VERITAS

VERITAS® File System
System Administrator's Guide
Release 3.3.2

Contents

Preface

Introduction

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

- An understanding of system administration.

- A working knowledge of the UNIX operating system.

- A general understanding of file systems.

Organization

- Chapter 1, "The VERITAS File System," introduces the features and characteristics of this product.

- Chapter 2, "Disk Layout," describes and illustrates the major components of VxFS disk layouts.

- Chapter 3, "Extent Attributes," describes the policies associated with allocation of disk space.

- Chapter 4, "Online Backup," describes the snapshot backup feature of VxFS.

- Chapter 5, "Performance and Tuning," describes VxFS tools that optimize system performance. This section includes information on mount options.

- Chapter 6, "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 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 product that improves the performance of log writes.

- Appendix A, "Kernel Messages," lists VxFS kernel error messages in numerical order and provides explanations and suggestions for dealing with these problems.

- "Glossary" contains a list of terms and definitions relevant to VxFS.

Related Documents

> **Note:** The *VERITAS File System Installation Guide* provides information on installation procedures and verification. Make sure that either the VxFS or VxFS Advanced feature set is properly installed on your system.

- The *VERITAS File System Quick Start Guide* provides information on common file system tasks and examples of typical VxFS operations.
- The online manual pages provide additional details on VxFS commands and utilities.

Conventions

The following table describes the typographic conventions in this guide:

Typeface	Usage	Examples
`courier`	Computer output, user input, command names, files, and directories	™ (referred to as VxFS® or `vxfs`)ou have mail. The `cat` command displays files. `$ ls -a`
italics	New terms, document titles, emphasis, glossary cross references, variables replaced with names or values	`$ cat` *filename* See the *User's Guide* for details.
bold	Glossary terms	
Symbol	**Usage**	
`%`	C shell prompt	
`$`	Bourne/Korn shell prompt	
`#`	Superuser prompt (all shells)	
`\`	Continued input on the next line; you do not have to type this character	`# mount -F vxfs \` `/h/filesys`

Technical Support

For license information or information about VERITAS service packages, contact VERITAS Customer Support.

U.S. and Canadian Customers: 1-800-342-0652

International Customers: +1 (650) 335-8555

Fax: (650) 335-8428

Electronic mail: support@veritas.com

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

```
www.veritas.com
```

The VERITAS File System   1

Introduction

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

The following topics are covered in this chapter:

- VxFS Features

- Disk Layout Options

- File System Performance Enhancements

- Extent Based Allocation

- Extent Attributes

- Fast File System Recovery

- Online System Administration

- Online Backup

- Application Interface

- Extended mount Options

- Enhanced I/O Performance

- Quotas

- Access Control Lists
- Support for Large Files
- Support for Databases
- VERITAS QuickLog

VxFS Features

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

- extent based allocation
- extent attributes
- fast file system recovery
- access control lists (ACLs)
- online administration
- online backup
- enhanced application interface
- enhanced mount options
- improved synchronous write performance
- support for large file systems (up to 1 terabyte)
- support for large files (up to 2 terabytes)
- enhanced I/O performance
- support for BSD-style quotas
- support for improved database performance
- support for improved network file server (NFS) performance through use of VERITAS QuickLog[TM]

VxFS supports all `ufs` file system features and facilities except for the linking, removing, or renaming of "." and ".." directory entries. Such operations may disrupt file system sanity.

Disk Layout Options

Three disk layout formats are available with VxFS:

Version 1

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

Version 2

The Version 2 disk layout supports such features as:

- filesets
- dynamic inode allocation

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

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

Version 4

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

- files up to 2 terabytes
- file systems up to 1 terabyte
- Access Control Lists

See Chapter 2, "Disk Layout," for a description of the disk layouts.

File System Performance Enhancements

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

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

VxFs provides the following performance enhancements:

- extent based allocation

- enhanced mount options

- VERITAS Quick I/O for Databases

- data synchronous I/O

- direct I/O and discovered direct I/O

- caching advisories

- enhanced directory features

- explicit file alignment, extent size, and preallocation controls

- tuneable I/O parameters

- tuneable indirect data extent size

- integration with VERITAS Volume Manager™ (VxVM®)

The rest of this chapter along with Chapter 5, "Performance and Tuning," and Chapter 6, "Application Interface," provide more details on some of these features.

Extent Based Allocation

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

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

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

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

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

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

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

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

Typed Extents

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

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

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

type   Uniquely identifies an extent descriptor record and defines the record's length and format.

offset   Represents the logical file offset in blocks for a given descriptor. Used to optimize lookups and eliminate hole descriptor entries.

starting block
                The starting file system block of the extent.

number of blocks
                The number of contiguous blocks in the extent.

Some notes about typed extents:

- 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 10 direct extents in the inode. The inode's block map is converted to the typed format when indirection is needed to offer the advantages of both formats.

Extent Attributes

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

The setext and getext commands allow the administrator to set or view extent attributes associated with a file, as well as to preallocate space for a file. Refer to Chapter 3, "Extent Attributes," Chapter 6, "Application Interface," and the setext(1) and getext(1) manual pages for discussions on how to use extent attributes.

The vxtunefs command allows the administrator to set or view the default indirect data extent size. Refer to Chapter 5, "Performance and Tuning," and the vxtunefs(1M) manual page for discussions on how to use the indirect data extent size feature.

Fast File System Recovery

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

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

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

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

Online System Administration

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

Defragmentation

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

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

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

- removing unused space from directories

- making all small files contiguous

- consolidating free blocks for file system use

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

Resizing

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

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

- Move some users to a different file system.

- Move a subdirectory of the file system to a new file system.

- Copy the entire file system to a larger file system.

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

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

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

Online Backup

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

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

Backups require one of the following methods:

- copying selected files from the snapshot file system (using `find` and `cpio`)

- backing up the entire file system (using `fscat`)

- initiating a full or incremental backup (using `vxdump`)

For detailed information about performing online backups, see Chapter 4, "Online Backup."

Application Interface

The VxFS file system conforms to the System V Interface Definition (SVID) requirements and supports user access through the Network File System (NFS). Applications that require performance features not available with other file systems can take advantage of VxFS enhancements that are introduced in this section and covered in detail in Chapter 6, "Application Interface."

Application Transparency

In most cases, any application designed to run on the `ufs` file system should run transparently on the VxFS file system.

Expanded Application Facilities

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

- preallocate space for a file

- specify a fixed extent size for a file

- bypass the system buffer cache for file I/O

- specify the expected access pattern for a file

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

Extended mount Options

The VxFS file system supports extended mount options to specify:

- enhanced data integrity modes
- enhanced performance modes
- temporary file system modes
- improved synchronous writes

See Chapter 5, "Performance and Tuning," and the mount_vxfs(1M) manual page for details on the VxFS mount options.

Enhanced Data Integrity Modes

Note: Performance tradeoffs are associated with these mount options.

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

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

Using `blkclear` **for Data Integrity**

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

Using `closesync` **for Data Integrity**

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

Enhanced Performance Mode

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

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

Using `delaylog` **for Enhanced Performance**

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

Using `vxldlog` **for Enhanced Performance**

VxFS provides the `mount -o vxldlog=` option to activate QuickLog^{TM} for a file system. QuickLog increases VxFS performance by exporting the file system log to a separate physical volume. This eliminates the disk seek time between the VxFS data and log areas on disk and increases the performance of synchronous log writes. See Chapter 9, "VERITAS QuickLog," for details.

