> can be set. This parameter should generally be set to some multiple of the <code>read_pref_io</code> parameter. <code>default_indir_size</code> is not applicable on Version 4 disk layouts.

<code>discovered_direct_iosz</code>	Any file I/O requests larger than the <code>discovered_direct_iosz</code> 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.
<code>hsm_write_prealloc</code>	For a file managed by a hierarchical storage management (HSM) application, <code>hsm_write_prealloc</code> 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 <code>hsm_write_prealloc</code> (<code>hsm_write_prealloc=1</code>), 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 <code>hsm_write_prealloc</code> 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 <code>hsm_write_prealloc</code> only when recommended by the HSM application controlling the file system.
<code>initial_extent_size</code>	Changes the default initial extent size. VxFS determines, based on the first write to a new file, the size of the first extent to be allocated to the file. Normally the first extent is the smallest power of 2 that is larger than the size of the first write. If that power of 2 is less than 8K, the first extent allocated is 8K. After the initial extent, the file system increases the size of subsequent extents (see <code>max_seqio_extent_size</code>) with each allocation. Since most applications write to files using a buffer size of 8K or less, the increasing extents start doubling from a small initial extent. <code>initial_extent_size</code> can change the default initial extent size to be larger, so the doubling policy will start from a much larger initial size and the file system will not allocate a set of small extents at the start of file. Use this parameter only on file systems that will have a very large average file size. On these file systems it will result in fewer extents per file and less fragmentation. <code>initial_extent_size</code> is measured in file system blocks.



<code>max_direct_iosz</code>	The maximum size of a direct I/O request that will be issued by the file system. If a larger I/O request comes in, then it is broken up into <code>max_direct_iosz</code> 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.
<code>max_diskq</code>	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 <code>max_diskq</code> , processes will block until the amount of data being flushed decreases. Although this doesn't limit the actual disk queue, it prevents flushing processes from making the system unresponsive. The default value is 1 MB.
<code>max_seqio_extent_size</code>	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. <code>max_seqio_extent_size</code> is measured in file system blocks.
<code>qio_cache_enable</code>	Enables or disables caching on Quick I/O files. The default behavior is to disable caching. To enable caching, set <code>qio_cache_enable</code> to 1. On systems with large memories, the database cannot always use all of the memory as a cache. By enabling file system caching as a second level cache, performance may be improved. If the database is performing sequential scans of tables, the scans may run faster by enabling file system caching so the file system will perform aggressive read-ahead on the files.

`write_throttle`

The `write_throttle` parameter is useful in special situations where a computer system has a combination of a 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 will generate before flushing the pages to disk. After the number of dirty pages for a file reaches the `write_throttle` threshold, the file system starts flushing pages to disk even if free memory is still available.

The default value of `write_throttle` 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 will complete much faster.

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



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

If the file system is being used with a hardware disk array or volume manager other than VxVM, try to align the parameters to match the geometry of the logical disk. With striping or RAID-5, it is common to set `read_pref_io` to the stripe unit size and `read_nstream` to the number of columns in the stripe. For 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 issue read requests that are equal to the product of `read_nstream` multiplied by `read_pref_io`. Generally, any multiple or factor of `read_nstream` multiplied by `read_pref_io` should be a good size for performance. For writing, the same rule of thumb applies to the `write_pref_io` and `write_nstream` parameters. When tuning a file system, the best thing to do is try out the tuning parameters under a real life workload.

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

Introduction

The VERITAS File System (VxFS) allocates disk space to files in groups of one or more adjacent blocks called *extents*. VxFS defines an application interface that allows programs to control various aspects of the extent allocation for a given file (see “[Extent Information](#)” on page 46). The extent allocation policies associated with a file are referred to as *extent attributes*.

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

The following topics are covered in this chapter:

- ◆ [Attribute Specifics](#)
 - [Reservation: Preallocating Space to a File](#)
 - [Fixed Extent Size](#)
 - [Other Controls](#)
- ◆ [Commands Related to Extent Attributes](#)
 - [Failure to Preserve Extent Attributes](#)



Attribute Specifics

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

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

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

Some of the extent attributes are persistent and become part of the on-disk information about the file, while other attributes are temporary and are lost after the file is closed or the system is rebooted. The persistent attributes are similar to the file's permissions and are written in the inode for the file. When a file is copied, moved, or archived, only the persistent attributes of the source file are preserved in the new file (see [“Other Controls”](#) on page 40 for more information).

In general, the user will only set extent attributes for reservation. Many of the attributes are designed for applications that are tuned to a particular pattern of I/O or disk alignment (see the `mkfs_vxfs(1M)` manual page and [“Application Interface”](#) on page 43 for more information).

Reservation: Preallocating Space to a File

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

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

Fixed Extent Size

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

Fixed extent sizes are particularly appropriate in the following situations:

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

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



Other Controls

The auxiliary controls on extent attributes determine:

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

Alignment

Specific alignment restrictions coordinate a file's allocations with a particular I/O pattern or disk alignment (see the `mkfs_vxfs(1M)` manual page and [“Application Interface”](#) on page 43 for details). Alignment can only be specified if a fixed extent size has also been set. Setting alignment restrictions on allocations is best left to well designed applications.

Contiguity

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

Note Fixed extent sizes or alignment 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.

Commands Related to Extent Attributes

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

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

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

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

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



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

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

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

Failure to Preserve Extent Attributes

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

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

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

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

Introduction

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

If you are writing applications, you can optimize them for use with the VxFS. To optimize VxFS for use with applications, see [“VxFS Performance: Creating, Mounting, and Tuning File Systems”](#) on page 17.

The following topics are covered in this chapter:

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



Cache Advisories

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

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

Direct I/O

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

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

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

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.

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

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

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

Unbuffered I/O

If the `VX_UNBUFFERED` advisory is set, I/O behavior is the same as direct I/O with the `VX_DIRECT` advisory set, so the alignment constraints that apply to direct I/O also apply to unbuffered 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.

Discovered Direct I/O

Discovered Direct I/O is a file system tunable you can 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.

For information on how to set the `discovered_direct_iosz`, see [“I/O Tuning”](#) on page 30.

Data Synchronous I/O

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

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

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



Other Advisories

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

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

The `VX_NOREUSE` advisory is used as a modifier. If both `VX_RANDOM` and `VX_NOREUSE` are set, VxFS notifies the operating system that the pages are free and may be reclaimed. If `VX_NOREUSE` is set when doing sequential I/O, pages are also freed when they are flushed to disk. The `VX_NOREUSE` advisory may slow down access to the file, but it can reduce the cached data held by the system. This can allow more data to be cached for other files and may speed up those accesses.

Extent Information

The `VX_SETEXT` ioctl command allows an application to reserve space for a file, and set fixed extent sizes and file allocation flags. Applications can obtain status information on VxFS ioctls by using the `VX_GETTEXT` ioctl. The `getext` command also provides access to this information. See the `getext(1)`, `setext(1)`, and `vxfsio(7)` manual pages for more information.

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

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

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

Caution Follow this procedure carefully. A fixed extent size may be inadvertently cleared when the reservation is changed.

Space Reservation

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

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

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

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

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

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



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

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

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

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

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

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

Fixed Extent Sizes

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

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

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

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

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

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



Freeze and Thaw

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

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

Get I/O Parameters ioctl

VxFS provides the `VX_GET_IOPARAMETERS` ioctl to get the recommended I/O sizes to use on a file system. This ioctl can be used by the application to make decisions about the I/O sizes issued to VxFS for a file or file device. For more details on this ioctl, refer to the `vxfsio(7)` manual page. For a discussion on various I/O parameters, refer to [“VxFS Performance: Creating, Mounting, and Tuning File Systems”](#) on page 17 and the `vxtunefs(1M)` manual page.

Storage Checkpoints are a feature of the VERITAS File System (VxFS) that provide *point-in-time* images of file system contents. These frozen images of VxFS file systems can be used in a variety of applications such as full and incremental online backups, fast error recovery, and product development testing. Storage Checkpoint replicas of real time databases can also be used for decision support and an assortment of database analyses.

Note You can use Storage Checkpoints only on file systems using the Version 4 or Version 5 disk layout; if you are using an older disk layout, you must upgrade to use this feature (see the `vxupgrade(1M)` and `vxfsconvert(1M)` manual page for information on how to upgrade the VxFS disk layout).

The following topics are covered in this chapter:

- ◆ [What is a Storage Checkpoint?](#)
- ◆ [How a Storage Checkpoint Works](#)
- ◆ [Types of Storage Checkpoints](#)
 - [Data Storage Checkpoints](#)
 - [Nodata Storage Checkpoints](#)
 - [Removable Storage Checkpoints](#)
 - [Non-mountable Storage Checkpoints](#)
- ◆ [Storage Checkpoint Administration](#)
 - [Creating a Storage Checkpoint](#)
 - [Removing a Storage Checkpoint](#)
 - [Accessing a Storage Checkpoint](#)
 - [Converting a Data Storage Checkpoint to a Nodata Storage Checkpoint](#)
- ◆ [Space Management Considerations](#)



What is a Storage Checkpoint?

The VERITAS File System provides a unique Storage Checkpoint facility which quickly creates a persistent image of a file system at an exact point in time. Storage Checkpoints significantly reduce I/O overhead by identifying and maintaining only the file system blocks that have changed since the last Storage Checkpoint or backup via a *copy-on-write* technique (see “[How a Storage Checkpoint Works](#)” on page 53). Unlike a disk-based mirroring technology that requires a separate storage space, this VERITAS technology minimizes the use of disk space by creating a Storage Checkpoint within the same free space available to the file system.

Storage Checkpoints are data objects which are managed and controlled by the file system; as a result, Storage Checkpoints are persistent across system reboots and crashes. You can create, remove, and rename Storage Checkpoints because they are data objects with associated names (see “[Storage Checkpoint Administration](#)” on page 57). After you create a Storage Checkpoint of a mounted file system, you can also continue to create, remove, and update files on the file system without affecting the logical image of the Storage Checkpoint. This technology preserves not only the name space (directory hierarchy) of the file system, but also the user data as it existed at the moment the Storage Checkpoint was taken.

Storage Checkpoints differ from VERITAS File System snapshots in the following ways:

- ◆ 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.
- ◆ Multiple, read-only Storage Checkpoints 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 which only need to retrieve the changed data. Storage Checkpoints significantly minimize data movement and may promote higher availability and data integrity by increasing the frequency of backup and replication solutions.

Storage Checkpoints can be taken in environments with a large number of files (for example, 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. Storage Checkpoint creation, however, may take several minutes to complete depending on the number of files in the file system.

How a Storage Checkpoint Works

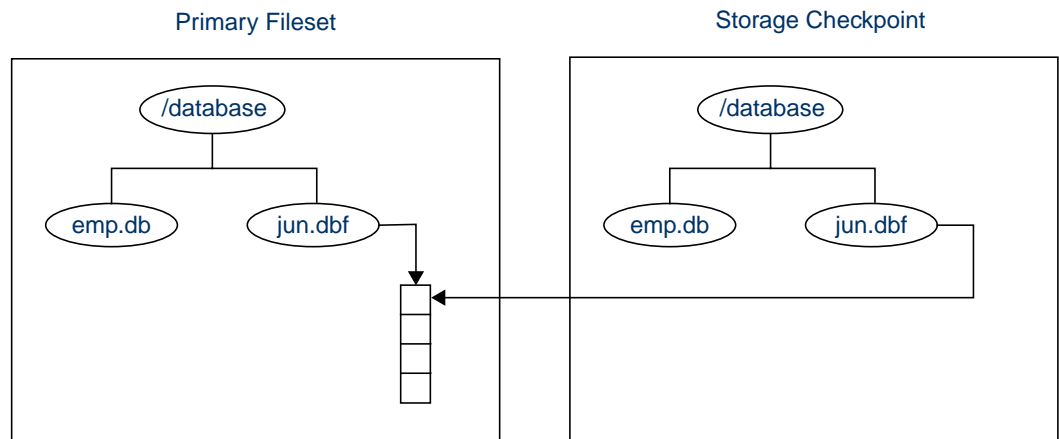
The Storage Checkpoint facility freezes the mounted file system (known as the *primary fileset*), initializes the Storage Checkpoint, and thaws the file system. Specifically, the file system is first brought to a stable state where all of its data is written to disk, and the freezing process momentarily 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* (described below) of the primary fileset. The *thawing* process that follows restarts I/O operations to the file system.

You can create a Storage Checkpoint on a single file system or a list of file systems. A multiple file system Storage Checkpoint simultaneously freezes the file systems, creates a Storage Checkpoint on all file systems, and thaws the file systems. As a result, the Storage Checkpoints for multiple file systems have the same creation timestamp. The Storage Checkpoint facility guarantees that multiple file system Storage Checkpoints are created on all or none of the specified file systems (unless 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.

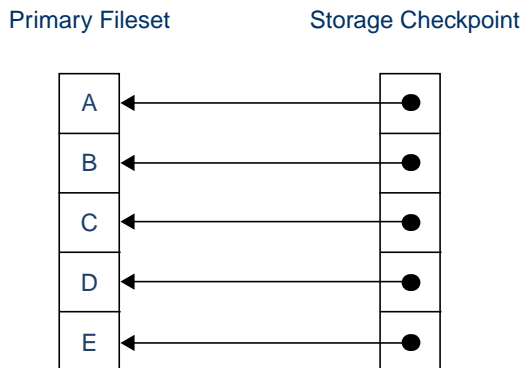
As mentioned above, a Storage Checkpoint of the primary fileset initially contains a pointer to the file system block map rather than to any actual data. The block map points to the data on the primary fileset. The figure below shows the file system `/database` and its Storage Checkpoint. The Storage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

Primary Fileset and Its Storage Checkpoint



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

Initializing a Storage Checkpoint

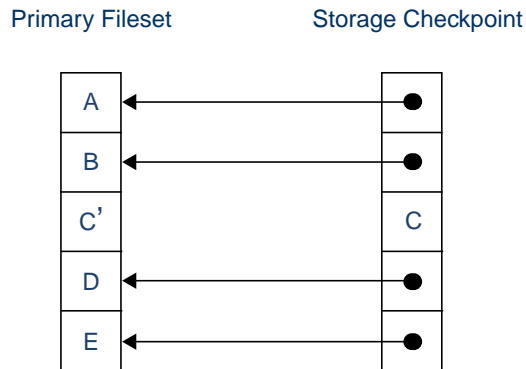


The Storage Checkpoint presents the exact image of the file system by finding the data from the primary fileset. As the primary fileset is updated, the original data is copied to the Storage Checkpoint before the new data is written. When a write operation changes a specific data block in the primary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

In the following figure, 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.

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 which require a consistent and stable image of an active file system. Data Storage Checkpoints introduce some overhead to the system and to the application performing the write operation. For best results, you can limit the life of data Storage Checkpoints to minimize the impact on system resources.

Nodata Storage Checkpoints

A nodata Storage Checkpoint only contains file system metadata—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. Nodata Storage Checkpoints are useful with the Block Level Incremental Backup feature.