Temporary File System Modes

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

Using `tmplog` **For Temporary File Systems**

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

Improved Synchronous Writes

VxFS provides superior performance for synchronous write applications.

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

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

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

Enhanced I/O Performance

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

Enhanced I/O Clustering

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

VxVM Integration

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

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

Application-Specific Parameters

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

• Default Indirect Extent Size

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

- Discovered Direct I/O

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

- Maximum Direct I/O Size

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

For a discussion on VxVM integration and performance benefits, refer to Chapter 5, "Performance and Tuning," Chapter 6, "Application Interface," and the vxtunefs(1M) and tunefstab(1M) manual pages.

Quotas

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

- The *hard limit* represents an absolute limit on data blocks or files. The user may never exceed the hard limit under any circumstances.

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

The system administrator is responsible for assigning hard and soft limits to users. See Chapter 7, "Quotas," for more information.

Access Control Lists

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

Support for Large Files

The changes implemented with the Version 4 disk layout have greatly expanded file system scalability. VxFS can now support files up to two terabytes in size because file system structures are no longer in fixed locations (see Chapter 2, "Disk Layout").

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

Creating a File System with Large Files

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

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

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

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

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

Mounting a File System with Large Files

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

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

Try to avoid specifying the `largefiles` or `nolargefiles` options so the `mount` command will default to match the current setting of the on-disk flag. Use `fsadm` to manage the system after a file system is mounted.

Managing a File System with Large Files

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

    # mkfs -F vxfs -m *special_device*

    # fsadm *mount_point* | *special_device*

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

    # fsadm -o [no]largefiles *mount_point*

You can also switch capabilities on an unmounted file system:

    # fsadm -o [no]largefiles *special_device*

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

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

Support for Databases

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

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

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

- Commercial database servers such as Oracle Server can issue kernel supported asynchronous I/O calls on these pseudo devices but not on regular files.

- `read()` and `write()` system calls issued by the database server can avoid the acquisition and release of `read/write` locks inside the kernel that take place on regular files.

- VxFS can avoid double buffering of data already buffered by the database server. This ability frees up resources for other purposes and results in better performance.

- Since I/O to these devices bypasses the system buffer cache, VxFS saves on the cost of copying data between user space and kernel space when data is read from or written to a regular file. This process significantly reduces CPU time per I/O transaction compared to that of buffered I/O.

See Chapter 8, "Quick I/O for Databases," for details on VxFS database support.

VERITAS QuickLog

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

▼ 1

Disk Layout

2

Introduction

The following topics are covered in this chapter:

- Disk Space Allocation
- The VxFS Version 1 Disk Layout
  - Overview
  - Super-Block
  - Intent Log
  - Allocation Unit
- The VxFS Version 2 Disk Layout
  - Overview
  - Basic Layout
  - Filesets and Structural Files
  - Locating Dynamic Structures
- The VxFS Version 4 Disk Layout

Three disk layouts are available with the VERITAS File System:

Version 1   The Version 1 disk layout is the original VxFS disk layout provided
              with pre-2.0 versions of VxFS.

Version 2   The Version 2 disk layout was designed to support features such as
              filesets, dynamic inode allocation, and enhanced security. The
              Version 2 layout is available with and without quotas support.

Version 4   The Version 4 disk layout encompasses all file system structural
              information in files, rather than at fixed locations on disk, allowing
              for greater scalability. Version 4 supports files up to two terabytes in
              size and file systems up to one terabyte in size.

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

All disk layout versions are supported by VxFS. After VxFS Release 3.3 is
installed on a system, new file systems are created with the Version 4 layout by
default. Although `mkfs` allows the user to specify other disk layouts, it is
generally preferable to use the Version 4 layout for new file systems.

The `vxupgrade` command is provided to upgrade an existing VxFS file system
to the Version 4 layout while the file system remains online. See the
`vxupgrade`(1M) manual page for details on upgrading VxFS file systems.

Disk Space Allocation

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

The VxFS Version 1 Disk Layout

This section describes the VxFS Version 1 disk layout.

Overview

The VxFS Version 1 disk layout, as shown in Figure 1, includes:

- the super-block
- the intent log
- one or more allocation units

These elements are discussed in detail in the sections that follow.

Figure 1   VxFS Version 1 Disk Layout
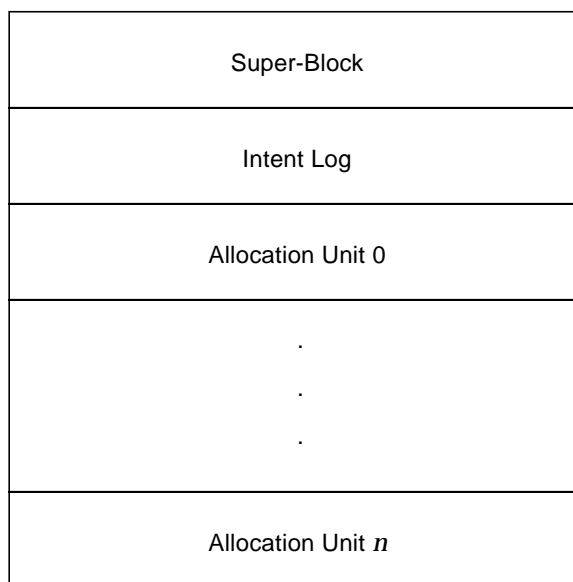
Super-Block
Intent Log
Allocation Unit 0
. . .
Allocation Unit n

Super-Block

The super-block contains important information about the file system, such as:

- the file system type
- creation and modification dates
- label information
- information about the size and layout of the file system
- the count of available resources
- the file system disk layout version number

Refer to the fs_vxfs(4) manual page for details on the contents of the super-block.

The super-block is always in a fixed location, offset from the start of the file system by 8192 bytes. This fixed location enables utilities to easily locate the super-block when necessary. The super-block is 1024 bytes long.

Copies of the super-block are kept in allocation unit headers: these copies can be used for recovery purposes if the super-block is corrupted or destroyed (see the fsck(1M) manual page for more details).

Intent Log

In the event of system failure, the VxFS file system uses intent logging to guarantee file system integrity.

The *intent log* is a circular activity log with a default size of 1024 blocks. If the file system is smaller than 4 MB, the default log size is reduced (by mkfs) to avoid wasting space. The intent log contains records of the intention of the system to update a file system structure. An update to the file system structure (a *transaction*) is divided into separate subfunctions for each data structure that needs to be updated. A composite log record of the transaction is created, containing the subfunctions constituting the transaction.

For example, the creation of a file that would expand the directory in which the file is contained would produce a transaction consisting of the following subfunctions:

- a free extent map update for the allocation of the new directory block

- a directory block update

- an inode modification for the directory size change

- an inode modification for the new file

- a free inode map update for the allocation of the new file

VxFS maintains log records in the intent log for all pending changes to the file system structure and ensures that the log records are written to disk in *advance* of the changes to the file system. Once the intent log has been written, the transaction's other updates to the file system can be written in any order. In the event of a system failure, the pending changes to the file system are either nullified or completed by the `fsck` utility. The VxFS intent log generally only records changes to the file system structure. File data changes are not normally logged.