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 68 for more information). 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

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

Storage Checkpoint Administration

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

Creating a Storage Checkpoint

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

This example shows the creation of a nodata Storage Checkpoint (see “[Space Management Considerations](#)” on page 68) named `thu_7pm` on `/mnt0` and lists all Storage Checkpoints of the `/mnt0` file system:

```
# fsckptadm -n create thu_7pm /mnt0
# fsckptadm list /mnt0
/mnt0
thu_7pm:
  ctime= Thu Jun 1 19:02:17 2001
  mtime= Thu Jun 1 19:02:17 2001
  flags= nodata
```

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

```
# fsckptadm -r create thu_8pm /mnt0
# fsckptadm list /mnt0
/mnt0
thu_8pm:
  ctime= Thu Jun 1 20:01:19 2001
  mtime= Thu Jun 1 20:01:19 2001
  flags= removable
thu_7pm:
  ctime= Thu Jun 1 19:02:17 2001
  mtime= Thu Jun 1 19:02:17 2001
  flags= nodata
```



Removing a Storage Checkpoint

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

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

```
# fsckptadm remove thu_8pm /mnt0
# fsckptadm list /mnt0
/mnt0
thu_7pm:
  ctime= Thu Jun 1 19:02:17 2001
  mtime= Thu Jun 1 19:02:17 2001
  flags= nodata
# fsckptadm -s remove thu_7pm /mnt0
# fsckptadm list /mnt0
/mnt0
```

Accessing a Storage Checkpoint

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

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

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

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

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

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



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

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

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

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

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

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

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

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

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

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

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

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

Converting a Data Storage Checkpoint to a Nodata Storage Checkpoint

A nodata Storage Checkpoint does not contain actual file data. Instead, this type of Storage Checkpoint contains a collection of markers indicating the location of all the changed blocks since the Storage Checkpoint was created (see “[Types of Storage Checkpoints](#)” on page 56 for more information).

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.



Difference Between a Data and a Nodata Storage Checkpoint

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

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

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

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

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

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

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

4. Change the content of the original file:

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

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

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

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

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

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

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

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

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



Conversion with Multiple Storage Checkpoints

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

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

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

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

```
# fsckptadm create oldest /mnt0
# fsckptadm create older /mnt0
# fsckptadm create old /mnt0
# fsckptadm create latest /mnt0
# fsckptadm list /mnt0
/mnt0
latest:
  ctime                =Mon Oct 16 11:56:55 2001
  mtime                =Mon Oct 16 11:56:55 2001
  flags                =none
old:
  ctime                =Mon Oct 16 11:56:51 2001
  mtime                =Mon Oct 16 11:56:51 2001
  flags                =none
older:
  ctime                =Mon Oct 16 11:56:46 2001
  mtime                =Mon Oct 16 11:56:46 2001
  flags                =none
oldest:
  ctime                =Mon Oct 16 11:56:41 2001
  mtime                =Mon Oct 16 11:56:41 2001
  flags                =none
```


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

```
# fsckptadm -s set nodata latest /mnt0
vxfs fsckptadm: checkpoint set failed on latest.
Do not specify on an existing file (17)
```

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

```
# fsckptadm set nodata latest /mnt0
# fsckptadm list /mnt0
/mnt0
latest:
  ctime          =Mon Oct 16 11:56:55 2001
  mtime          =Mon Oct 16 11:56:55 2001
  flags          =nodata, delayed
old:
  ctime          =Mon Oct 16 11:56:51 2001
  mtime          =Mon Oct 16 11:56:51 2001
  flags          =none
older:
  ctime          =Mon Oct 16 11:56:46 2001
  mtime          =Mon Oct 16 11:56:46 2001
  flags          =none
oldest:
  ctime          =Mon Oct 16 11:56:41 2001
  mtime          =Mon Oct 16 11:56:41 2001
  flags          =none
```



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

```
# fsckptadm remove latest /mnt0
# fsckptadm list /mnt0
/mnt0
old:
  ctime          =Mon Oct 16 11:56:51 2001
  mtime          =Mon Oct 16 11:56:51 2001
  flags          =none
older:
  ctime          =Mon Oct 16 11:56:46 2001
  mtime          =Mon Oct 16 11:56:46 2001
  flags          =none
oldest:
  ctime          =Mon Oct 16 11:56:41 2001
  mtime          =Mon Oct 16 11:56:41 2001
  flags          =none
```

Then recreate it as a nodata Storage Checkpoint:

```
# fsckptadm -n create latest /mnt0
# fsckptadm list /mnt0
/mnt0
latest:
  ctime          = Mon Oct 16 12:06:42 2001
  mtime          = Mon Oct 16 12:06:42 2001
  flags          = nodata, delayed
old:
  ctime          = Mon Oct 16 11:56:51 2001
  mtime          = Mon Oct 16 11:56:51 2001
  flags          = none
older:
  ctime          = Mon Oct 16 11:56:46 2001
  mtime          = Mon Oct 16 11:56:46 2001
  flags          = none
oldest:
  ctime          = Mon Oct 16 11:56:41 2001
  mtime          = Mon Oct 16 11:56:41 2001
  flags          = none
```

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

```
# fsckptadm -s set nodata oldest /mnt0
# fsckptadm list /mnt0
/mnt0
latest:
  ctime          =Mon Oct 16 12:06:42 2001
  mtime          =Mon Oct 16 12:06:42 2001
  flags          =nodata, delayed
old:
  ctime          =Mon Oct 16 11:56:51 2001
  mtime          =Mon Oct 16 11:56:51 2001
  flags          =none
older:
  ctime          =Mon Oct 16 11:56:46 2001
  mtime          =Mon Oct 16 11:56:46 2001
  flags          =none
oldest:
  ctime          =Mon Oct 16 11:56:41 2001
  mtime          =Mon Oct 16 11:56:41 2001
  flags          =nodata
```

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

```
# fsckptadm remove older /mnt0
# fsckptadm remove old /mnt0
# fsckptadm list /mnt0
/mnt0
latest:
  ctime          =Mon Oct 16 12:06:42 2001
  mtime          =Mon Oct 16 12:06:42 2001
  flags          =nodata
oldest:
  ctime          =Mon Oct 16 11:56:41 2001
  mtime          =Mon Oct 16 11:56:41 2001
  flags          =nodata
```



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. Usually these operations do not fail because of insufficient space on the file system, but these operations on a file system containing Storage Checkpoints can cause a data block copy which, in turn, may require extent allocation. If the system cannot allocate sufficient space, the operation will fail.

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

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

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

Introduction

This chapter describes the online backup facility provided with the VERITAS File System (VxFS). The snapshot feature of VxFS can be used to create a snapshot image of a mounted file system, which becomes a duplicate read-only copy of the mounted file system. This chapter also provides a description of how to create a snapshot file system and some examples of backing up all or part of a file system using the snapshot mechanism.

The following topics are covered in this chapter:

- ◆ [Snapshot File Systems](#)
- ◆ [Using a Snapshot File System for Backup](#)
- ◆ [Creating a Snapshot File System](#)
- ◆ [Making a Backup](#)
- ◆ [Performance of Snapshot File Systems](#)
- ◆ [Differences Between Snapshots and Storage Checkpoints](#)
- ◆ [Snapshot File System Internals](#)
 - [Snapshot File System Disk Structure](#)
 - [How a Snapshot File System Works](#)



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 SANPoint Foundation Suite Installation and Configuration Guide* for more information on creating snapshots on cluster file systems.

Using a Snapshot File System for Backup

After a snapshot file system is created, the snapshot performs 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 `snpread` 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.

If you create a complete backup of a snapshot file system using a utility such as `vxdump` and later restore it, you must run the `fsck` command on the restored file system because the snapshot file system is consistent, but not clean. That is, the file system may have some extended inode operations to complete, but there should be no other changes. Because a snapshot file system is not writable, it cannot be fully checked, but the `fsck -n` command can be used to report any inconsistencies.

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. Use the following syntax to create a snapshot file system:

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

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, it 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.

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



Making a Backup

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

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

```
# mount -F vxfs -o snapof=/home,snapsize=100000 \  
/dev/vx/dsk/fsvol/vol1 /backup/home  
# cd /backup  
# find home -ctime -7 -depth -print | cpio -oc > /dev/rmt/c0s0  
# umount /backup/home
```
- ◆ To do a full backup of /home, which exists on disk /dev/dsk/c0t0d0s7, and use `dd` to control blocking of output onto tape device using `vxdump`:

```
# vxdump f - /dev/vx/dsk/fsvol/vol1 | dd bs=128k > /dev/rmt/c0s0
```
- ◆ To do a level 3 backup of /dev/dsk/c0t0d0s7 and collect those files that have changed in the current directory:

```
# vxdump 3f - /dev/vx/dsk/fsvol/vol1 | vxrestore -xf -
```
- ◆ To do a full backup of a snapshot file system:

```
# 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
```

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

Performance of Snapshot File Systems

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

The performance of reads from the snapped file system are generally not affected. Writes to the snapped file system, however, 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:

- ◆ Snapshots require a separate device for storage. Storage Checkpoints reside on the same device as the original file system.
- ◆ Snapshots are read-only. Storage Checkpoints can be read-only or read-write.
- ◆ Snapshots are transient. Storage Checkpoints are persistent.
- ◆ Snapshots cease to exist after being unmounted. Storage Checkpoints can exist and be mounted on their own
- ◆ Snapshots track changed blocks on the file system level. Storage Checkpoints track changed blocks on each file in the file system.
- ◆ Although there can be more than one snapshot of a file system, they are all based on a single, parent file system. Storage Checkpoints can be based on other Storage Checkpoints.

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 “[Storage Checkpoints](#)” on page 51 for more information.



Snapshot File System Internals

The following sections describe the internal structure of a snapshot file system and how it copies changed data blocks from the original snapped file system.

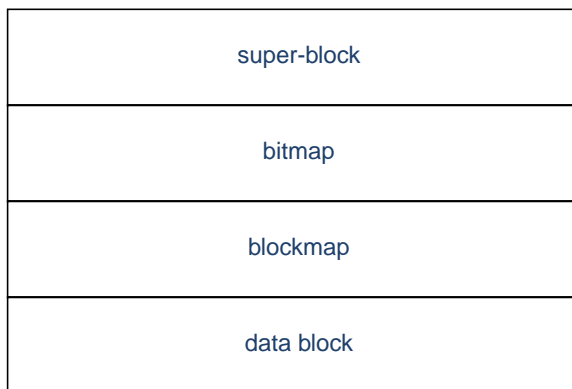
Snapshot File System Disk Structure

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:

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 (see “Freeze and Thaw” on page 50, for a description of the `VX_FREEZE` ioctl). 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.

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 snapped file system). The blockmap entry for block n is changed from 0 to the block number on the snapshot file system containing the old data.

A subsequent read request for block n on the snapshot file system will be satisfied by checking the bitmap entry for block n and reading the data from the indicated block on the snapshot file system, 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.

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

Introduction

The 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.

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.

The following topics are covered in this chapter:

- ◆ [Quota Limits](#)
- ◆ [Quota Files on VxFS](#)
- ◆ [Quota Commands](#)
- ◆ [Quota Checking With VxFS](#)
- ◆ [Using Quotas](#)



Quota Limits

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

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

A typical use of soft limits is when a user must run an application that could generate large temporary files. In this case, you can allow the user to exceed the quota limit for a limited time. No allocations are allowed after the expiration of the time limit. Use the `vxedquota` command to set limits (see “[Using Quotas](#)” on page 80 for an example).

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 “[Quota Files on VxFS](#)” below).

The quota soft limit can be exceeded when VxFS preallocates space to a file. See “[Attribute Specifics](#)” on page 38 for information on extent allocation policies.

Quota limits cannot exceed one terabyte on a Version 5 disk layout.

Quota Files on VxFS

A quotas file (named `quotas`) must exist in the root directory of a file system for any of the quota commands to work. For group quotas to work, there must be a `quotas.grp` file. The files in the root directory 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.

Quota Commands

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. This is discussed in more detail in [“Quota Checking With VxFS.”](#)

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

VxFS supports the following quota-related commands:

- ◆ `vxedquota`—used to edit quota limits for users 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`—used to view quota limits and usage.
- ◆ `vxquotaon`—used to turn quotas on for a mounted VxFS file system.
- ◆ `vxquotaoff`—used to turn quotas off for a mounted VxFS file system.

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

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



Quota Checking With VxFS

The standard practice with most quota implementations is to mount all file systems and then run a quota check on each one. The quota check reads all the inodes on disk and calculates the usage for each user 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 `fsck_vxfs(1M)`).

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

This section shows usage examples of the VxFS quota commands.

vxquotaon

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

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

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

```
# vxquotaon /mount_point
```

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

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

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

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


mount

You can also turn on user or group quotas for a file system at mount time by specifying the `-o quota` option to the `mount` command:

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

To turn on only user quotas, enter:

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

To turn on only group quotas, enter:

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

vxedquota

You can set up user and group quotas using the `vxedquota` command. You must have superuser privileges to edit quotas. By default, or when you specify the `-u` option, `vxedquota` edits the quotas of one or more users specified by *username*:

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

When you specify the `-g` option, `vxedquota` edits the quotas of one or more groups specified by *groupname*:

```
# vxedquota -g groupname
```

`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.