Note: Using QuickLog does not affect the general operation of the intent log.

Allocation Unit

An allocation unit is a group of consecutive blocks in a file system that contain a resource summary, free resource maps, inodes, data blocks, and a copy of the super-block. An allocation unit in the VxFS file system is similar in concept to the ufs "cylinder group." Each component of an allocation unit begins on a block boundary. The VxFS Version 1 allocation unit is shown in Figure 2.

Figure 2   Allocation Unit Structure

Allocation Unit Header
Allocation Unit Summary
Free Inode Map
Extended Inode Operations Map
Free Extent Map
Inode List
Padding
Data Blocks
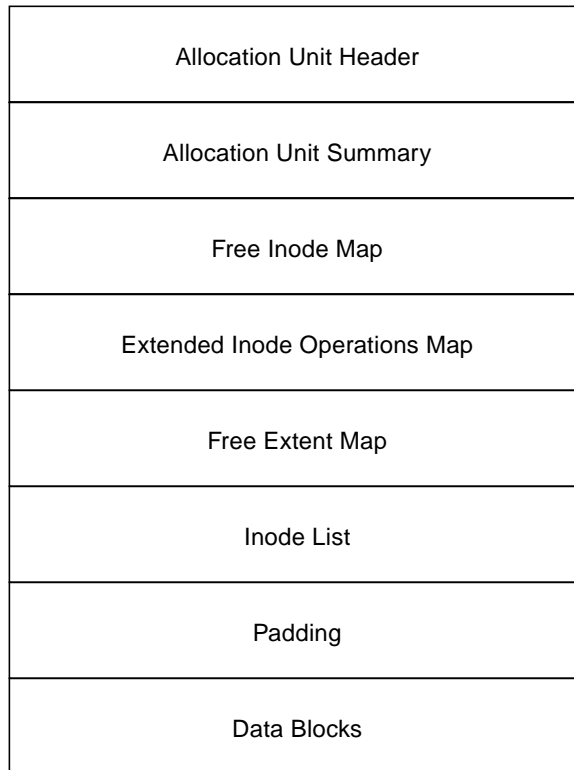
One or more allocation units exist per file system. Allocation units are located immediately after the intent log. The number and size of allocation units can be specified when the file system is made. All of the allocation units, except possibly the last one, are of equal size. If space is limited, the last allocation unit can have a partial set of data blocks to allow use of all remaining blocks.

Allocation Unit Header

The allocation unit header contains a copy of the file system's super-block that is used to verify that the allocation unit matches the super-block of the file system. The super-block copies contained in allocation unit headers can also be used for recovery purposes if the super-block is corrupted or destroyed. The allocation unit header occupies the first block of each allocation unit.

Allocation Unit Summary

The allocation unit summary contains the number of inodes with extended operations pending, the number of free inodes, and the number of free extents in the allocation unit.

Free Inode Map

The free inode map is a bitmap that indicates which inodes are free and which are allocated. A free inode is indicated by the bit being on. Inodes zero and one are reserved by the file system; inode two is the inode for the root directory; inode three is the inode for the `lost+found` directory.

Extended Inode Operations Map

The extended inode operations map keeps track of inodes on which operations would remain pending for too long to reside in the intent log. The extended inode operations map is in the same format as the free inode map. To prevent the intent log from wrapping and the transaction from getting overwritten, the required operations are stored in the affected inode (if the transaction has not completed, it does not get overwritten, the new log waits and the file system is frozen). This map is then updated to identify the inodes that have extended operations that need to be completed.

Free Extent Map

The free extent map is a series of independent 512-byte bitmaps that are each referred to as a free extent map section. Each section is broken down into multiple regions. The first region, of 2048 bits, represents a section of 2048 one-block extents. The second region, of 1024 bits, represents a section of 1024 two-block extents. This regioning continues for all powers of 2 up to the single bit that represents one 2048 block extent.

The file system uses this bitmapping scheme to find an available extent closest in size to the space required. This keeps files as contiguous as possible for faster performance.

Inode List

An inode is a data structure that contains information about a file. The VxFS inode size is 256 bytes. Each inode stores information about a particular file such as:

- file length

- link count

- owner and group IDs

- access privileges

- time of last access

- time of last modification

- pointers to the extents that contain the file's data

There are up to ten direct extent address size pairs per inode. Each direct extent address indicates the starting block number of a direct extent; direct extent sizes can vary. If all of the direct extents are used, two indirect address extents are available for use in each inode:

- The first indirect address extent is used for single indirection, where each entry in the extent indicates the starting block number of an indirect data extent.

- The second indirect address extent is used for double indirection, where each entry in the extent indicates the starting block number of a single indirect address extent.

Each indirect address extent is 8K long and contains 2048 entries. All indirect data extents for a given file have the same size, which is determined when the file's first indirect data extent is allocated.

The inode list is a series of inodes. There is one inode in the list for every file in the file system.

Padding

It may be desirable to align data blocks to a physical boundary. To facilitate this, the system administrator may specify that a gap be left between the end of the inode list and the first data block.

Data Blocks

The balance of the allocation unit is occupied by data blocks. Data blocks contain the actual data stored in files and directories.

The VxFS Version 2 Disk Layout

This section describes the VxFS Version 2 disk layout.

Due to the relatively complex nature of the Version 2 layout, the sections that follow are arranged to cover the following general areas:

- Structural elements of the file system that exist in fixed locations. These elements are discussed in the section entitled "Basic Layout."

- Structural elements of the file system that do not exist in fixed locations. These elements are discussed in the section entitled "Filesets and Structural Files."

- How the various structural elements of the file system are located and used when the file system is mounted. This is discussed in the section entitled "Locating Dynamic Structures."

Overview

Many aspects of the Version 1 disk layout are preserved in the Version 2 disk layout. However, the Version 2 layout differs from the Version 1 layout in that it includes support for the following features:

- filesets (sets of files within a file system)

- dynamic inode allocation (allocation of inodes on an as-needed basis)

- enhanced security

The addition of filesets and dynamic allocation of inodes has affected the disk layout in various ways. In particular, many of the file system structures are now located in files (referred to as *structural files*) rather than in fixed disk areas. This provides a simple mechanism for dynamic growth of structures. For example, inodes are now stored in structural files and allocated as needed. In general, file system structures that deal with space allocation are still in fixed disk locations, while most other structures are dynamically allocated and have become clients of the file system's disk space allocation scheme.

The Version 2 disk layout for VxFS 2.3 differs from previous VxFS releases because of the addition of BSD-style quota support. The differences include the fileset header structure modification to store a quota inode and preallocation of an internal quotas file.

Basic Layout

This section describes the structural elements of the file system that exist in fixed locations on the disk.

The VxFS Version 2 disk layout is illustrated in Figure 3 and is composed of:

- the super-block

- the object location table

- the intent log

- a replica of the object location table

- one or more allocation units

These and other elements are discussed in detail in the sections that follow.

Figure 3   VxFS Version 2 Disk Layout
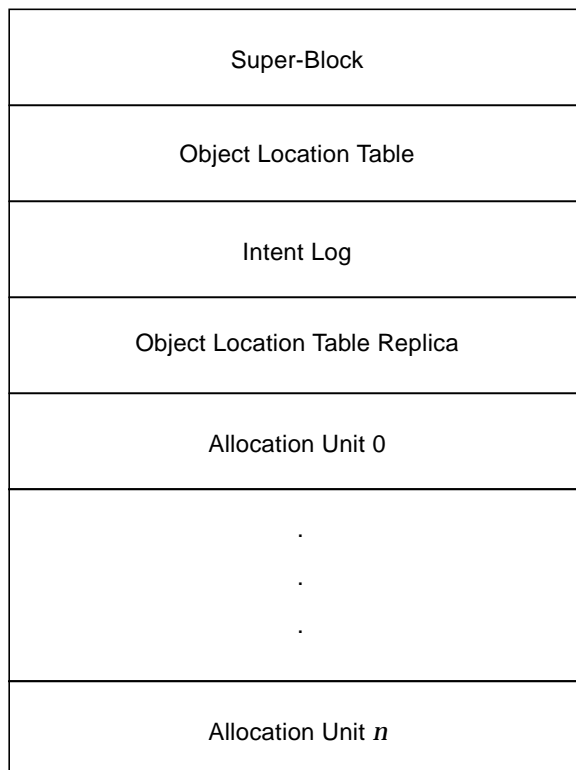
Super-Block
Object Location Table
Intent Log
Object Location Table Replica
Allocation Unit 0
. . .
Allocation Unit n

Super-Block

The super-block contains important information about the file system, such as

- the file system type
- creation and modification dates
- label information
- information about the size and layout of the file system
- the count of available resources
- the file system disk layout version number
- pointers to the object location table and its replica

The super-block is always in a fixed location, offset from the start of the file system by 8192 bytes. This fixed location enables utilities to easily locate the super-block when necessary. The super-block is 1024 bytes long.

Copies of the super-block are kept in allocation unit headers: these copies can be used for recovery purposes if the super-block is corrupted or destroyed (see the fsck(1M) manual page).

Object Location Table

The object location table (OLT) can be considered an extension of the super-block. The OLT contains information used at mount time to locate file system structures that are not in fixed locations. The OLT is typically located immediately after the super-block and is 8K long. However, if a Version 1 file system is upgraded to Version 2, the placement of the OLT depends on the availability of space.

The OLT is replicated and its replica is located immediately after the intent log. The OLT and its replica are separated in order to minimize the potential for losing both copies of the vital OLT information in the event of localized disk damage.

The contents and use of the OLT are described in detail in the section entitled "Locating Dynamic Structures."

Intent Log

The VxFS file system uses intent logging to guarantee file system integrity in the event of system failure

The *intent log* is a circular activity log with a default size of 512 blocks. If the file system is less than 4 MB, the log size will be reduced to avoid wasting space. The intent log contains records of the intention of the system to update a file system structure. An update to the file system structure (a *transaction*) is divided into separate subfunctions for each data structure that needs to be updated. A composite log record of the transaction is created that contains the subfunctions that constitute the transaction.

For example, the creation of a file that would expand the directory in which the file is contained will produce a transaction consisting of the following subfunctions:

- a free extent map update for the allocation of the new directory block

- a directory block update

- an inode modification for the directory size change

- an inode modification for the new file

- a free inode map update for the allocation of the new file

VxFS maintains log records in the intent log for all pending changes to the file system structure, and ensures that the log records are written to disk in *advance* of the changes to the file system. Once the intent log has been written, the transaction's other updates to the file system can be written in any order. In the event of a system failure, the pending changes to the file system are either nullified or completed by the fsck utility. The VxFS intent log generally only records changes to the file system structure. File data changes are not normally logged.

Note: Using QuickLog does not affect the general operation of the intent log.

Allocation Unit

An allocation unit is a group of consecutive blocks in a file system that contain
a resource summary, a free resource map, data blocks, and a copy of the
super-block. An allocation unit in the VxFS file system is similar in concept to
the ufs "cylinder group." Each component of an allocation unit begins on a
block boundary. All of the Version 2 allocation unit components deal with the
allocation of disk space. Those components of the Version 1 allocation unit that
deal with inode allocation have been relocated elsewhere for Version 2. In
particular, the inode list now resides in an inode list file and the inode
allocation information now resides in an inode allocation unit (described later).
The VxFS Version 2 allocation unit is depicted in Figure 4.

Figure 4   Allocation Unit Structure

Allocation Unit Header
Allocation Unit Summary
Free Extent Map
Padding
Data Blocks
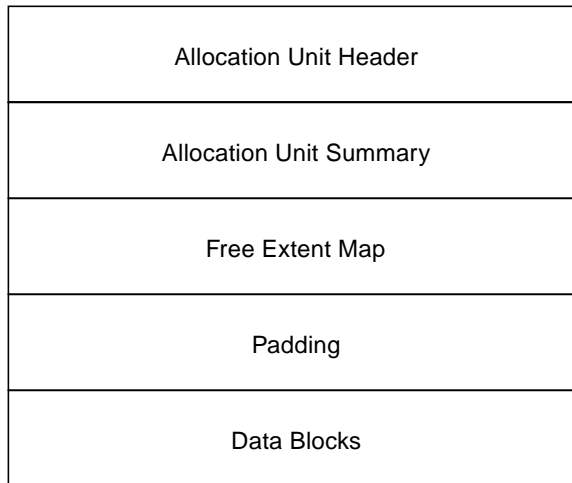
One or more allocation units exist per file system. Allocation units are located
after the OLT replica. The number and size of allocation units can be specified
when the file system is made. All of the allocation units, except possibly the
last one, are of equal size. If space is limited, the last allocation unit can have a
partial set of data blocks to allow use of all remaining blocks.

Allocation Unit Header

The allocation unit header contains a copy of the file system's super-block that is used to verify that the allocation unit matches the super-block of the file system. The super-block copies contained in allocation unit headers can also be used for recovery purposes if the super-block is corrupted or destroyed. The allocation unit header occupies the first block of each allocation unit.

Allocation Unit Summary

The allocation unit summary summarizes the resources (data blocks) used in the allocation unit. This includes information such as the number of free extents of each size in the allocation unit.

Free Extent Map

The free extent map is a series of independent 512-byte bitmaps that are each referred to as a free extent map section. Each section is broken down into multiple regions. The first region of 2048 bits represents a section of 2048 one-block extents. The second region of 1024 bits represent a section of 1024 two-block extents. This regioning continues for all powers of 2 up to the single bit that represents one 2048 block extent.

The file system uses this bitmapping scheme to find an available extent closest in size to the space required. This keeps files as contiguous as possible for faster performance.

Padding

It may be desirable to align data blocks to a physical boundary. To facilitate this, the system administrator may specify that a gap be left between the end of the free extent map and the first data block. Refer to the "Alignment" section in Chapter 6, "Application Interface," for additional information.

Data Blocks

The balance of the allocation unit is occupied by data blocks. Data blocks contain the actual data stored in files and directories.

Filesets and Structural Files

This section describes the structural elements of the file system that are not necessarily in fixed locations on the disk.

With the Version 2 layout, many structural elements of the file system are encapsulated in files to allow dynamic allocation of the file system structure. Files that store this file system structural data are referred to as *structural files*. As the file system grows, more space is allocated to the structural files. Structural files are intended for file system use only and are not generally visible to users.