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. Time limits can be modified for any user with the command:

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

Time limits can be modified for any group with the command:

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

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



vxquota

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

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
```

vxquot

Use the `vxquot` command to display the number of blocks owned by each user or group in a file system. The following command displays the number of files and the space owned by each user:

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

The following command displays the number of files and the space owned by each group:

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

vxquotaoff

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

```
# vxquotaoff /mount_point
```

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

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

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

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

Introduction

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

Quick I/O is part of the `VRTSVxfs` package, but is available for use only with other VERITAS products. See the *VERITAS File System Release Notes* for current product information.

Topics covered in this chapter:

- ◆ Quick I/O Functionality and Performance
- ◆ Using VxFS Files as Raw Character Devices
- ◆ Creating a Quick I/O File Using `qiomkfile`
- ◆ Accessing Regular VxFS Files Through Symbolic Links
- ◆ Using Quick I/O with Oracle Databases
- ◆ Using Quick I/O with Sybase Databases
- ◆ Enabling and Disabling Quick I/O
- ◆ Cached Quick I/O For Databases
- ◆ Quick I/O Statistics
- ◆ Quick I/O Summary



Quick I/O Functionality and Performance

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

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

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

Supporting Kernel Asynchronous I/O

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

Supporting Direct I/O

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

Avoiding Kernel Write Locks

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

Avoiding Double Buffering

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

Using VxFS Files as Raw Character Devices

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

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

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

Quick I/O Naming Convention

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

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

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

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

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



Use Restrictions

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

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

Creating a Quick I/O File Using `qiomkfile`

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

- | | |
|----|--|
| -a | Creates a symbolic link with an absolute path name for a specified file. The default is to create a symbolic link with a relative path name. |
| -e | (For Oracle database files to allow tablespace resizing.) Extends the file size <i>by</i> 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 <i>to</i> 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.

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

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

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

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

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

or:

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

If you specify the -a option to qiomkfile, an absolute path name (see [“Using Absolute or Relative Path Names”](#) on page 88) is used so /database/dbfile points to /database/.dbfile::cdev:vxfs:. To check the results, enter:

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

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



Accessing Regular VxFS Files Through Symbolic Links

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

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

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

Any database or application can then access the file `dbfile` as a raw character device. See the VERITAS Editions product documentation for more information.

Using Absolute or Relative Path Names

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

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

It is also possible to use the absolute path name when creating a symbolic link. If the database file is relocated to another directory, 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. The following example shows how to use `setext` to create a 100 MB database file for an Oracle database:

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

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

Using Quick I/O with Oracle Databases

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

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

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

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



Using Quick I/O with Sybase Databases

Quick I/O works similarly on Sybase database devices.

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

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

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

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

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

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

Enabling and Disabling Quick I/O

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

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

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

Cached Quick I/O For Databases

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

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

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.

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



Enabling Cached Quick I/O

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

1. Set the `qio_cache_enable` parameter of `vxtunefs` to enable caching on a file system.
2. Enable the Cached Quick I/O feature for specific files using the `qioadmin` command.

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

Enabling Cached Quick I/O for File Systems

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

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

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

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

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

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

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

Enabling Cached Quick I/O for Individual Files

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

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

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

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

To disable the caching for that file, enter:

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

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

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

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

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

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

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

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

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

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



Tuning Cached Quick I/O

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

See the `qiostat(1)` man page for information on gathering statistics, and the VERITAS Editions products documentation for a description of the Cached Quick I/O tuning methodology.

Quick I/O Statistics

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

Quick I/O Summary

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

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

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

```
forceload: drv/fdd
```
3. Create a regular VxFS file and preallocate it to the required size, 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:.`

For information on how to configure VxFS and set up file devices for use with new and existing Oracle databases, see the VERITAS Editions product documentation.

Introduction

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

VERITAS QuickLog is part of the `VRTSvxfs` package, but is available for use only with other VERITAS products. See the *VERITAS File System Release Notes* for current product information.

Topics in this chapter include:

- ◆ VERITAS QuickLog Overview
- ◆ QuickLog Setup
 - Creating a QuickLog Device
 - Removing a QuickLog Device
- ◆ VxFS Administration Using QuickLog
 - Enabling a QuickLog Device
 - Disabling a QuickLog Device
- ◆ QuickLog Administration and Troubleshooting
 - QuickLog Load Balancing
 - QuickLog Statistics
 - QuickLog Recovery
- ◆ Cluster QuickLog Devices



VERITAS QuickLog Overview

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

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

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

Note QuickLog cannot be enabled on a root file system.

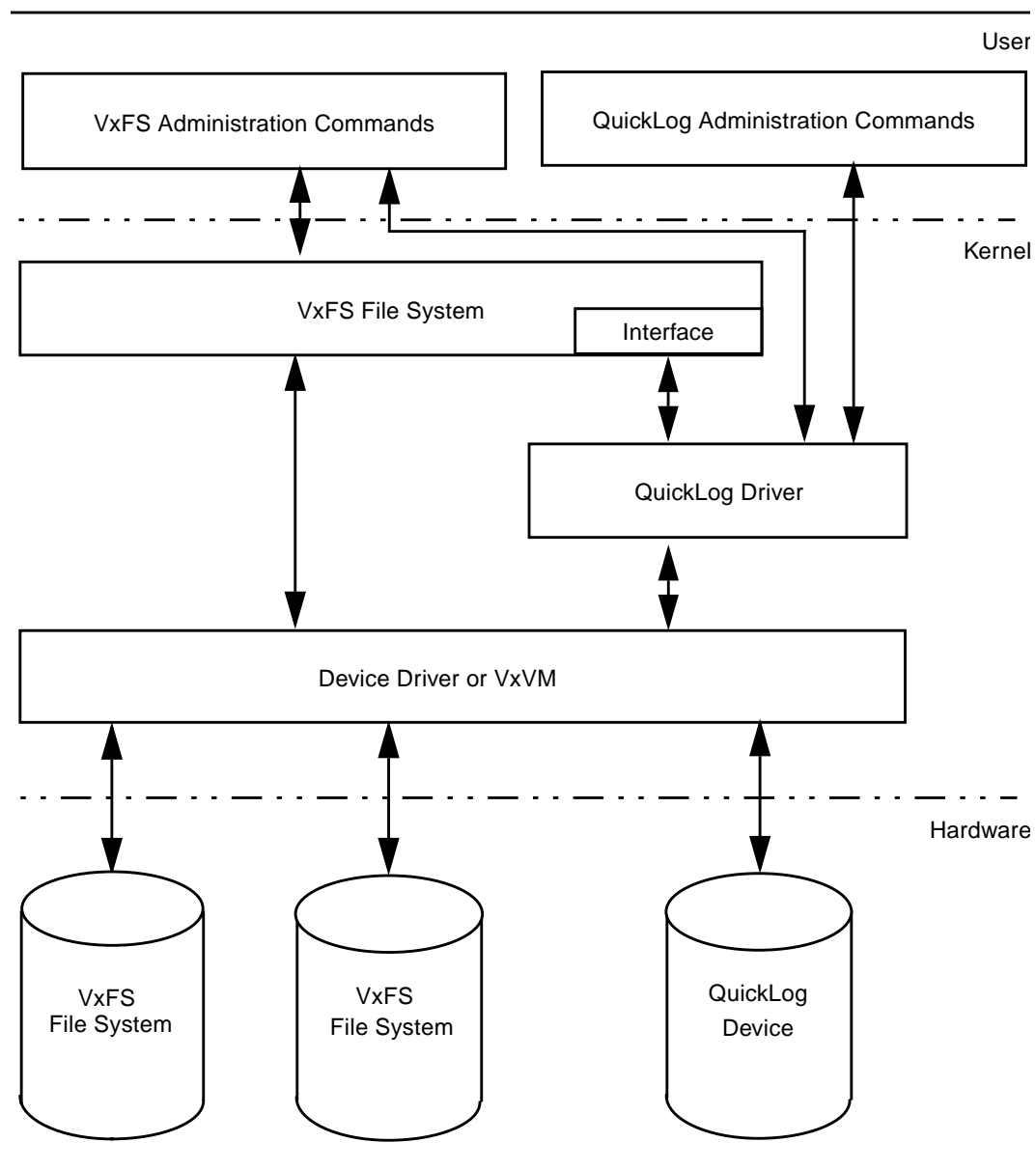
The figure on the following page shows a logical view of QuickLog and how it interfaces with the operating system.

QuickLog Setup

VERITAS QuickLog supports:

- ◆ Up to 63 QuickLog devices
 - Up to 31 local QuickLog devices
 - Up to 32 cluster QuickLog devices
- ◆ Up to 32 VxFS file systems per QuickLog device
- ◆ From one to four QuickLog volumes per QuickLog device (see “[QuickLog Load Balancing](#)” on page 100 for details)
- ◆ Communication between QuickLog and VxFS through an integrated interface

QuickLog Logical View



Creating a QuickLog Device

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

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

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

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

Note As the number of file systems enabled on a QuickLog device increases, the larger the QuickLog device is, the better the performance will be. Because a QuickLog log file is circular, very large logs typically reduce the overhead associated with wrapping when the end of the log file is reached.

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

$$(4 \times 16 \text{ MB}) \times 1.5 = 96 \text{ MB}$$

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

```
# qllogmk -g diskgroup vxlog[x] qlog_volume
```

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

The size of the QuickLog device can be spread out across the one to four QuickLog volumes to be attached (see “[QuickLog Load Balancing](#)” on page 100 for details).

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

Removing a QuickLog Device

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

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

`qlogrm` detaches a QuickLog volume from its QuickLog device. If the QuickLog volume is the only volume attached to the QuickLog device, all file systems that are logging to the QuickLog device must have logging by QuickLog disabled prior to using `qlogrm` (see [“Disabling a QuickLog Device”](#) on page 100 for details).

Use `vxedit` to remove the VxVM volume:

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

VxFS Administration Using QuickLog

Enabling a QuickLog Device

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

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

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

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

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

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

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

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

```
# device    device  mount FS   fsck mount    mount
# to mount to fsck point type pass at boot  options
#
/dev/vx/dsk/vol1 /dev/vx/rdsk/vol1 /vol1 vxfs 1 no qlog=
```



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

Disabling a QuickLog Device

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

```
# qlogdisable /mountpoint
```

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

QuickLog Administration and Troubleshooting

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

QuickLog Load Balancing

QuickLog can perform load balancing when two or more physical volumes are attached to a QuickLog device. QuickLog supports from one to four QuickLog volumes attached to each of the 63 QuickLog devices.

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

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

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

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

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

```
# qlogrm qlog_volume
```

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

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

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

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

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

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

QuickLog Statistics

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

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

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



QuickLog Recovery

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

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

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

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

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

Cluster QuickLog Devices

Cluster QuickLog supports logging of a cluster file system. After a cluster QuickLog device is configured into a cluster, the status of the device is *free*, so the device can be used from any node in the cluster. When a node first accesses the free cluster QuickLog device, that node becomes the *master* of the device. From that point on, only the master node can access the device until the node leaves the cluster or relinquishes mastership of the device. Configuration updates (such as attaching or detaching a QuickLog volume and enabling or disabling file system logging) can only be done from the master. See *The VERITAS SANPoint Foundation Suite Installation and Configuration Guide*, included in the SANPoint Foundation Suite product, for more information.

Introduction

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

- ◆ [Creating a File System](#)
- ◆ [Mounting a File System](#)
- ◆ [Unmounting a File System](#)
- ◆ [Displaying Information on Mounted File Systems](#)
- ◆ [Identifying File System Types](#)
- ◆ [Resizing a File System](#)
- ◆ [Backing Up and Restoring a File System](#)
- ◆ [Using Quotas](#)



Creating a File System

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

Before running `mkfs`, you must 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.

How to Create a File System

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

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

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

See the following manual pages for more information about creating VxFS file systems:

- ◆ `mkfs(1M)`
- ◆ `mkfs_vxfs(1M)`

Example

To create a VxFS file system 12288 sectors in size on VxVM volume, enter:

```
# mkfs -F vxfs /dev/vx/rdisk/diskgroup/volume 12288
```

Information similar to the following displays:

```
version 5 layout
```

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

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



Mounting a File System

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

How to Mount a File System

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

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

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

Mount Options

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

- ◆ Security feature

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

- ◆ Support for large files

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

- ◆ Support for cluster file systems

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

- ◆ Using Storage Checkpoints

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

- ◆ Using databases

If you are using databases with VxFS and if you have installed a license key for the VERITAS Quick I/O for Databases feature, the `mount` command enables Quick I/O by default (the same as specifying the `qio` option). The `noqio` option disables Quick I/O. If you do not have Quick I/O, `mount` ignores the `qio` option. Alternatively, you can increase database performance using the `mount` option `convosync=direct`, which utilizes direct I/O. See “[Quick I/O for Databases](#)” on page 83 for more information.

- ◆ News file systems

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

- ◆ VERITAS QuickLog

If you are using QuickLog, you can improve I/O performance by moving logging to a separate disk device by using `qlog=[dev]`. See “[VERITAS QuickLog](#)” on page 95 for more information.

- ◆ Temporary file systems

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



See “[Choosing mount Command Options](#)” on page 19 and the following manual pages for more information about the `mount` command and its available options:

```
fckptadm(1M)
mount(1M)
mount_vxfs(1M)
vfstab(4)
```

Example

To mount the file system `/dev/vx/dsk/fsvol/vol1` on the `/ext` directory with read/write access and delayed logging, enter:

```
# mount -F vxfs -o delaylog /dev/vx/dsk/fsvol/vol1 /ext
```

How to Edit the `vfstab` File

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

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

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

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

# device # to mount #	device to fsck	mount point	FS type	fsck pass	mount at boot	mount options
# /dev/dsk/c1d0s2	/dev/rdisk/c1d0s2	/usr	ufs	1	yes	—
/proc	—	/proc	proc	—	no	—
fd	—	/dev/fd	fd	—	no	—
swap	—	/tmp	tmpfs	—	yes	—
/dev/dsk/c0t3d0s0	/dev/rdisk/c0t3d0s0	/	ufs	1	no	—
/dev/dsk/c0t3d0s1	—	—	swap	—	no	—
/dev/vx/dsk/fsvol/vol1	/dev/vx/rdisk/fsvol/vol1	/ext	vxfs	1	yes	—



Unmounting a File System

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

How to Unmount a File System

To unmount a file system, use the following syntax:

```
umount special | mount_point
```

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

Example

To unmount the file system `/dev/vx/dsk/fsvol/vol1`, enter:

```
# umount /dev/vx/dsk/fsvol/vol1
```

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

```
# umount -a
```

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

Displaying Information on Mounted File Systems

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

How to Display File System Information

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

```
mount -v
```

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

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

```
mount(1M)
```

```
mount_vxfs(1M)
```

Example

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

```
# mount
/ on /dev/root read/write/setuid on Thu May 26 16:58:24 2001
/proc on /proc read/write on Thu May 26 16:58:25 2001
/dev/fd on /dev/fd read/write on Thu May 26 16:58:26 2001
/tmp on /tmp read/write on Thu May 26 16:59:33 2001
/var/tmp on /var/tmp read/write on Thu May 26 16:59:34 2001
```



Identifying File System Types

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

How to Identify a File System

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

```
fstyp -v special
```

special The character (raw) device.

`-v` Specifies verbose mode.

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

`fstyp(1M)`

`fstyp_vxfs(1M)`

Example

To find out what kind of file system is on the device `/dev/vx/dsk/fsvol/voll`, enter:

```
# fstyp -v /dev/vx/dsk/fsvol/voll
```

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

```
vxfs
magic a501fcf5  version 5  ctime Tue Jun 25 18:29:39 2002
logstart 17  logend 1040
bsize 1024 size 1048576 dsize 1047255  ninode 0  nau 8
defiextsize 64  ilbsize 0  immedien 96  ndaddr 10
aufirst 1049  emap 2  imap 0  iextop 0  istart 0
bstart 34  femap 1051  fimap 0  fiextop 0  fistart 0  fbstart 1083
nindir 2048  aulen 131106  auimlen 0  auemlen 32
aulen 0  aupad 0  aublocks 131072  maxtier 17
inopb 4  inopau 0  ndiripau 0  iaddrlen 8  bshift 10
inoshift 2  bmask fffffffc00  boffmask 3ff  checksum d7938aa1
oltext1 9  oltext2 1041  oltsize 8  checksum2 52a
free 382614  ifree 0
efree 676 413 426 466 612 462 226 112 85 35 14 3 6 5 4 4 0 0
```


Resizing a File System

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

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

How to Extend a File System Using `fsadm`

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

Note If a file system is full, busy, or too fragmented, the resize operation may fail.

To extend a VxFS file system, use the syntax:

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

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

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

Example

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

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



How to Shrink a File System

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

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

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

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

Example

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

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

Caution 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.

How to Reorganize a File System

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

Note If a file system is full or busy, the reorg operation may fail.

To reorganize a VxFS file system, use the syntax:

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

<code>-d</code>	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.
<code>-D</code>	Reports on directory fragmentation.
<code>-e</code>	Minimizes file system fragmentation. Files are reorganized to have the minimum number of extents.
<code>-E</code>	Reports on extent fragmentation.
<code>mount_point</code>	The file system's mount point.
<code>-r rawdev</code>	Specifies the path name of the raw device if there is no entry in <code>/etc/vfstab</code> and <code>fsadm</code> cannot determine the raw device.

Example

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

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



Backing Up and Restoring a File System

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

See “[Online Backup](#)” on page 69 and the following manual pages for more information about the `mount`, `vxdump`, and `vxrestore` commands and their available options:

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

How to Create and Mount a Snapshot File System

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

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

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

Example

To create a snapshot file system of the file system at `/home` on `/dev/vx/dsk/fsvol/vol1` and mount it at `/snapmount`, enter:

```
# mount -F vxfs -o snapof=/dev/vx/dsk/fsvol/vol1, \  
    snapsize=32768 /dev/vx/dsk/fsvol/vol1 /snapmount
```

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

How to Back Up a File System

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

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

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

Example

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

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

How to Restore a File System

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

```
vxrestore [-v] [-x] [filename]
```

<code>-v</code>	Specifies verbose mode.
<code>-x</code>	Extracts the named files from the tape.
<i>filename</i>	The file or directory to restore. If <i>filename</i> is omitted, the root directory (and thus the entire tape) is extracted.

Example

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

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



Using Quotas

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

See “[Quotas](#)” on page 77 and the following manual pages for more information about the `vxquota`, `vxquotaon`, `vxquotaoff`, and `vxedquota` commands and their available options:

- `vxquota(1M)`
- `vxquotaon(1M)`
- `vxquotaoff(1M)`
- `vxedquota(1M)`

How to Turn On Quotas

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

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

```
vxquotaon mount_point
```

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

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

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

Example

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

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

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

```
# mount -F vxfs -o quota /dev/vx/dsk/fsvol/vol1 /mnt
```

How to Set Up User Quotas

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

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

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

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

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

To invoke the quota editor, use the syntax:

```
vxedquota username
```

To modify the time limit, use the syntax:

```
vxedquota -t
```



How to View Quotas

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

```
vxquota -v username
```

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

How to Turn Off Quotas

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

```
vxquotaoff mount_point
```

Example

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

```
# vxquotaoff /mnt
```


Introduction

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

The following topics are covered in this chapter:

- ◆ [File System Response to Problems](#)
 - [Marking an Inode Bad](#)
 - [Disabling Transactions](#)
 - [Disabling a File System](#)
 - [Recovering a Disabled File System](#)
- ◆ [Kernel Messages](#)
 - [Global Message IDs](#)



File System Response to Problems

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

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

Marking an Inode Bad

Inodes can be marked bad if an inode update or a directory-block update fails. In these types of failures, the file system does not know what information is on the disk, and considers all the information that it finds to be invalid. After an inode is marked bad, the kernel still permits access to the file name, but any attempt to access the data in the file or change the inode fails.

Disabling Transactions

If the file system detects an error while writing the intent log, it disables transactions. After transactions are disabled, the files in the file system can still be read or written, but no block or inode frees or allocations, structural changes, directory entry changes, or other changes to metadata are allowed.

Disabling a File System

If an error occurs that compromises the integrity of the file system, VxFS disables itself. If the intent log fails or an inode-list error occurs, the super-block is ordinarily updated (setting the `VX_FULLFSCK` flag) so that the next `fsck` does a full structural check. If this super-block update fails, any further changes to the file system can cause inconsistencies that are undetectable by the intent log replay. To avoid this situation, the file system disables itself.

Recovering a Disabled File System

When the file system is disabled, no data can be written to the disk. Although some minor file system operations still work, most simply return `EIO`. The only thing that can be done when the file system is disabled is to do a `umount` and run a full `fsck`.

Although a log replay may produce a clean file system, do a full structural check to be safe. To execute a full structural check, enter:

```
# fsck -F vxfs -o full -y /dev/vx/rdisk/diskgroup/volume
```

Caution Be careful when running this command. By specifying the `-y` option, `fsck` can make irreversible changes if it performs a full file system check.

The file system usually becomes disabled because of disk errors. Disk failures that disable a file system should be fixed as quickly as possible (see `fsck_vxfs(1M)`).

Kernel Messages

This section lists the VxFS kernel error messages in numerical order. The *Description* subsection for each message describes the problem, the *Action* sub-section suggests possible solutions.

Global Message IDs

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.



Message Number	Message and Definition
001	<p>NOTICE: msgcnt x: vxfs: mesg 001: vx_nospace - <i>mount_point</i> file system full (<i>n</i> block extent)</p> <p>◆ Description</p> <p>The file system is out of space.</p> <p>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.</p> <p>◆ Action</p> <p>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.</p> <p>To remove files, use the <code>find</code> 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 <code>fsadm</code> (see the <code>fsadm_vxfs(1M)</code> manual page).</p>
002	<p>WARNING: msgcnt x: vxfs: mesg 002: vx_snap_strategy - <i>mount_point</i> file system write attempt to read-only file system</p> <p>WARNING: msgcnt x: vxfs: mesg 002: vx_snap_copyblk - <i>mount_point</i> file system write attempt to read-only file system</p> <p>◆ Description</p> <p>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.</p> <p>◆ Action</p> <p>The file system was not written, so no action is required. Report this as a bug to your customer support organization.</p>

Message Number	Message and Definition
003, 004, 005	<p>WARNING: msgcnt x: vxfs: mesg 003: vx_mapbad - <i>mount_point</i> file system free extent bitmap in au <i>aun</i> marked bad</p> <p>WARNING: msgcnt x: vxfs: mesg 004: vx_mapbad - <i>mount_point</i> file system free inode bitmap in au <i>aun</i> marked bad</p> <p>WARNING: msgcnt x: vxfs: mesg 005: vx_mapbad - <i>mount_point</i> file system inode extended operation bitmap in au <i>aun</i> marked bad</p> <p>◆ Description</p> <p>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 <i>df</i> may report an adequate amount of free space.</p> <p>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 <i>fsdb</i> to change the file system.</p> <p>The <i>VX_FULLFCK</i> flag is set. If the map that failed was a free extent bitmap, and the <i>VX_FULLFCK</i> flag can't be set, then the file system is disabled.</p> <p>◆ Action</p> <p>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 <i>fsck</i> to run a full structural check.</p>
006, 007	<p>WARNING: msgcnt x: vxfs: mesg 006: vx_sumupd - <i>mount_point</i> file system summary update in au <i>aun</i> failed</p> <p>WARNING: msgcnt x: vxfs: mesg 007: vx_sumupd - <i>mount_point</i> file system summary update in inode au <i>iaun</i> failed</p> <p>◆ Description</p> <p>An I/O error occurred while writing the allocation unit or inode allocation unit bitmap summary to disk. This sets the <i>VX_FULLFCK</i> flag on the file system. If the <i>VX_FULLFCK</i> flag can't be set, the file system is disabled.</p> <p>◆ Action</p> <p>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 <i>fsck</i> to run a full structural check.</p>



Message Number	Message and Definition
008, 009	<p>WARNING: msgcnt x: vxfs: msg 008: vx_direrr - <i>mount_point</i> file system inode <i>inumber</i> block <i>blkno</i> error <i>errno</i></p> <p>WARNING: msgcnt x: vxfs: msg 009: vx_direrr - <i>mount_point</i> file system inode <i>inumber</i> immediate directory error <i>errno</i></p> <p>◆ Description</p> <p>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.</p> <p>The VX_FULLFSCK flag is set in the super-block so that <code>fsck</code> will do a full structural check the next time it is run.</p> <p>◆ Action</p> <p>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 <code>fsck</code> to run a full structural check.</p>
010	<p>WARNING: msgcnt x: vxfs: msg 010: vx_ialloc - <i>mount_point</i> file system inode <i>inumber</i> not free</p> <p>◆ Description</p> <p>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.</p> <p>The VX_FULLFSCK flag is set in the super-block so that <code>fsck</code> will do a full structural check the next time it is run.</p> <p>◆ Action</p> <p>Unmount the file system and use <code>fsck</code> to run a full structural check.</p>

Message Number	Message and Definition
011	<p>NOTICE: msgcnt x: vxfs: msg 011: vx_noinode - <i>mount_point</i> file system out of inodes</p> <p>◆ Description</p> <p>The file system is out of inodes.</p> <p>◆ Action</p> <p>Monitor the free inodes in the file system. If the file system is getting full, create more inodes either by removing files or by expanding the file system. File system resizing is described in “Online System Administration” on page 8, and in the <code>fsadm_vxfs(1M)</code> online manual page.</p>
012	<p>WARNING: msgcnt x: vxfs: msg 012: vx_iget - <i>mount_point</i> file system invalid inode number <i>inumber</i></p> <p>◆ Description</p> <p>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.</p> <p>The <code>VX_FULFSCK</code> flag is set in the super-block so that <code>fsck</code> will do a full structural check the next time it is run.</p> <p>◆ Action</p> <p>Unmount the file system and use <code>fsck</code> to run a full structural check.</p>
013	<p>WARNING: msgcnt x: vxfs: msg 013: vx_iposition - <i>mount_point</i> file system inode <i>inumber</i> invalid inode list extent</p> <p>◆ Description</p> <p>For a Version 2 and above disk layout, the inode list is dynamically allocated. When the kernel tries to read an inode, it must look up the location of the inode in the inode list file. If the kernel finds a bad extent, the inode can't be accessed. All of the inode list extents are validated when the file system is mounted, so if the kernel finds a bad extent, the integrity of the inode list is questionable. This is a very serious error.</p> <p>The <code>VX_FULFSCK</code> flag is set in the super-block and the file system is disabled.</p> <p>◆ Action</p> <p>Unmount the file system and use <code>fsck</code> to run a full structural check.</p>



Message Number	Message and Definition
014	<p>WARNING: msgcnt x: vxfs: mesg 014: vx_iget - inode table overflow</p> <p>◆ Description</p> <p>All the system in-memory inodes are busy and an attempt was made to use a new inode.</p> <p>◆ Action</p> <p>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 <code>vxfs_ninode</code> parameter in the kernel (see “Kernel Tunables” on page 27).</p>
015	<p>WARNING: msgcnt x: vxfs: mesg 015: vx_ibadinactive - <i>mount_point</i> file system can't mark inode <i>inumber</i> bad</p> <p>WARNING: msgcnt x: vxfs: mesg 015: vx_ilisterr - <i>mount_point</i> file system can't mark inode <i>inumber</i> bad</p> <p>◆ Description</p> <p>An attempt to mark an inode bad on disk, and the super-block update to set the <code>VX_FULLFSCK</code> 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.</p> <p>◆ Action</p> <p>Unmount the file system and use <code>fsck</code> 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.</p>
016	<p>WARNING: msgcnt x: vxfs: mesg 016: vx_ilisterr - <i>mount_point</i> file system error reading inode <i>inumber</i></p> <p>◆ Description</p> <p>An I/O error occurred while reading the inode list. The <code>VX_FULLFSCK</code> flag is set.</p> <p>◆ Action</p> <p>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 <code>fsck</code> to run a full structural check.</p>

Message Number	Message and Definition
017	<p>WARNING: msgcnt x: vxfs: mesg 017: vx_attr_getblk - <i>mount_point</i> file system inode <i>inumber</i> marked bad in core</p> <p>WARNING: msgcnt x: vxfs: mesg 017: vx_attr_iget - <i>mount_point</i> file system inode <i>inumber</i> marked bad in core</p> <p>WARNING: msgcnt x: vxfs: mesg 017: vx_attr_indadd - <i>mount_point</i> file system inode <i>inumber</i> marked bad in core</p> <p>WARNING: msgcnt x: vxfs: mesg 017: vx_attr_indtrunc - <i>mount_point</i> file system inode <i>inumber</i> marked bad in core</p> <p>WARNING: msgcnt x: vxfs: mesg 017: vx_attr_iremove - <i>mount_point</i> file system inode <i>inumber</i> marked bad in core</p> <p>WARNING: msgcnt x: vxfs: mesg 017: vx_bmap - <i>mount_point</i> file system inode <i>inumber</i> marked bad in core</p> <p>WARNING: msgcnt x: vxfs: mesg 017: vx_bmap_indirect_ext4 - <i>mount_point</i> file system inode <i>inumber</i> marked bad in core</p> <p>WARNING: msgcnt x: vxfs: mesg 017: vx_delbuf_flush - <i>mount_point</i> file system inode <i>inumber</i> marked bad in core</p> <p>WARNING: msgcnt x: vxfs: mesg 017: vx_dio_iovec - <i>mount_point</i> file system inode <i>inumber</i> marked bad in core</p> <p>WARNING: msgcnt x: vxfs: mesg 017: vx_dirbread - <i>mount_point</i> file system inode <i>inumber</i> marked bad in core</p> <p>WARNING: msgcnt x: vxfs: mesg 017: vx_dircreate - <i>mount_point</i> file system inode <i>inumber</i> marked bad in core</p> <p>WARNING: msgcnt x: vxfs: mesg 017: vx_dirlook - <i>mount_point</i> file system inode <i>inumber</i> marked bad in core</p> <p>WARNING: msgcnt x: vxfs: mesg 017: vx_doextop_iau - <i>mount_point</i> file system inode <i>inumber</i> marked bad in core</p> <p>WARNING: msgcnt x: vxfs: mesg 017: vx_doextop_now - <i>mount_point</i> file system inode <i>inumber</i> marked bad in core</p> <p>WARNING: msgcnt x: vxfs: mesg 017: vx_do_getpage - <i>mount_point</i> file system inode <i>inumber</i> marked bad in core</p> <p>WARNING: msgcnt x: vxfs: mesg 017: vx_enter_ext4 - <i>mount_point</i> file system inode <i>inumber</i> marked bad in core</p> <p>WARNING: msgcnt x: vxfs: mesg 017: vx_exttrunc - <i>mount_point</i> file system inode <i>inumber</i> marked bad in core</p> <p>WARNING: msgcnt x: vxfs: mesg 017: vx_get_alloc - <i>mount_point</i> file system inode <i>inumber</i> marked bad in core</p>



Message Number	Message and Definition
017 (continued)	<p>WARNING: msgcnt x: vxfs: msg 017: vx_ilisterr - <i>mount_point</i> file system inode <i>inumber</i> marked bad in core</p> <p>WARNING: msgcnt x: vxfs: msg 017: vx_indtrunc - <i>mount_point</i> file system inode <i>inumber</i> marked bad in core</p> <p>WARNING: msgcnt x: vxfs: msg 017: vx_iread - <i>mount_point</i> file system inode <i>inumber</i> marked bad in core</p> <p>WARNING: msgcnt x: vxfs: msg 017: vx_iremove - <i>mount_point</i> file system inode <i>inumber</i> marked bad in core</p> <p>WARNING: msgcnt x: vxfs: msg 017: vx_iremove_attr - <i>mount_point</i> file system inode <i>inumber</i> marked bad in core</p> <p>WARNING: msgcnt x: vxfs: msg 017: vx_logwrite_flush - <i>mount_point</i> file system inode <i>inumber</i> marked bad in core</p> <p>WARNING: msgcnt x: vxfs: msg 017: vx_oltmount_iget - <i>mount_point</i> file system inode <i>inumber</i> marked bad in core</p> <p>WARNING: msgcnt x: vxfs: msg 017: vx_overlay_bmap - <i>mount_point</i> file system inode <i>inumber</i> marked bad in core</p> <p>WARNING: msgcnt x: vxfs: msg 017: vx_readnomap - <i>mount_point</i> file system inode <i>inumber</i> marked bad in core</p> <p>WARNING: msgcnt x: vxfs: msg 017: vx_reorg_trunc - <i>mount_point</i> file system inode <i>inumber</i> marked bad in core</p> <p>WARNING: msgcnt x: vxfs: msg 017: vx_stablestore - <i>mount_point</i> file system inode <i>inumber</i> marked bad in core</p> <p>WARNING: msgcnt x: vxfs: msg 017: vx_tranitimes - <i>mount_point</i> file system inode <i>inumber</i> marked bad in core</p> <p>WARNING: msgcnt x: vxfs: msg 017: vx_trunc - <i>mount_point</i> file system inode <i>inumber</i> marked bad in core</p> <p>WARNING: msgcnt x: vxfs: msg 017: vx_write_alloc2 - <i>mount_point</i> file system inode <i>inumber</i> marked bad in core</p> <p>WARNING: msgcnt x: vxfs: msg 017: vx_write_default - <i>mount_point</i> file system inode <i>inumber</i> marked bad in core</p> <p>WARNING: msgcnt x: vxfs: msg 017: vx_zero_alloc - <i>mount_point</i> file system inode <i>inumber</i> marked bad in core</p>

Message Number	Message and Definition
017 (continued)	<div><div>◆ Description</div><p>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 <code>mount</code> command <code>ioerror</code> option is set to <code>disable</code>, or there is subsequent I/O failure when updating the inode on disk. No further operations can be performed on the inode.</p><p>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.</p><p>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 <code>fsdb</code> to change the file system.</p><p>The <code>VX_FULFSCK</code> flag is set in the super-block so <code>fsck</code> will do a full structural check the next time it is run.</p><div>◆ Action</div><p>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 <code>fsck</code> unless the <code>VX_FULFSCK</code> flag is set for the file system.</p></div>



Message Number	Message and Definition
019	<p>WARNING: msgcnt x: vxfs: mesg 019: vx_log_add - <i>mount_point</i> file system log overflow</p> <p>◆ Description</p> <p>Log ID overflow. When the log ID reaches <code>VX_MAXLOGID</code> (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 <code>VX_DISLOGID</code> (approximately <code>VX_MAXLOGID</code> 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.</p> <p>◆ Action</p> <p>Unmount the file system and use <code>fsck</code> to run a full structural check.</p>
020	<p>WARNING: msgcnt x: vxfs: mesg 020: vx_logerr - <i>mount_point</i> file system log error <i>errno</i></p> <p>◆ Description</p> <p>Intent log failed. The kernel will try to set the <code>VX_FULLFSCK</code> and <code>VX_LOGBAD</code> flags in the super-block to prevent running a log replay. If the super-block can't be updated, the file system is disabled.</p> <p>◆ Action</p> <p>Unmount the file system and use <code>fsck</code> 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.</p>

Message Number	Message and Definition
021	<p>WARNING: msgcnt x: vxfs: msg 021: vx_fs_init - <i>mount_point</i> file system validation failure</p> <p>◆ Description</p> <p>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.</p> <p>If there is any I/O problem or the structure is inconsistent, the kernel sets the <code>VX_FULLFCK</code> flag and the mount fails.</p> <p>If the error isn't related to an I/O failure, this may have occurred because a user or process has written directly to the device or used <code>fsdb</code> to change the file system.</p> <p>◆ Action</p> <p>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 <code>fsck</code> to run a full structural check.</p>
022	<p>WARNING: msgcnt x: vxfs: msg 022: vx_mountroot - root file system remount failed</p> <p>◆ Description</p> <p>The remount of the root file system failed. The system will not be usable if the root file system can't be remounted for read/write access.</p> <p>When a VERITAS root file system is first mounted, it is mounted for read-only access. After <code>fsck</code> is run, the file system is remounted for read/write access. The remount fails if <code>fsck</code> completed a resize operation or modified a file that was opened before the <code>fsck</code> was run. It also fails if an I/O error occurred during the remount.</p> <p>Usually, the system halts or reboots automatically.</p> <p>◆ Action</p> <p>Reboot the system. The system either remounts the root cleanly or runs a full structural <code>fsck</code> and remounts cleanly. If the remount succeeds, no further action is necessary.</p> <p>Check the console log for I/O errors. If the disk has failed, replace it before the file system is mounted for write access.</p> <p>If the system won't come up and a full structural <code>fsck</code> hasn't been run, reboot the system on a backup root and manually run a full structural <code>fsck</code>. If the problem persists after the full structural <code>fsck</code> and there are no I/O errors, contact your customer support organization.</p>



Message Number	Message and Definition
023	<p>WARNING: msgcnt x: vxfs: mesg 023: vx_unmountroot - root file system is busy and can't be unmounted cleanly</p> <p>◆ Description</p> <p>There were active files in the file system and they caused the unmount to fail. When the system is halted, the root file system is unmounted. This happens occasionally when a process is hung and it can't be killed before unmounting the root.</p> <p>◆ Action</p> <p><code>fsck</code> will run when the system is rebooted. It should clean up the file system. No other action is necessary.</p> <p>If the problem occurs every time the system is halted, determine the cause and contact your customer support organization.</p>
024	<p>WARNING: msgcnt x: vxfs: mesg 024: vx_cutwait - <i>mount_point</i> file system current usage table update error</p> <p>◆ Description</p> <p>Update to the current usage table (CUT) failed.</p> <p>For a Version 2 disk layout, the CUT contains a fileset version number and total number of blocks used by each fileset.</p> <p>The <code>VX_FULLFSCK</code> flag is set in the super-block. If the super-block can't be written, the file system is disabled.</p> <p>◆ Action</p> <p>Unmount the file system and use <code>fsck</code> to run a full structural check.</p>
025	<p>WARNING: msgcnt x: vxfs: mesg 025: vx_wsUPER - <i>mount_point</i> file system superblock update failed</p> <p>◆ Description</p> <p>An I/O error occurred while writing the super-block during a resize operation. The file system is disabled.</p> <p>◆ Action</p> <p>Unmount the file system and use <code>fsck</code> 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.</p>

Message Number	Message and Definition
026	<p>WARNING: msgcnt x: vxfs: msg 026: vx_snap_copyblk - <i>mount_point</i> primary file system read error</p> <p>◆ Description</p> <p>Snapshot file system error.</p> <p>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.</p> <p>◆ Action</p> <p>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.</p>