The Version 2 layout supports *filesets*, which are collections of files that exist within a file system. In the current release, each file system contains two filesets:

attribute fileset
> A special fileset that stores the structural elements of the file system in the form of structural files. These files are the "property" of the file system and are not normally visible to the user.

primary fileset
> A fileset that contains files that are visible to and accessible by users.

Structural files exist in the attribute fileset only and include the following:

fileset header file
> A file that contains a series of fileset headers.

inode list file
> A file that contains a series of inodes.

inode allocation unit (IAU) file
> A file that contains a series of inode allocation units.

current usage table (CUT) file
> A file that contains a series of fileset usage entries.

link count table file
> A file that contains a link count for each inode in the attribute fileset.

quotas file
> A file containing user quota information (for the primary fileset only).

Structural files and their components are discussed in the sections that follow.

Although structural files are contained in the structural fileset, they can "belong" to another fileset. For example, the inode list file for the primary fileset is in the structural fileset, but the structural details that it contains are only applicable to the primary fileset.

Each fileset is defined by structural files as follows:

- an inode list file, which contains the inodes belonging to the fileset

- an inode allocation unit file, which contains a series of inode allocation units

- an entry in the fileset header file, which contains one fileset header per fileset

- an entry in the current usage table file, which contains usage information for each fileset

In addition, the primary fileset has a user quotas file and the structural fileset has a link count table file.

Fileset metadata that cannot be reconstructed using the inode list is replicated to help fsck reconstruct the file system in the event of disk damage.

Figure 5 shows a fileset and the structural files by which it is defined.

Fileset Header

Each fileset has a header containing information about the fileset's contents and characteristic. All fileset headers are stored in a single fileset header file in the attribute fileset. The fileset header file contains one fileset header per fileset (see Figure 6). Each fileset header entry is 1 block long. The fileset header file is replicated because fileset headers cannot be rebuilt from other data structures.

Figure 5   Filesets and Structural Files

Fileset

Inode List File   IAU File

| Inode 0 | IAU 0 |

Fileset Header File

| Inode 1 | IAU 1 |

| Fileset Header Entry | | |

| Inode 2 | IAU 2 |

CUT File

| . . . | . . . |

| CUT Entry | | |

| Inode *n* | IAU *n* |

Quotas File
(primary fileset only)

LCT File (attribute fileset only)
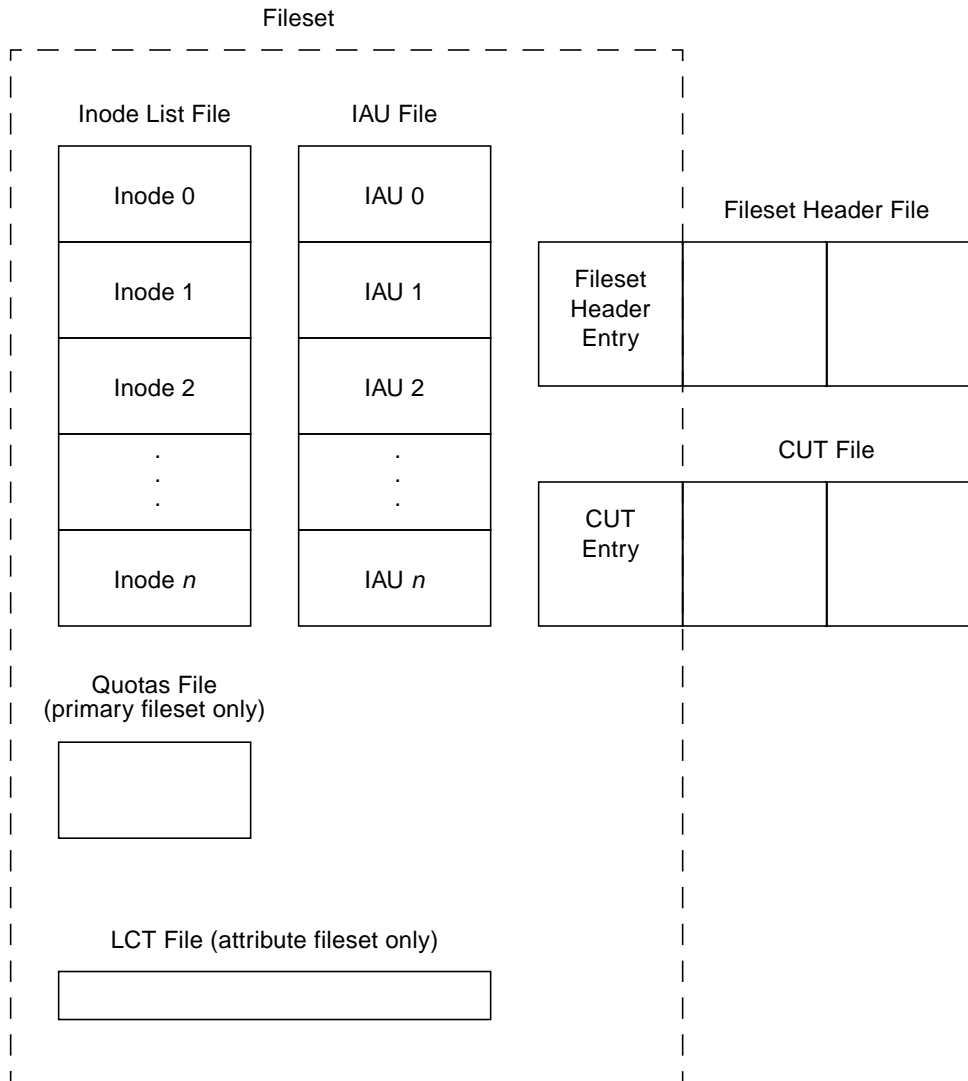
Figure 6   Fileset Header File
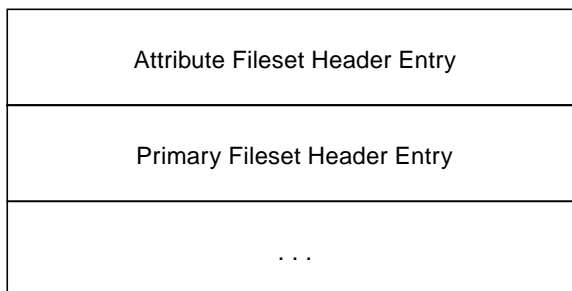
The fileset header for a given fileset includes information such as:

- the fileset index (1 for the attribute fileset and 999 for the primary fileset)

- the fileset name

- the inode numbers of the fileset's inode list file and its replica

- the total number of allocated inodes

- the maximum number of inodes allowed in the fileset

- the inode list extent size (in blocks)

- the inode number of the file containing the inode allocation units for the fileset

- the inode number of the fileset's link count table (attribute fileset only)

- the inode number of the fileset's quotas file (primary fileset only)

Note: The quotas file inode is present only in VxFS 2.3 and later and is not applicable to earlier releases of VxFS.

Inodes

An inode is a data structure that contains information about a file. The VxFS inode size is 256 bytes. Each inode stores information about a particular file such as:

- file length

- link count

- owner and group IDs

- access privileges

- time of last access

- time of last modification

- pointers to the extents that contain the file's data

Refer to the inode_vxfs(4) manual page for details on the contents of a VxFS inode.