027	<p>WARNING: msgcnt x: vxfs: msg 027: vx_snap_bpcopy - <i>mount_point</i> snapshot file system write error</p> <p>◆ Description</p> <p>A write to the snapshot file system failed.</p> <p>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.</p> <p>◆ Action</p> <p>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.</p>



Message Number	Message and Definition
028	<p>WARNING: msgcnt x: vxfs: mesg 028: vx_snap_alloc - <i>mount_point</i> snapshot file system out of space</p> <p>◆ Description</p> <p>The snapshot file system ran out of space to store changes.</p> <p>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.</p> <p>◆ Action</p> <p>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.</p> <p>Rerun any backups that failed when the error occurred.</p>
029, 030	<p>WARNING: msgcnt x: vxfs: mesg 029: vx_snap_getbp - <i>mount_point</i> snapshot file system block map write error</p> <p>WARNING: msgcnt x: vxfs: mesg 030: vx_snap_getbp - <i>mount_point</i> snapshot file system block map read error</p> <p>◆ Description</p> <p>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.</p> <p>◆ Action</p> <p>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.</p>

Message Number	Message and Definition
031	<p>WARNING: msgcnt x: vxfs: mesg 031: vx_disable - <i>mount_point</i> file system disabled</p> <p>◆ Description</p> <p>File system disabled, preceded by a message that specifies the reason. This usually indicates a serious disk problem.</p> <p>◆ Action</p> <p>Unmount the file system and use <code>fsck</code> 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.</p>
032	<p>WARNING: msgcnt x: vxfs: mesg 032: vx_disable - <i>mount_point</i> snapshot file system disabled</p> <p>◆ Description</p> <p>Snapshot file system disabled, preceded by a message that specifies the reason.</p> <p>◆ Action</p> <p>Unmount the snapshot file system, correct the problem specified by the message, and rerun any backups that failed due to the error.</p>
033	<p>WARNING: msgcnt x: vxfs: mesg 033: vx_check_badblock - <i>mount_point</i> file system had an I/O error, setting VX_FULLFSCK</p> <p>◆ Description</p> <p>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 <code>VX_FULLFSCK</code> flag as a precautionary measure. Since no other error has set the <code>VX_FULLFSCK</code> flag, the failure probably occurred on a data block.</p> <p>◆ Action</p> <p>Unmount the file system and use <code>fsck</code> 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.</p>



Message Number	Message and Definition
034	<p>WARNING: msgcnt <i>x</i>: vxfs: mesg 034: vx_resetlog - <i>mount_point</i> file system can't reset log</p> <p>◆ Description</p> <p>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.</p> <p>◆ Action</p> <p>Unmount the file system and use <code>fsck</code> 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.</p>
035	<p>WARNING: msgcnt <i>x</i>: vxfs: mesg 035: vx_inactive - <i>mount_point</i> file system inactive of locked inode <i>inumber</i></p> <p>◆ Description</p> <p><code>VOP_INACTIVE</code> was called for an inode while the inode was being used. This should never happen, but if it does, the file system is disabled.</p> <p>◆ Action</p> <p>Unmount the file system and use <code>fsck</code> to run a full structural check. Report as a bug to your customer support organization.</p>
036	<p>WARNING: msgcnt <i>x</i>: vxfs: mesg 036: vx_lctbad - <i>mount_point</i> file system link count table <i>lctnumber</i> bad</p> <p>◆ Description</p> <p>Update to the link count table (LCT) failed.</p> <p>For a Version 2 and above disk layout, the LCT contains the link count for all the structural inodes. The <code>VX_FULLFSCK</code> flag is set in the super-block. If the super-block can't be written, the file system is disabled.</p> <p>◆ Action</p> <p>Unmount the file system and use <code>fsck</code> to run a full structural check.</p>

Message Number	Message and Definition
037	<p>WARNING: msgcnt x: vxfs: msg 037: vx_metaioerr - file system meta data error</p> <p>◆ Description</p> <p>A read or a write error occurred while accessing file system metadata. The full <code>fsck</code> flag on the file system was set. The message specifies whether the disk I/O that failed was a read or a write.</p> <p>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.</p> <p>◆ Action</p> <p>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 <code>fsck</code> to run a full structural check (possibly with loss of data).</p> <p>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 <code>fsck</code> is run since <code>fsck</code> 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.</p>



Message Number	Message and Definition
038	<p>WARNING: msgcnt x: vxfs: mesg 038: vx_dataioerr - file system file data error</p> <p>◆ Description</p> <p>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.</p> <p>◆ Action</p> <p>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 <code>ncheck(1M)</code> manual page for more information.)</p> <p>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.</p>
039	<p>WARNING: msgcnt x: vxfs: mesg 039: vx_writesuper - file system super-block write error</p> <p>◆ Description</p> <p>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 <code>fsck</code> 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 <code>sync</code> operation, no other action is taken.</p> <p>◆ Action</p> <p>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 <code>fsck</code> to run a full structural check.</p> <p>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.</p>

Message Number	Message and Definition
040	<p>WARNING: msgcnt x: vxfs: mesg 040: vx_dqbad - <i>mount_point</i> file system user group quota file update error for id <i>id</i></p> <p>◆ Description</p> <p>An update to the user quotas file failed for the user ID.</p> <p>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 <code>VX_FULLFCK</code> flag is set in the super-block. If the super-block cannot be written, the file system is disabled.</p> <p>◆ Action</p> <p>Unmount the file system and use <code>fck</code> 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.</p>
041	<p>WARNING: msgcnt x: vxfs: mesg 041: vx_dqget - <i>mount_point</i> file system user group quota file can't read quota for id <i>id</i></p> <p>◆ Description</p> <p>A read of the user quotas file failed for the <code>uid</code>.</p> <p>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 <code>VX_FULLFCK</code> flag is set in the super-block. If the super-block cannot be written, the file system is disabled.</p> <p>◆ Action</p> <p>Unmount the file system and use <code>fck</code> 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.</p>
042	<p>WARNING: msgcnt x: vxfs: mesg 042: vx_bsdquotaupdate - <i>mount_point</i> file system user group id disk limit reached</p> <p>◆ Description</p> <p>The hard limit on blocks was reached. Further attempts to allocate blocks for files owned by the user will fail.</p> <p>◆ Action</p> <p>Remove some files to free up space.</p>



Message Number	Message and Definition
043	<p>WARNING: msgcnt x: vxfs: mesg 043: vx_bsdquotaupdate - <i>mount_point</i> file system user group id disk quota exceeded too long</p> <p>◆ Description</p> <p>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.</p> <p>◆ Action</p> <p>Remove some files to free up space.</p>
044	<p>WARNING: msgcnt x: vxfs: mesg 044: vx_bsdquotaupdate - <i>mount_point</i> file system user group id disk quota exceeded</p> <p>◆ Description</p> <p>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.</p> <p>◆ Action</p> <p>Remove some files to free up space.</p>
045	<p>WARNING: msgcnt x: vxfs: mesg 045: vx_bsdiquotaupdate - <i>mount_point</i> file system user group id inode limit reached</p> <p>◆ Description</p> <p>The hard limit on inodes was exceeded. Further attempts to create files owned by the user will fail.</p> <p>◆ Action</p> <p>Remove some files to free inodes.</p>
046	<p>WARNING: msgcnt x: vxfs: mesg 046: vx_bsdiquotaupdate - <i>mount_point</i> file system user group id inode quota exceeded too long</p> <p>◆ Description</p> <p>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.</p> <p>◆ Action</p> <p>Remove some files to free inodes.</p>

Message Number	Message and Definition
047	<p>WARNING: msgcnt x: vxfs: mesg 047: vx_bsdiquotaupdate - warning: <i>mount_point</i> file system user group id inode quota exceeded</p> <p>◆ Description</p> <p>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.</p> <p>◆ Action</p> <p>Remove some files to free inodes.</p>
048, 049	<p>WARNING: msgcnt x: vxfs: mesg 048: vx_dqread - warning: <i>mount_point</i> file system external user group quota file read failed</p> <p>WARNING: msgcnt x: vxfs: mesg 049: vx_dqwrite - warning: <i>mount_point</i> file system external user group quota file write failed</p> <p>◆ Description</p> <p>To maintain reliable usage counts, VxFS maintains the user quotas file as a structural file in the structural fileset. 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.</p> <p>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.</p> <p>◆ Action</p> <p>Determine the reason for the failure on the external quotas file and correct it. Recreate the quotas file.</p>
050	<p>WARNING: msgcnt x: vxfs: mesg 050: vx_ldlogwrite - <i>mount_point</i> file system log write failed</p> <p>◆ Description</p> <p>A write to VERITAS QuickLog log failed. This marks the log bad and sets the full file system check flag in the super block.</p> <p>◆ Action</p> <p>No immediate action required. When the file system is unmounted, run a full file system check using <code>fsck</code> before mounting it again.</p>



Message Number	Message and Definition
051	<p>WARNING: msgcnt x: vxfs: msg 051: vx_ldlog_start - <i>mount_point</i> file system log start failed</p> <p>◆ Description</p> <p>vx_ldlog_start failed. QuickLog logging is disabled and file system continues to use its own log.</p> <p>◆ Action</p> <p>No corrective action required on the file system. Determine why the log didn't start and do administrative tasks on QuickLog (for more information on QuickLog, see “VERITAS QuickLog” on page 95”).</p>
052	<p>WARNING: msgcnt x: vxfs: msg 052: vx_ldlog_stop - <i>mount_point</i> file system log stop failed</p> <p>◆ Description</p> <p>QuickLog copies the log back to the file system after stopping logging activity. If the stop failed, VxFS treats the failure as the log going bad.</p> <p>◆ Action</p> <p>No immediate action required. When the file system is unmounted, run a full file system check using <code>fsck</code> before mounting it again.</p>
053	<p>WARNING: msgcnt x: vxfs: msg 053: vx_ldlog_suspend - <i>mount_point</i> file system log suspend failed</p> <p>◆ Description</p> <p>When the file system is frozen, QuickLog is suspended; it is activated again on thaw. If this operation fails, the kernel marks the log bad and sets the full file system check flag in the super block.</p> <p>◆ Action</p> <p>No immediate action required. When the file system is unmounted, run a full file system check using <code>fsck</code> before mounting it again.</p>

Message Number	Message and Definition
054	<p>WARNING: msgcnt x: vxfs: mesg 054: vx_ldlog_resume - <i>mount_point</i> file system log resume failed</p> <p>◆ Description</p> <p>When the file system is thawed, QuickLog must be resumed. If this operation fails, the kernel marks the log bad and sets the full file system check flag in the super block.</p> <p>◆ Action</p> <p>No immediate action required. When the file system is unmounted, run a full file system check using <code>fsck</code> before mounting it again.</p>
056	<p>WARNING: msgcnt x: vxfs: mesg 056: vx_mapbad - <i>mount_point</i> file system extent allocation unit state bitmap number <i>number</i> marked bad</p> <p>◆ Description</p> <p>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 <code>df</code> may report an adequate amount of free space.</p> <p>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 <code>fsdb</code> to change the file system.</p> <p>The <code>VX_FULFSCCK</code> flag is set. If the <code>VX_FULFSCCK</code> flag can't be set, the file system is disabled.</p> <p>◆ Action</p> <p>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 <code>fsck</code> to run a full structural check.</p>



Message Number	Message and Definition
057	<p>WARNING: msgcnt x: vxfs: msg 057: vx_esum_bad - <i>mount_point</i> file system extent allocation unit summary number <i>number</i> marked bad</p> <p>◆ Description</p> <p>An I/O error occurred reading or writing an extent allocation unit summary. The <code>VX_FULFSCK</code> flag is set. If the <code>VX_FULFSCK</code> flag can't be set, the file system is disabled.</p> <p>◆ Action</p> <p>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 <code>fsck</code> to run a full structural check.</p>
058	<p>WARNING: msgcnt x: vxfs: msg 058: vx_isum_bad - <i>mount_point</i> file system inode allocation unit summary number <i>number</i> marked bad</p> <p>◆ Description</p> <p>An I/O error occurred reading or writing an inode allocation unit summary. The <code>VX_FULFSCK</code> flag is set. If the <code>VX_FULFSCK</code> flag cannot be set, the file system is disabled.</p> <p>◆ Action</p> <p>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 <code>fsck</code> to run a full structural check.</p>
059	<p>WARNING: msgcnt x: vxfs: msg 059: vx_snap_getbitbp - <i>mount_point</i> snapshot file system bitmap write error</p> <p>◆ Description</p> <p>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.</p> <p>◆ Action</p> <p>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.</p>

Message Number	Message and Definition
060	<p>WARNING: msgcnt x: vxfs: mesg 060: vx_snap_getbitbp - <i>mount_point</i> snapshot file system bitmap read error</p> <p>◆ Description</p> <p>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.</p> <p>◆ Action</p> <p>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.</p>
061	<p>WARNING: msgcnt x: vxfs: mesg 061: vx_resize - <i>mount_point</i> file system remount failed</p> <p>◆ Description</p> <p>During a file system resize, the remount to the new size failed. The <code>VX_FULLFSCK</code> flag is set and the file system is disabled.</p> <p>◆ Action</p> <p>Unmount the file system and use <code>fsck</code> to run a full structural check. After the check, the file system shows the new size.</p>
062	<p>NOTICE: msgcnt x: vxfs: mesg 062: vx_attr_creatop - invalid disposition returned by attribute driver</p> <p>◆ Description</p> <p>A registered extended attribute intervention routine returned an invalid return code to the VxFS driver during extended attribute inheritance.</p> <p>◆ Action</p> <p>Determine which vendor supplied the registered extended attribute intervention routine and contact their customer support organization.</p>
063	<p>WARNING: msgcnt x: vxfs: mesg 063: vx_fset_markbad - <i>mount_point</i> file system <i>mount_point</i> fileset (index <i>number</i>) marked bad</p> <p>◆ Description</p> <p>An error occurred while reading or writing a fileset structure. <code>VX_FULLFSCK</code> flag is set. If the <code>VX_FULLFSCK</code> flag can't be set, the file system is disabled.</p> <p>◆ Action</p> <p>Unmount the file system and use <code>fsck</code> to run a full structural check.</p>



Message Number	Message and Definition
064	<p>WARNING: msgcnt x: vxfs: mesg 064: vx_ivalidate - <i>mount_point</i> file system inode <i>number</i> version number exceeds fileset's</p> <p>◆ Description</p> <p>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.</p> <p>VX_FULFCK flag is set. If the VX_FULFCK flag can't be set, the file system is disabled.</p> <p>◆ Action</p> <p>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 <code>fsck</code> to run a full structural check.</p>
066	<p>NOTICE: msgcnt x: vxfs: mesg 066: DMAPI mount event - <i>buffer</i></p> <p>◆ Description</p> <p>An HSM (Hierarchical Storage Management) agent responded to a DMAPI mount event and returned a message in <i>buffer</i>.</p> <p>◆ Action</p> <p>Consult the HSM product documentation for the appropriate response to the message.</p>
067	<p>WARNING: msgcnt x: vxfs: mesg 067: mount of <i>device_path</i> requires HSM agent</p> <p>◆ Description</p> <p>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.</p> <p>◆ Action</p> <p>Restart the HSM agent and try to mount the file system again.</p>

Message Number	Message and Definition
068	<p>WARNING: msgcnt x: vxfs: mesg 068: ncsiz parameter is greater than 80% of the vxfs_ninode parameter; increasing the value of vxfs:vxfs_ninode</p> <p>◆ Description</p> <p>The value auto-tuned for the vxfs_ninode parameter is less than 125% of the ncsiz parameter.</p> <p>◆ Action</p> <p>To prevent this message from occurring, set vxfs_ninode to at least 125% of the value of ncsiz. The best way to do this is to adjust ncsiz down, rather than adjusting vxfs_ninode up. For more information on performance and tuning, see “VxFS Performance: Creating, Mounting, and Tuning File Systems” on page 17.</p>
069	<p>WARNING: msgcnt x: vxfs: mesg 069: memory usage specified by the vxfs:vxfs_ninode and vxfs:vx_bc_bufhwm parameters exceeds available memory; the system may hang under heavy load</p> <p>◆ Description</p> <p>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.</p> <p>◆ Action</p> <p>To avoid a system hang, reduce the value of one or both parameters to less than 50% of physical memory or to 66% of kernel virtual memory. For more information on performance and tuning, see “VxFS Performance: Creating, Mounting, and Tuning File Systems” on page 17.</p>
070	<p>WARNING: msgcnt x: vxfs: mesg 070: checkpoint <i>checkpoint_name</i> removed from file system <i>mount_point</i></p> <p>◆ Description</p> <p>The file system ran out of space while updating a Storage Checkpoint. The Storage Checkpoint was removed to allow the operation to complete.</p> <p>◆ Action</p> <p>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 <i>fsadm_vxfs(1M)</i> manual page more information.</p>



Message Number	Message and Definition
071	<p>NOTICE: msgcnt x: vxfs: mesg 071: cleared data I/O error flag in <i>mount_point</i> file system</p> <p>◆ Description</p> <p>The user data I/O error flag was reset when the file system was mounted. This message indicates that a read or write error occurred (see Message Number 038) while the file system was previously mounted.</p> <p>◆ Action</p> <p>Informational only, no action required.</p>