There are up to ten direct extent address size pairs per inode. Each direct extent address indicates the starting block number of a direct extent; direct extent sizes can vary. If all of the direct extents are used, two indirect address extents are available for use in each inode. The first indirect address extent is used for single indirection, where each entry in the extent indicates the starting block number of an indirect data extent. The second indirect address extent is used for double indirection, where each entry in the extent indicates the starting block number of a single indirect address extent. Each indirect address extent is 8K long and contains 2048 entries. All indirect data extents for a given file have the same size, which is determined when the file's first indirect data extent is allocated.

Version 2 inodes differ from Version 1 inodes in that they are located in structural files to facilitate *dynamic inode allocation*, which is the allocation of inodes on an as-needed basis. Instead of allocating a fixed number of inodes into the file system, mkfs allocates a minimum number of inodes. Additional inodes are later allocated as the file system needs them.

The *inode list* is a series of inodes located in the inode list file. There is one inode in the list for every file in a given fileset. For recovery purposes, the inode list file is referenced by two inodes that point to the *same* set of data blocks. Although the inode addresses are replicated for recovery purposes, the inodes themselves are not.

An *inode extent* is an extent that contains inodes and is 8K long, by default. Inode extents are dynamically allocated to store inodes as they are needed.

Initial Inode List Extents

The initial inode list extents contain the inodes first allocated by mkfs for each fileset in a file system. During file system use, inodes are allocated as needed and are added into the inode list files for the filesets.

Figure 7 shows the initial inode list extents allocated for the primary and attribute filesets. Each of these extents contain 32 inodes and is 8K long.

The construction of the primary fileset's inode list resembles that of the VxFS Version 1 file system layout, with the first two inodes reserved and inodes 2 and 3 pre-assigned to the root and lost+found directories. The structural fileset's inode list is similarly constructed, with certain inodes allocated for specific files and other inodes reserved or unallocated.

There are two initial inode list extents for the attribute fileset. These contain the inodes for all structural files needed to find and set up the file system.

Some of the entries in the structural fileset's inode list are replicas of one another. For example, inodes 4 and 36 both reference copies of the fileset header file. The replicated inodes are used by fsck to reconstruct the file system in the event of damage to either one of the replicas. Although the two initial inode list extents belonging to the attribute fileset are logically contiguous, they are physically separated. This helps to ensure the integrity of the replicated information and reduces the chance that localized disk damage might result in complete loss of the file system.

Note that inodes 6 and 38 in the attribute fileset reference the inode list file for the attribute fileset. In a newly created file system, this file contains the two inode extents pictured for the attribute fileset. Likewise, the attribute fileset inodes 7 and 39 reference the inode list file for the primary fileset. In a newly created file system, this file contains the single extent pictured for the primary fileset. All of the unused inodes in the initial extents of the structural inode list are reserved for future use.

Figure 7   Inode Lists

Primary Fileset Inode List		Attribute Fileset Inode List			
0		0		32	primary fileset quotas file
1		1		33	
2	root	2		34	
3	lost+found	3	CUT	35	LCT
4		4	fileset header	36	fileset header (replica)
5		5	attribute fileset IAU	37	primary fileset IAU
6		6	attribute fileset inode list	38	attribute fileset inode list (replica)
7		7	primary fileset inode list	39	primary fileset inode list (replica)
8		8		40	
...		...		...	
31		31		63	

Inode Allocation Unit

An Inode Allocation Unit (IAU) contains inode allocation information for a given fileset. Each fileset contains one or more IAUs, each of which details allocation for a set number of inodes. The number of inodes per IAU varies, depending on the block size being used. One IAU exists for every 16,384 inodes in a fileset with the default block size (1024 bytes). If an IAU is damaged, the information that it contains can be reconstructed by examining the fileset's inode list.

The IAUs for a fileset are stored in sequential order in the fileset's IAU file. The fileset header identifies the attribute fileset inode associated with that fileset's IAU file.

Figure 8 shows the inode allocation unit structure. All IAU components begin on a block boundary.

Figure 8   Inode Allocation Unit (IAU) Structure

IAU Header
IAU Summary
Free Inode Map
Extended Inode Operations Map
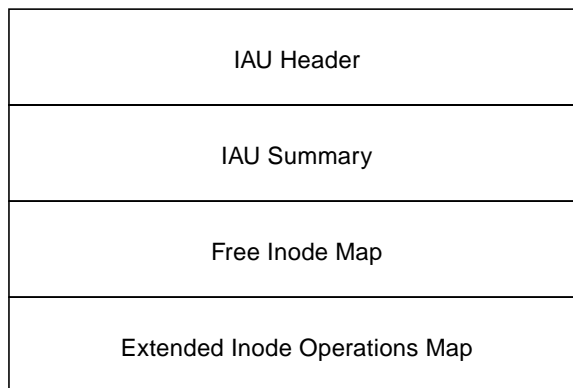
IAU Header

The IAU header verifies that the inode allocation unit matches the fileset. The IAU header occupies the first block of each inode allocation unit. If damaged, the IAU can be reconstructed from inodes and other information.

IAU Summary

The IAU summary summarizes the resources used in the IAU. It includes information on the number of free inodes in the IAU and the number of inodes with extended operation sets in the IAU. The IAU summary is 1 block long.

Free Inode Map

The free inode map is a bitmap that indicates which inodes are free and which are allocated. A free inode is indicated by the bit being on. The length of the free inode map is 2K for file systems with 1K or 2K block sizes and is equal to the block size for file systems with larger block sizes.

Extended Inode Operations Map

The extended inode operations map keeps track of inodes on which operations would remain pending for too long to reside in the intent log. The extended inode operations map is in the same format as the free inode map. To prevent the intent log from wrapping and the transaction from getting overwritten, the required operations are stored in the affected inode. This map is then updated to identify the inodes that have extended operations that need to be completed. This map allows the fsck utility to quickly identify which inodes had extended operations pending at the time of a system failure. The length of the extended inode operations map is 2K for file systems with 1K or 2K block sizes and is equal to the block size for file systems with larger block sizes.

Link Count Table

The link count table (LCT) contains a reference count for each inode in the associated fileset. This reference count is identical to the conventional link field of an inode. Each LCT entry contains the actual reference count for the associated fileset inode. The link count field in an inode itself is set to either 0 or 1, and the actual number of links is stored in the LCT entry for the associated fileset inode.

The link count table can be reconstructed using the inode list, so it is not replicated.

The current layout only uses the LCT for inodes in the attribute fileset. The LCT supports quick updates of the link count for structural fileset inodes.

Current Usage Table

The current usage table (CUT) is a file that contains usage related information for each fileset. The information contained in the CUT changes frequently and is not replicated. The information in the CUT can, however, be reconstructed using the inode list if the CUT is damaged.

The CUT file contains one entry per fileset (see Figure 9). The CUT entry for a given fileset contains information such as the following:

- The number of blocks currently used by the fileset.

- The *fileset version number*, which is a 64-bit integer that is guaranteed to be at least as large as the largest inode version number. An *inode version number* is a 64-bit integer that is incremented every time its inode is modified or written to disk and can be used to indicate whether an inode has been modified in any way since the last time it was examined. It is possible to find out which inodes have been modified since a specific time by saving the fileset version number and then later looking for inodes with a larger version number.

Figure 9   Current Usage Table (CUT) File

Attribute Fileset CUT Entry
Primary Fileset CUT Entry
. . .

Quotas File

VxFS supports BSD-style quotas for users. Quota information is stored in a quotas file. User quotas files track the resources used by each user ID. The quotas file keeps track of soft limits, hard limits, block usage, and inode usage for users within a file system.

Because quotas apply to mountable filesets only, the attribute fileset does not have quotas. However, the primary fileset's quotas file exists as a structural file in the attribute fileset. The primary fileset's user quotas file is referenced by the attribute fileset's initial inode list extent.

Note: The quotas file is present only in VxFS 2.3 and later and is not applicable to earlier releases of VxFS.

Locating Dynamic Structures

The existence of dynamic structures in the Version 2 disk layout makes the task of initially locating those structures difficult. The object location table (OLT) contains information needed to initially locate important file system structural elements. In particular, the OLT records the starting block numbers of the initial inode list extents for the attribute fileset and indicates which inodes within those initial extents reference the fileset header file.

Object Location Table Contents

The OLT is composed of records for the following:

fileset header inodes
> This record identifies the inode numbers of the fileset header file and its replica.

initial inode list extent addresses
> This record identifies the addresses of the beginning of each of two 8K inode extents. These are the initial inode list extents for the attribute fileset, which contain the inodes for all structural files belonging to the attribute fileset.

current usage table inode
> This record identifies the inode number of the file that contains the current usage table.

Mounting and the Object Location Table

At mount time, the object location table provides essential information about the location of key file system components. The super-block plays an important role in locating the OLT, in that it contains pointers to both the OLT and its replica.

Using the OLT, the process of mounting a VxFS Version 2 file system is as follows:

1. Read in the super-block. Validate the super-block and its replicas (located in the allocation unit headers).

2. Read and validate the OLT and its replica at the locations recorded in the super-block.

3. Obtain the addresses of the initial inode list extents for the attribute fileset from the OLT. Read in these initial inode extents.

4. Find the fileset header file, based on the fileset header file inode number recorded in the OLT.

5. Read the contents of the fileset header file. Each fileset header file entry represents a particular fileset and indicates the inode numbers of its inode list file and IAU file. The attribute fileset is set up first so that subsequent references to its inode list can be resolved.

The VxFS Version 4 Disk Layout

The Version 4 disk layout was designed to allow the file system to scale easily to accommodate large files and large file systems.

The Version 1 and 2 disk layouts divided up the file system space into allocation units. The first AU started part way into the file system which caused potential alignment problems depending on where the first AU started. Each allocation unit also had its own summary, bitmaps, and data blocks. Because this AU structural information was stored at the start of each AU, this also limited the maximum size of an extent that could be allocated. By replacing the allocation unit model of previous versions, the need for alignment of allocation units and the restriction on extent sizes was removed.

The VxFS Version 4 disk layout divides the entire file system space into fixed size allocation units. The first allocation unit starts at block zero and all allocation units are a fixed length of 32K blocks. (An exception may be the last AU, which occupies whatever space remains at the end of the file system). Because the first AU starts at block zero instead of part way through the file system as in previous versions, there is no longer a need for explicit AU alignment or padding to be added when creating a file system (see mkfs(1M)).

The Version 4 file system also moves away from the model of storing AU structural data at the start of an AU and puts all structural information in files. So expanding the file system structures simply requires extending the appropriate structural files. This removes the extent size restriction imposed by the Version 1 and Version 2 layouts.

All Version 4 structural files reside in the *structural fileset*, which is similar to the Version 2 attribute fileset. The structural files in the Version 4 disk layout are:

Object Location Table File
> Contains the object location table (OLT). As with the Version 2 disk layout, the OLT, which is referenced from the super-block, is used to locate the other structural files.

Label File Encapsulates the super-block and super-block replicas. Although the location of the primary super-block is known, the label file can be used to locate super-block copies if there is structural damage to the file system.

Device File
> Records device information such as volume length and volume label, and contains pointers to other structural files.

Fileset Header File
> Holds information on a per-fileset basis. This may include the inode of the fileset's inode list file, the maximum number of inodes allowed, an indication of whether the file system supports large files, and the inode number of the quotas file if the fileset supports quotas.
>
> When a file system is created, there are two filesets, the structural fileset, which defines the file system structure, and the *primary fileset*, which contains user data.

Inode List File
> Both the structural fileset and the primary fileset have their own inode lists which are stored in inode list files. Increasing the number of inodes involves increasing the size of the file after expanding the inode allocation unit 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

If the file system supports quotas, there is a quotas file which is used to track the resources allocated to each user.

Figure 10 shows how the kernel and utilities build information about the structure of the file system. The super-block location is in a known location from which the OLT can be located. From the OLT, the initial extents of the structural inode list can be located along with the inode number of the fileset header file. The initial inode list extents contain the inode for the fileset header file from which the extents associated with the fileset header file are obtained.

As an example, when mounting the file system, the kernel needs to access the primary fileset in order to access its inode list, inode allocation unit, quotas file and so on. The required information is obtained by accessing the fileset header file from which the kernel can locate the appropriate entry in the file and access the required information.