075	<p>WARNING: msgcnt x: vxfs: mesg 075: replay fsck failed for <i>mount_point</i> file system</p> <p>◆ Description</p> <p>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.</p> <p>◆ Action</p> <p>Unmount the file system from the cluster. Use <code>fsck</code> to run a full structural check and mount the file system again.</p>
076	<p>NOTICE: msgcnt x: vxfs: mesg 076: checkpoint asynchronous operation on <i>mount_point</i> file system still in progress</p> <p>◆ Description</p> <p>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.</p> <p>◆ Action</p> <p>The operation may take a considerable length of time. You can do a forced unmount (see <code>umount_vxfs(1M)</code>), or simply wait for the operation to complete so file system can be unmounted cleanly.</p>

Message Number	Message and Definition
077	<p>WARNING: msgcnt x: vxfs: msg 077: vx_fshdchange - <i>mount_point</i> file system <i>number</i> fileset, fileset header: checksum failed</p> <p>◆ Description</p> <p>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.</p> <p>◆ Action</p> <p>Unmount the file system and use <code>fsck</code> to run a full structural check.</p>
078	<p>WARNING: msgcnt x: vxfs: msg 078: vx_ilealloc - <i>mount_point</i> file system <i>mount_point</i> fileset (index <i>number</i>) ilist corrupt</p> <p>◆ Description</p> <p>The inode list for the fileset was corrupted and the corruption was detected while allocating new inodes. The failed system call returns an <code>ENOSPC</code> error. Any subsequent inode allocations will fail unless a sufficient number of files are removed.</p> <p>◆ Action</p> <p>Unmount the file system and use <code>fsck</code> to run a full structural check.</p>



Message Number	Message and Definition
079	<p>WARNING: msgcnt x: vxfs: mesg 017: vx_attr_getblk - <i>mount_point</i> file system inode <i>inumber</i> marked bad on disk</p> <p>WARNING: msgcnt x: vxfs: mesg 017: vx_attr_iget - <i>mount_point</i> file system inode <i>inumber</i> marked bad on disk</p> <p>WARNING: msgcnt x: vxfs: mesg 017: vx_attr_indadd - <i>mount_point</i> file system inode <i>inumber</i> marked bad on disk</p> <p>WARNING: msgcnt x: vxfs: mesg 017: vx_attr_indtrunc - <i>mount_point</i> file system inode <i>inumber</i> marked bad on disk</p> <p>WARNING: msgcnt x: vxfs: mesg 017: vx_attr_iremove - <i>mount_point</i> file system inode <i>inumber</i> marked bad on disk</p> <p>WARNING: msgcnt x: vxfs: mesg 017: vx_bmap - <i>mount_point</i> file system inode <i>inumber</i> marked bad on disk</p> <p>WARNING: msgcnt x: vxfs: mesg 017: vx_bmap_indirect_ext4 - <i>mount_point</i> file system inode <i>inumber</i> marked bad on disk</p> <p>WARNING: msgcnt x: vxfs: mesg 017: vx_delbuf_flush - <i>mount_point</i> file system inode <i>inumber</i> marked bad on disk</p> <p>WARNING: msgcnt x: vxfs: mesg 017: vx_dio_iovec - <i>mount_point</i> file system inode <i>inumber</i> marked bad on disk</p> <p>WARNING: msgcnt x: vxfs: mesg 017: vx_dirbread - <i>mount_point</i> file system inode <i>inumber</i> marked bad on disk</p> <p>WARNING: msgcnt x: vxfs: mesg 017: vx_dircreate - <i>mount_point</i> file system inode <i>inumber</i> marked bad on disk</p> <p>WARNING: msgcnt x: vxfs: mesg 017: vx_dirlook - <i>mount_point</i> file system inode <i>inumber</i> marked bad on disk</p> <p>WARNING: msgcnt x: vxfs: mesg 017: vx_doextop_iau - <i>mount_point</i> file system inode <i>inumber</i> marked bad on disk</p> <p>WARNING: msgcnt x: vxfs: mesg 017: vx_doextop_now - <i>mount_point</i> file system inode <i>inumber</i> marked bad on disk</p> <p>WARNING: msgcnt x: vxfs: mesg 017: vx_do_getpage - <i>mount_point</i> file system inode <i>inumber</i> marked bad on disk</p> <p>WARNING: msgcnt x: vxfs: mesg 017: vx_enter_ext4 - <i>mount_point</i> file system inode <i>inumber</i> marked bad on disk</p> <p>WARNING: msgcnt x: vxfs: mesg 017: vx_exttrunc - <i>mount_point</i> file system inode <i>inumber</i> marked bad on disk</p> <p>WARNING: msgcnt x: vxfs: mesg 017: vx_get_alloc - <i>mount_point</i> file system inode <i>inumber</i> marked bad on disk</p>

Message Number	Message and Definition
079 (continued)	<p>WARNING: msgcnt x: vxfs: msg 017: vx_ilisterr - <i>mount_point</i> file system inode <i>inumber</i> marked bad on disk</p> <p>WARNING: msgcnt x: vxfs: msg 017: vx_indtrunc - <i>mount_point</i> file system inode <i>inumber</i> marked bad on disk</p> <p>WARNING: msgcnt x: vxfs: msg 017: vx_iread - <i>mount_point</i> file system inode <i>inumber</i> marked bad on disk</p> <p>WARNING: msgcnt x: vxfs: msg 017: vx_iremove - <i>mount_point</i> file system inode <i>inumber</i> marked bad on disk</p> <p>WARNING: msgcnt x: vxfs: msg 017: vx_iremove_attr - <i>mount_point</i> file system inode <i>inumber</i> marked bad on disk</p> <p>WARNING: msgcnt x: vxfs: msg 017: vx_logwrite_flush - <i>mount_point</i> file system inode <i>inumber</i> marked bad on disk</p> <p>WARNING: msgcnt x: vxfs: msg 017: vx_oltmount_iget - <i>mount_point</i> file system inode <i>inumber</i> marked bad on disk</p> <p>WARNING: msgcnt x: vxfs: msg 017: vx_overlay_bmap - <i>mount_point</i> file system inode <i>inumber</i> marked bad on disk</p> <p>WARNING: msgcnt x: vxfs: msg 017: vx_readnomap - <i>mount_point</i> file system inode <i>inumber</i> marked bad on disk</p> <p>WARNING: msgcnt x: vxfs: msg 017: vx_reorg_trunc - <i>mount_point</i> file system inode <i>inumber</i> marked bad on disk</p> <p>WARNING: msgcnt x: vxfs: msg 017: vx_stablestore - <i>mount_point</i> file system inode <i>inumber</i> marked bad on disk</p> <p>WARNING: msgcnt x: vxfs: msg 017: vx_tranitimes - <i>mount_point</i> file system inode <i>inumber</i> marked bad on disk</p> <p>WARNING: msgcnt x: vxfs: msg 017: vx_trunc - <i>mount_point</i> file system inode <i>inumber</i> marked bad on disk</p> <p>WARNING: msgcnt x: vxfs: msg 017: vx_write_alloc2 - <i>mount_point</i> file system inode <i>inumber</i> marked bad on disk</p> <p>WARNING: msgcnt x: vxfs: msg 017: vx_write_default - <i>mount_point</i> file system inode <i>inumber</i> marked bad on disk</p> <p>WARNING: msgcnt x: vxfs: msg 017: vx_zero_alloc - <i>mount_point</i> file system inode <i>inumber</i> marked bad on disk</p>



Message Number	Message and Definition
079 (continued)	<p>◆ Description</p> <p>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.</p> <p>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 <code>fsdb</code> to change the file system.</p> <p>The <code>VX_FULLEFSCK</code> flag is set in the super-block so <code>fsck</code> will do a full structural check the next time it is run.</p> <p>◆ Action</p> <p>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 <code>fsck</code> to run a full structural check.</p>

Message Number	Message and Definition
080	<p>WARNING: msgcnt x: vxfs: mesg 080: 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 vxupgrade(1M) for information on upgrading a VxFS file system and see the VxFS Release Notes for information on disk layout support.</p> <p>◆ Action</p> <p>Use the vxupgrade command to begin upgrading file systems using older disk layouts to Version 5. Consider the following when planning disk layout upgrades:</p> <ul style="list-style-type: none"> ◆ 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. Offline conversions tools will be provided in the next VxFS feature release to aid in migrating volume-image backup copies of Version 1 and Version 2 disk layout file systems to a Version 4 disk layout.
081	<p>WARNING: msgcnt x: vxfs: mesg 081: possible network partition detected</p> <p>◆ Description</p> <p>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. Thus the cluster is in one of two possible states. Either the last network link is broken (called a <i>network partition</i> condition), or the last network link is okay, but the node crashed, in which case it is not a network partition problem. It is not possible to identify whether it is the first or second state, so this warning is issued to indicate that a network partition may exist and there is a possibility of data corruption.</p> <p>◆ Action</p> <p>A network link failure is typically detected by the underlying communication software, at which time you must rectify the network problem. If a problem goes unnoticed, however, and then the last network link also fails, CFS indicates all file systems were mounted on the shared disk devices.</p>



Message Number	Message and Definition
082	<p>WARNING: msgcnt x: vxfs: mesg 082: <i>mount_point</i> file system is on a shared volume and may become corrupted if the cluster is in a partitioned state</p> <p>◆ Description</p> <p>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.</p> <p>◆ Action</p> <p>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.</p>
083	<p>WARNING: msgcnt x: vxfs: mesg 083: <i>mount_point</i> file system log is not compatible with the specified intent log I/O size</p> <p>◆ Description</p> <p>Either the specified <code>mount logiosize</code> size is not compatible with the file system layout, or the file system is corrupted.</p> <p>◆ Action</p> <p>Mount the file system again without specifying the <code>logiosize</code> option, or use a <code>logiosize</code> value compatible with the intent log specified when the file system was created. If the error persists, unmount the file system and use <code>fsck</code> to run a full structural check.</p>

Introduction

The disk layout is the way file system information is stored on disk. On VxFS, five different disk layout versions were created to take advantage of evolving technological developments. The disk layout versions used on VxFS were:

Version 1	The Version 1 disk layout is the original VxFS disk layout provided with pre-2.0 versions of VxFS.
Version 2	The Version 2 disk layout was designed to support features such as filesets, dynamic inode allocation, and enhanced security. The Version 2 layout is available with and without quotas support.
Version 3	The 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.
Version 4	The Version 4 disk layout encompasses all file system structural information in files, rather than at fixed locations on disk, allowing for greater scalability. Version 4 supports files and file systems up to one terabyte in size.
Version 5	Version 5 enables the creation of file system sizes up to 32 terabytes. Files can be a maximum of two terabytes. File systems larger than 1TB must be created on a VERITAS Volume Manager volume.

Some of the disk layout versions were not supported on all UNIX operating systems. Currently, only the Version 4 and 5 disk layouts can be created. Version 1 and 2 file systems can still be mounted, but this will be disallowed in future releases.

The `vxupgrade` command is provided to upgrade an existing VxFS file system to the Version 5 layout while the file system remains online. See the `vxupgrade(1M)` manual page for details on upgrading VxFS file systems.

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. Version 4 disk layouts cannot be upgraded offline. See the `vxfsconver(1M)` manual page for details on upgrading VxFS disk layouts.



The following additional topics are covered in this appendix:

- ◆ [Disk Space Allocation](#)
- ◆ [The VxFS Version 4 Disk Layout](#)
- ◆ [The VxFS Version 5 Disk Layout](#)
- ◆ [Using UNIX Commands on File Systems Larger than One TB](#)

Disk Space Allocation

Disk space is allocated by the system in 512-byte sectors. An integral number of sectors are grouped together to form a logical block. VxFS supports logical block sizes of 1024, 2048, 4096, and 8192 bytes. The default block size is 1024 bytes. The block size may be specified as an argument to the `mkfs` utility and may vary between VxFS file systems mounted on the same system. VxFS allocates disk space to files in extents. An extent is a set of contiguous blocks.

The VxFS Version 4 Disk Layout

Note The information in this section also applies to the Version 5 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 <i>structural fileset</i> defines the file system structure, the <i>primary fileset</i> 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	There is a <code>quotas</code> file which is used to track the resources allocated to each user and a <code>quotas.group</code> file to track the resources allocated to each group.

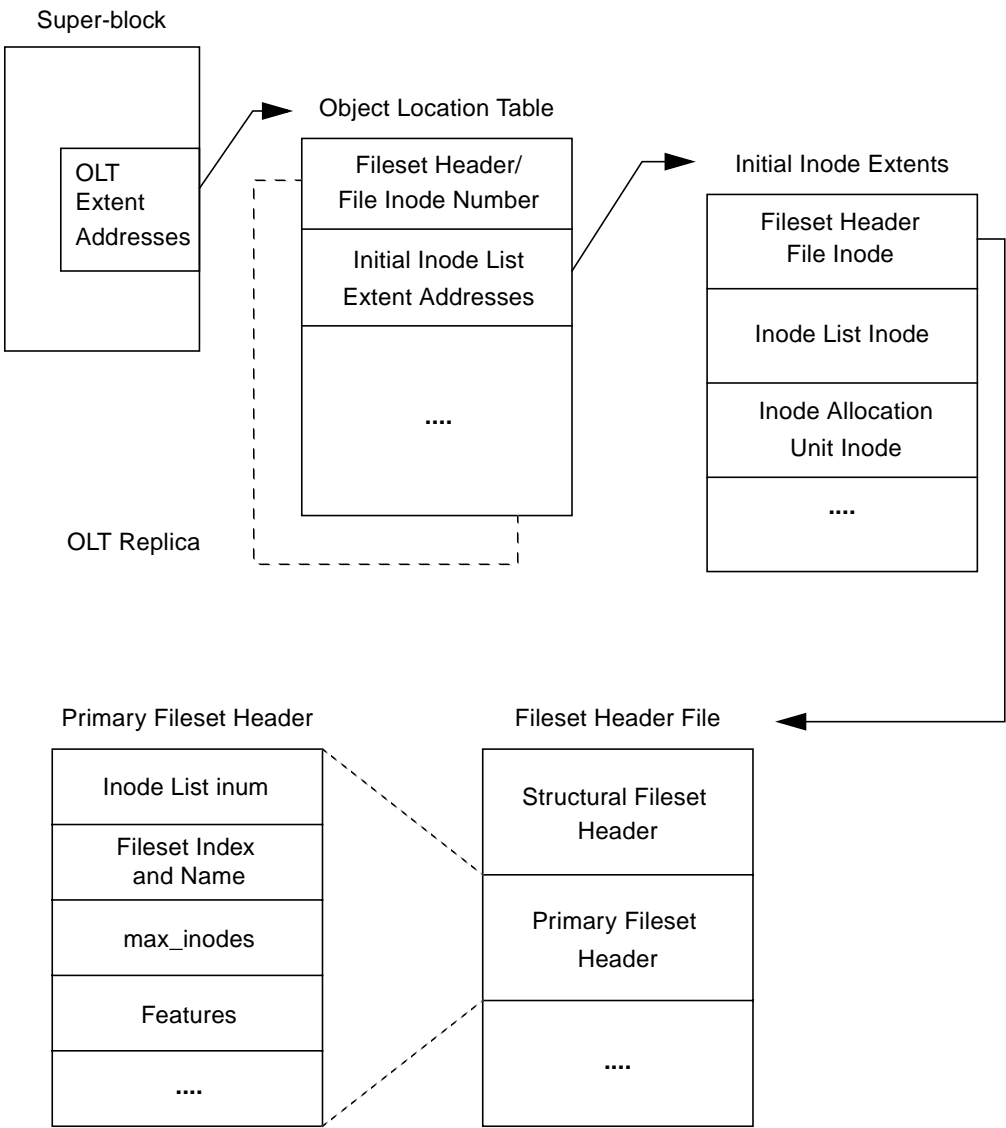


The following figure shows how the kernel and utilities build information about the structure of the file system. The super-block location is in a known location from which the OLT can be located. From the OLT, the initial extents of the structural inode list can be located along with the inode number 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.

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. This file system technology is implemented in other VERITAS products. For information on how to use this feature, contact your sales channel.

VxFS Version 4 Disk Layout



The VxFS Version 5 Disk Layout

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	8,589,934,078 sectors (≈ 4 TB)
2048 bytes	17,179,868,156 sectors (≈ 8 TB)
4096 bytes	34,359,736,312 sectors (≈ 16 TB)
8192 bytes	68,719,472,624 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 for more information).

The Version 5 disk layout also supports group quotas (see “[Quota Files on VxFS](#)” on page 78). Quota limits cannot exceed one terabyte.

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 `E_OVERFLOW` 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 on large file *systems* (larger than one terabyte), must be large file aware to operate correctly, even if invoked on a small file. 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 for information on the large file compilation environment.

Glossary

access control list (ACL)

The information that identifies specific users or groups and their access privileges for a particular file or directory.

agent

A process that manages predefined VERITAS Cluster Server (VCS) resource types. Agents bring resources online, take resources offline, and monitor resources to report any state changes to VCS. When an agent is started, it obtains configuration information from VCS and periodically monitors the resources and updates VCS with the resource status.

allocation unit

A group of consecutive blocks on a file system that contain resource summaries, free resource maps, and data blocks. Allocation units also contain copies of the super-block.

asynchronous writes

A delayed write in which the data is written to a page in the system's page cache, but is not written to disk before the write returns to the caller. This improves performance, but carries the risk of data loss if the system crashes before the data is flushed to disk.

atomic operation

An operation that either succeeds completely or fails and leaves everything as it was before the operation was started. If the operation succeeds, all aspects of the operation take effect at once and the intermediate states of change are invisible. If any aspect of the operation fails, then the operation aborts without leaving partial changes.

Block-Level Incremental Backup (BLI Backup)

A VERITAS backup capability that does not store and retrieve entire files. Instead, only the data blocks that have changed since the previous backup are backed up.



buffered I/O

During a read or write operation, data usually goes through an intermediate kernel buffer before being copied between the user buffer and disk. If the same data is repeatedly read or written, this kernel buffer acts as a cache, which can improve performance. See *unbuffered I/O* and *direct I/O*.

CFS

VERITAS Cluster File System.

cluster mounted file system

A shared file system that enables multiple hosts to mount and perform file operations on the same file. A cluster mount requires a shared storage device that can be accessed by other cluster mounts of the same file system. Writes to the shared device can be done concurrently from any host on which the cluster file system is mounted. To be a cluster mount, a file system must be mounted using the `mount -o cluster` option. See *local mounted file system*.

contiguous file

A file in which data blocks are physically adjacent on the underlying media.

CVM

The cluster functionality of VERITAS Volume Manager.

data block

A block that contains the actual data belonging to files and directories.

data synchronous writes

A form of synchronous I/O that writes the file data to disk before the write returns, but only marks the inode for later update. If the file size changes, the inode will be written before the write returns. In this mode, the file data is guaranteed to be on the disk before the write returns, but the inode modification times may be lost if the system crashes.

defragmentation

The process of reorganizing data on disk by making file data blocks physically adjacent to reduce access times.

direct extent

An extent that is referenced directly by an inode.

direct I/O

An unbuffered form of I/O that bypasses the kernel's buffering of data. With direct I/O, the file system transfers data directly between the disk and the user-supplied buffer. See *buffered I/O* and *unbuffered I/O*.

discovered direct I/O

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

encapsulation

A process that converts existing partitions on a specified disk to volumes. If any partitions contain file systems, `/etc/vfstab` entries are modified so that the file systems are mounted on volumes instead. Encapsulation is not applicable on some systems.

extent

A group of contiguous file system data blocks treated as a single unit. An extent is defined by the address of the starting block and a length.

extent attribute

A policy that determines how a file allocates extents.

external quotas file

A quotas file (named `quotas`) must exist in the root directory of a file system for quota-related commands to work. See *quotas file* and *internal quotas file*.

file system block

The fundamental minimum size of allocation in a file system. This is equivalent to the 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.

GB

Gigabyte (2^{30} bytes or 1024 megabytes).



hard limit

The hard limit is an absolute limit on system resources for individual users for file and data block usage on a file system. See *quota*.

indirect address extent

An extent that contains references to other extents, as opposed to file data itself. A *single* indirect address extent references indirect data extents. A *double* indirect address extent references single indirect address extents.

indirect data extent

An extent that contains file data and is referenced via an indirect address extent.

inode

A unique identifier for each file within a file system that contains the data and metadata associated with that file.

inode allocation unit

A group of consecutive blocks containing inode allocation information for a given fileset. This information is in the form of a resource summary and a free inode map.

intent logging

A method of recording pending changes to the file system structure. These changes are recorded in a circular *intent log* file.

internal quotas file

VxFS maintains an internal quotas file for its internal usage. The internal quotas file maintains counts of blocks and indices used by each user. See *quotas* and *external quotas file*.

K

Kilobyte (2^{10} bytes or 1024 bytes).

large file

A file larger than two gigabytes. VxFS supports files up to one terabyte in size.

large file system

A file system more than two gigabytes in size. VxFS supports file systems up to 32 terabytes 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.

local mounted file system

A file system mounted on a single host. The single host mediates all file system writes to storage from other clients. To be a local mount, a file system cannot be mounted using the `mount -o cluster` option. See *cluster mounted file system*.

metadata

Structural data describing the attributes of files on a disk.

MB

Megabyte (2^{20} bytes or 1024 kilobytes).

mirror

A duplicate copy of a volume and the data therein (in the form of an ordered collection of subdisks). Each mirror is one copy of the volume with which the mirror is associated.

node

One of the hosts in a cluster.

node abort

A situation where a node leaves a cluster (on an emergency basis) without attempting to stop ongoing operations.

node join

The process through which a node joins a cluster and gains access to shared disks.

object location table (OLT)

The information needed to locate important file system structural elements. The OLT is written to a fixed location on the underlying media (or disk).

object location table replica

A copy of the OLT in case of data corruption. The OLT replica is written to a fixed location on the underlying media (or disk).



page file

A fixed-size block of virtual address space that can be mapped onto any of the physical addresses available on a system.

preallocation

A method of allowing an application to guarantee that a specified amount of space is available for a file, even if the file system is otherwise out of space.

primary fileset

The files that are visible and accessible to the user.

Quick I/O file

A regular VxFS file that is accessed using the `::cdev:vxfs:` extension.

Quick I/O for Databases

Quick I/O is a VERITAS File System feature that improves database performance by minimizing read/write locking and eliminating double buffering of data. This allows online transactions to be processed at speeds equivalent to that of using raw disk devices, while keeping the administrative benefits of file systems.

QuickLog

VERITAS QuickLog is a high performance mechanism for receiving and storing intent log information for VxFS file systems. QuickLog increases performance by exporting intent log information to a separate physical volume.

quotas

Quota limits on system resources for individual users for file and data block usage on a file system. See *hard limit* and *soft limit*.

quotas file

The quotas commands read and write the external quotas file to get or change usage limits. When quotas are turned on, the quota limits are copied from the external quotas file to the internal quotas file. See *quotas*, *internal quotas file*, and *external quotas file*.

reservation

An extent attribute used to preallocate space for a file.

root disk group

A special private disk group that always exists on the system. The root disk group is named `rootdg`.



shared disk group

A disk group in which the disks are shared by multiple hosts (also referred to as a cluster-shareable disk group).

shared volume

A volume that belongs to a shared disk group and is open on more than one node at the same time.

snapshot file system

An exact copy of a mounted file system at a specific point in time. Used to do online backups.

snapped file system

A file system whose exact image has been used to create a snapshot file system.

soft limit

The soft limit is lower than a hard limit. The soft limit can be exceeded for a limited time. There are separate time limits for files and blocks. See *hard limit* and *quota*.

storage checkpoint

A facility that provides a consistent and stable view of a file system or database image and keeps track of modified data blocks since the last checkpoint.

structural fileset

The files that define the structure of the file system. These files are not visible or accessible to the user.

super-block

A block containing critical information about the file system such as the file system type, layout, and size. The VxFS super-block is always located 8192 bytes from the beginning of the file system and is 8192 bytes long.

synchronous writes

A form of synchronous I/O that writes the file data to disk, updates the inode times, and writes the updated inode to disk. When the write returns to the caller, both the data and the inode have been written to disk.

TB

Terabyte (2^{40} bytes or 1024 gigabytes).



transaction

Updates to the file system structure that are grouped together to ensure they are all completed

throughput

For file systems, this typically refers to the number of I/O operations in a given unit of time.

ufs

The UNIX file system type. Used as parameter in some commands.

UFS

The UNIX file system; derived from the 4.2 Berkeley Fast File System.

Unbuffered I/O

I/O that bypasses the kernel cache to increase I/O performance. This is similar to direct I/O, except when a file is extended; for direct I/O, the inode is written to disk synchronously, for unbuffered I/O, the inode update is delayed. See *buffered I/O* and *direct I/O*.

VCS

VERITAS Cluster Server.

volume

A virtual disk which represents an addressable range of disk blocks used by applications such as file systems or databases.

vxfs

The VERITAS File System type. Used as a parameter in some commands.

VxFS

The VERITAS File System.

VxVM

The VERITAS Volume Manager.

Index

A

- access control lists 14
- alias for Quick I/O files 85
- allocation policies 39
 - default 39
 - extent 5
 - extent based 5
- application
 - expanded facilities 9
 - transparency 9

B

- bad block revectoring 21
- blkclear 10
- blkclear mount option 19, 21
- block based architecture 3
- block size 5, 158
- blockmap for a snapshot file system 75
- buffered file systems 10
- buffered I/O 44

C

- cache advisories 44–46
- Cached Quick I/O 91
- Cached Quick I/O read-ahead 91
- CFS QuickLog 102
- closesync 10
- cluster mount 16
- configuration file, /etc/qlog/config 102
- contiguous reservation 40
- converting a data Storage Checkpoint to a nodata Storage Checkpoint 61
- convosync mount option 19, 23
- copy-on-write technique 52, 55
- cp_vxfs 41
- cpio_vxfs 41
- creating file systems with large files 24
- creating files with mkfs 104
- creating Quick I/O files 86

- cron 8, 29
- cron sample script 30

D

- data copy 44
- data integrity 10
- data Storage Checkpoints definition 56
- data synchronous I/O 22, 45
- data transfer 44
- default
 - allocation policy 39
 - block sizes 5, 158
- default_indir_size tunable parameter 32
- defragmentation 8
 - extent 29
 - scheduling with cron 29
- delaylog mount option 19, 20
- device file 159
- direct data transfer 44
- direct I/O 44
- directory reorganization 30
- disabled file system
 - snapshot 76
 - transactions 122
- discovered direct I/O 45
- discovered_direct_iosize tunable parameter 33
- disk layout
 - Version 4 158, 162
 - Version 5 157
- disk space allocation 5, 158
- displaying mounted file systems 111

E

- enabling Quick I/O 91
- enhanced data integrity modes 10
- ENOENT 126
- ENOSPC 68
- ENOTDIR 126



- expansion 8
- expansion of a file system 28
- extensions of Quick I/O files 85
- extent 5, 37
 - attributes 37
 - description 158
 - indirect 5
 - information 46
 - reorganization 30
- extent allocation 5
 - aligned 38
 - control 37
 - fixed size 38
 - unit state file 159
 - unit summary file 159
- extent size
 - fixed 49
 - indirect 5
- external quotas file 78

F

- file
 - device 159
 - extent allocation unit state 159
 - extent allocation unit summary 159
 - fileset header 159
 - free extent map 159
 - inode allocation unit 159
 - inode list 159
 - label 159
 - log 159
 - object location table 159
 - quotas 159
 - sparse 39, 49
- file system
 - block size 42
 - buffering 10
 - displaying mounted 111
 - increasing size 113
- fileset
 - header file 159
 - primary 53
- fixed extent size 38, 49
- fixed write size 39
- fragmentation
 - monitoring 29, 30
 - reorganization facilities 29
 - reporting 29
- fragmented file system characteristics 29

- free extent map file 159
- free space monitoring 28
- free space, monitoring 28
- freeze 50
- freezing and thawing, relation to Storage Checkpoints 53
- fsadm 8
 - how to reorganize a file system 115
 - how to resize a file system 113
 - reporting extent fragmentation 29
 - scheduling defragmentation using cron 29
- fsadm_vxfs 25
- fscat 71
- fsck 60
- fsckptadm, Storage Checkpoint administration 57
- fstyp, how to determine the file system type 112

G

- get I/O parameter ioctl 50
- gettext 41
- getfacl 14
- global message IDs 123

H

- how to access a Storage Checkpoint 59
- how to create a backup file system 116
- how to create a Storage Checkpoint 57
- how to determine the file system type 112
- how to display mounted file systems 110
- how to edit the vfstab file 108
- how to mount a Storage Checkpoint 59
- how to remove a Storage Checkpoint 58
- how to reorganize a file system 115
- how to resize a file system 113
- how to restore a file system 117
- how to set up user quotas 119
- how to turn on quotas 118
- how to unmount a Storage Checkpoint 60
- how to view quotas 120
- HSM agent error message 148
- hsm_write_prealloc 33

I

- I/O
 - direct 44
 - sequential 44
 - synchronous 44



- I/O requests
 - asynchronous 22
 - synchronous 21
 - increasing file system size 113
 - indirect extent
 - address size 5
 - double 5
 - single 5
 - initial_extent_size tunable parameter 33
 - inode allocation unit file 159
 - inode list error 122
 - inode list file 159
 - inode table 27
 - internal 27
 - sizes 27
 - inodes, block based 5
 - internal inode table 27
 - internal quotas file 78
 - ioctl interface 37
- K**
- kernel asynchronous I/O 84
 - kernel tunables 27
- L**
- label file 159
 - large files 12, 24
 - creating file systems with 24
 - mounting file systems with 25
 - largefiles mount option 25
 - load balancing 100
 - local mount 16
 - log failure 122
 - log file 159
 - log files 48
 - log mount option 19
 - logiosize mount option 20
- M**
- max_direct_iosize tunable parameter 34
 - max_diskq tunable parameter 34
 - max_seqio_extent_size tunable parameter 34
 - maximum I/O size 28
 - mincache mount option 19, 21
 - mkfs 158
 - creating files with 104
 - creating large files 25
 - modes, enhanced data integrity 10
 - monitoring fragmentation 29
 - mount 10, 25
 - how to display mounted file systems 110
 - how to mount a file system 106
 - mounting a Storage Checkpoint 59
 - pseudo device 59
 - mount options 19–24
 - blkclear 19, 21
 - choosing 19–24
 - combining 26
 - convosync 19, 23
 - delaylog 11, 19, 20
 - extended 10
 - largefiles 25
 - log 11, 19
 - logiosize 20
 - mincache 19, 21
 - nodatainlog 19, 21
 - qlog 24
 - tmplog 19, 20
 - mounted file system, displaying 111
 - mounting a file system 106
 - option combinations 26
 - with large files 25
 - with QuickLog 99
 - mounting a Storage Checkpoint 60
 - mounting a Storage Checkpoint of a cluster file system 60
 - msgcnt field 123
 - multiple block operations 5
 - mv_vxfs 41
- N**
- name space, preserved by Storage Checkpoints 52
 - naming convention, Quick I/O 85
 - NFS 9
 - nodata Storage Checkpoints 61
 - nodata Storage Checkpoints definition 56
 - nodatainlog mount option 19, 21
 - non-mountable Storage Checkpoints definition 57
- O**
- O_SYNC 19
 - object location table file 159
- P**
- parameters
 - default 31
 - tunable 32



- tuning 30
- performance
 - enhancing 43
 - overall 18
 - snapshot file systems 72
- preallocating space for Quick I/O files 89
- primary fileset relation to Storage Checkpoints 53
- pseudo device 59

Q

- qio module, loading on system reboot 94
- qio_cache_enable tunable parameter 34, 91
- qiomkfile 86
- qiostat 94
- qlog mount option 24
- qlogattach 102
- qlogck 102
- qlogdetach 100
- qlogenable 99
- qlogmk 98
- qlogrm 99
- qlogstat 101
- Quick I/O 83
 - access Quick I/O files as raw devices 85
 - access regular UNIX files 88
 - creating Quick I/O files 86
 - direct I/O 84
 - double buffering 85
 - extension 85
 - read/write locks 84
 - restrictions 86
 - special naming convention 85
- Quick I/O files
 - access regular UNIX files 88
 - preallocating space 89
 - statistics 94
 - using relative and absolute path names 88
- QuickLog
 - disabling 100
 - enabling 99
 - load balancing 100
 - logical view 96
 - number of supported devices 96
 - on CFS 102
 - overview 96
 - removing 99
 - troubleshooting 100

- quota commands 79
- quotacheck 80
- quotas 77
 - exceeding the soft limit 78
 - hard limit 78
 - how to view quotas 120
 - soft limit 78
- quotas file 78, 159
- quotas.grp file 78

R

- read_nstream tunable parameter 32
- read_pref_io tunable parameter 32
- read-ahead functionality in Cached Quick I/O 91
- read-only Storage Checkpoints 59
- recovery, QuickLog 102
- relative and absolute path names used with symbolic links 88
- removable Storage Checkpoints definition 56
- reorganization
 - directory 30
 - extent 30
- report extent fragmentation 29
- reservation space 38, 46, 48
- restrictions on Quick I/O 86

S

- sectors, forming logical blocks 158
- sequential I/O 44
- setext 41
- setfacl 14
- snapof 71
- snapped file systems 14, 70
 - performance 72
 - unmounting 70
- snapread 71
- snapshot 116
- snapshot file system
 - on CFS 70
- snapshot file systems 14, 70
 - blockmap 75
 - creating 71
 - data block area 75
 - disabled 76
 - errors 135
 - for backup 70
 - fscat 71
 - fsck 71

- fuser 70
- mounting 71
- multiple 70
- performance 72
- read 71
- super-block 74
- snapshot, how to create a backup file system 116
- snapsize 71
- space reservation 46–48
- sparse file 39, 49
- statistics
 - generated for Quick I/O 94
 - QuickLog 101
- storage
 - clearing 21
 - uninitialized 21
- Storage Checkpoints
 - accessing 59
 - administration of 57
 - converting a data Storage Checkpoint to a nodata Storage Checkpoint with multiple Storage Checkpoints 64
 - creating 57
 - data Storage Checkpoints 56
 - definition of 52
 - difference between a data Storage Checkpoint and a nodata Storage Checkpoint 62
 - freezing and thawing a file system 53
 - mounting 59
 - nodata Storage Checkpoints 56, 61
 - non-mountable Storage Checkpoints 57
 - operation failures 68
 - pseudo device 59
 - read-only Storage Checkpoints 59
 - removable Storage Checkpoints 56
 - removing 58
 - space management 68
 - synchronous vs. asynchronous conversion 61
 - types of 56
 - unmounting 60
 - using the fsck utility 60
 - writable Storage Checkpoints 59
- super-block 74
- SVID requirement, VxFS conformance to 9
- symbolic links, accessing Quick I/O files 88
- synchronous I/O 44

- system failure recovery 7
- system performance 17
 - enhancing 43
 - overall 18

T

- temporary directories 12
- thaw 50
- tmplog mount option 19, 20
- transaction disabling 122
- tunable I/O parameters 32
 - default_indir_size 32
 - discovered_direct_iosize 33
 - initial_extent_size 33
 - max_direct_iosize 34
 - max_diskq 34
 - max_seqio_extent_size 34
 - qio_cache_enable 34, 91
 - read_nstream 32
 - read_pref_io 32
 - Volume Manager maximum I/O size 28
 - write_nstream 32
 - write_pref_io 32
 - write_throttle 35
- tuning I/O parameters 30
- typed extents 6

U

- umount command 110
- uninitialized storage, clearing 21
- unmount 60, 123
 - a snapped file system 70
- utilities
 - cron 8
 - fsadm 8
 - fscat 14
 - getext 41
 - mkfs 158
 - qiostat 94
 - setext 41
 - vxassist 98
 - vxedit 99

V

- VEA 4
- VERITAS Enterprise Administrator 4
- Version 4 disk layout 158, 162
- Version 5 disk layout 157
- vfstab file, editing 108
- virtual disks 8



vol_maxio tunable I/O parameter 28
VOP_INACTIVE 138
VX_CHGFSIZE 47
VX_CONTIGUOUS 47
VX_DSYNC 45
VX_FREEZE 50, 80
VX_FULLFCK 122, 125, 126, 127, 128, 131,
132, 133, 134, 137, 138, 141, 145, 146, 147, 148,
154
VX_GETCACHE 44
VX_GETTEXT 46
VX_NOEXTEND 47
VX_NORESERVE 47
VX_NOREUSE 46
VX_RANDOM 46
VX_SEQ 46
VX_SETCACHE 44
VX_SETTEXT 46
VX_SNAPREAD 71
VX_THAW 50
VX_TRIM 47
VX_UNBUFFERED 45

vxassist 98
vxdump 41
vxedit, removing a VxVM volume 99
vxedquota, how to set up user quotas 119
VxFS
 disk layout 157
 disk structure 157
 storage allocation 18
vxfs_ninode 27
vxquota, how to view quotas 120
vxquotaoff, how to turn off quotas 120
vxquotaon 118
vxrestore 41, 117
vxtunefs, changing extent size 5

W

writable Storage Checkpoints 59
write size 39
write_nstream tunable parameter 32
write_pref_io tunable parameter 32
write_throttle tunable parameter 35