Figure 10   VxFS Version 4 Disk Layout

▼ 2
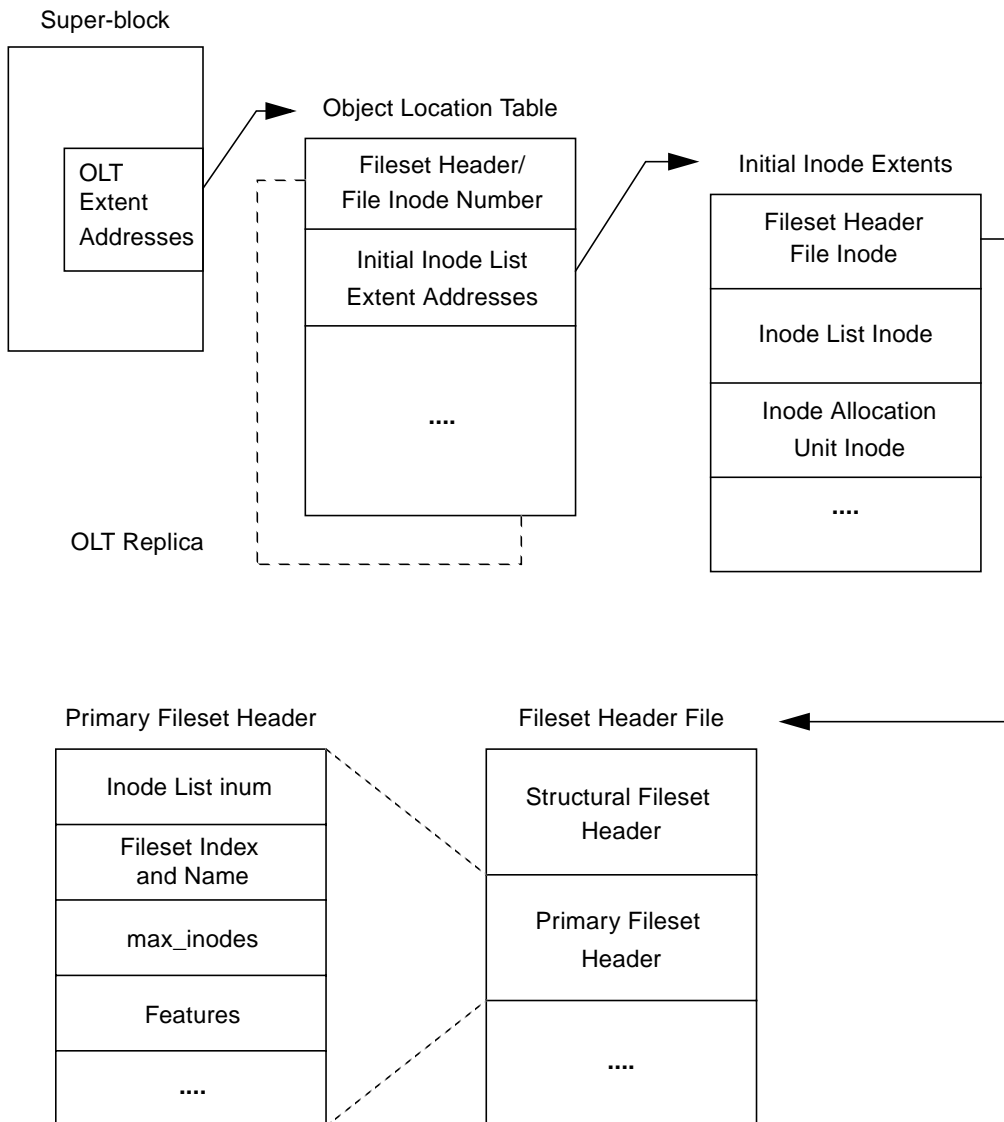
Extent Attributes

Introduction

The VERITAS file system allocates disk space to files in groups of one or more adjacent blocks called *extents*. VxFS defines an application interface that allows programs to control various aspects of the extent allocation for a given file (see Chapter 6, "Application Interface"). The extent allocation policies associated with a file are referred to as *extent attributes*.

The VxFS getext and setext commands allows users to 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*. The user can preallocate space to the file by manipulating a file's reservation; the user can also 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. The user can specify that:

- the space reserved for a file must be contiguous

- no allocations should be made for a file beyond the current reservation

- unused reservation should be released when the file is closed

- space should be allocated but no reservation should be assigned

- the file size should be 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 lost after the file is closed or the system is rebooted. The persistent attributes are similar to the file's permissions and are actually written in the inode for the file. When a file is copied, moved, or archived, only the persistent attributes of the source file can be preserved in the new file (see "Other Controls" on page 56 for more information).

In general, the user will only set extent attributes for reservation. Many of the attributes are designed for applications that are tuned to a particular pattern of I/O or disk alignment (see the mkfs_vxfs(1M) manual page and Chapter 6, "Application Interface," for more information).

Reservation: Preallocating Space to a File

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

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

Fixed Extent Size

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

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

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

Fixed extent sizes are particularly appropriate in the following situations:

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

• If a file is large and contiguous, a large fixed extent size can minimize the number of extents in the file.

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

Other Controls

The auxiliary controls on extent attributes determine:

• whether allocations are aligned

• whether allocations are contiguous

• whether the file can be written beyond its reservation

• whether 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 the introduction to Chapter 6, "Application Interface," for details). Alignment can only be specified if a fixed extent size has also been set. Setting alignment restrictions on allocations is best left to well designed applications.

Contiguity

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

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

Write Operations Beyond Reservation

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

Reservation Trimming

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

Reservation Persistence

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

Including Reservation in the File

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

Commands Related to Extent Attributes

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

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

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

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

warn	Issue a warning message if extent attribute information cannot be maintained (the default)
force	Fail the copy if extent attribute information cannot be maintained
ignore	Ignore extent attribute information entirely

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

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

- The `ignore` keyword loses any extent attributes for files.

- The `warn` keyword issues a warning if extent attributes for a file cannot be preserved. Such a situation may take place if the file is moved into a non-VxFS file system; the file would ultimately be moved while the extent attributes would be lost.

- The `force` keyword issues an error if attributes are lost and the file is not relocated.

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

Failure to Preserve Extent Attributes

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

- The file system receiving a copied, moved, or restored file from an archive is not a VxFS type. Since other file system types do not support the extent attributes of the VxFS file system, the attributes of the source file are lost during the migration.

- The file system receiving a copied, moved, or restored file is a VxFS type but does not have enough free space to satisfy the extent attributes. For example, consider a 50K file and a reservation of 1 MB. If the target file system has 500K free, it could easily hold the file but fail to satisfy the reservation.

- The file system receiving a copied, moved, or restored file from an archive is a VxFS type but the different block sizes of the source and target file system make extent attributes impossible to maintain. For example, consider a source file system of block size 1024, a target file system of block size 4096, and a file that has a fixed extent size of 3 blocks (3072 bytes). This fixed extent size adapts to the source file system but cannot translate onto the target file system.

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

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

Online Backup

 4

Introduction

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

The following topics are covered in this chapter:

- Snapshot File Systems

  - Snapshot File System Disk Structure

  - How a Snapshot File System Works

- Using a Snapshot File System for Backup

  - Creating a Snapshot File System

  - Making a Backup

- Performance of Snapshot File Systems

Snapshot File Systems

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

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

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

Snapshot File System Disk Structure

A snapshot file system consists of:

- a super-block
- a bitmap
- a blockmap
- data blocks copied from the snapped file system

Figure 11 shows the disk structure of a snapshot file system.

Figure 11   The Snapshot Disk Structure

Super-Block
Bitmap
Blockmap
Data Blocks
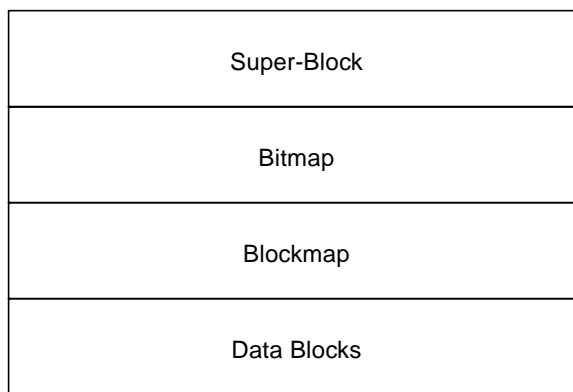
The super-block is similar to the super-block of a normal VxFS file system, however, the magic number is different and many of the fields are meaningless.

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

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

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

How a Snapshot File System Works

A snapshot file system is created by mounting an empty disk slice as a snapshot of a currently mounted file system. The bitmap, blockmap and super-block are initialized and then the currently mounted file system is frozen (see Chapter 6, "Application Interface," for a description of the VX_FREEZE ioctl). Once the file system to be snapped is frozen, the snapshot is enabled and mounted and the snapped file system is thawed. The snapshot appears as an exact image of the snapped file system at the time the snapshot was made.

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

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

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

The amount of disk space required for the snapshot depends on the rate of change of the snapped file system and the amount of time the snapshot is maintained. In the worst case, the snapped file system is completely full and every file is removed and rewritten. The snapshot file system would need enough blocks to hold a copy of every block on the snapped file system, plus additional blocks for the data structures that make up the snapshot file system. This is approximately 101 percent of the size of the snapped file system. Normally, most file systems do not undergo changes at this extreme rate. During periods of low activity, the snapshot should only require 2 to 6 percent

of the blocks of the snapped file system. During periods of high activity, the snapshot might require 15 percent of the blocks of the snapped file system. These percentages tend to be lower for larger file systems and higher for smaller ones.

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

Using a Snapshot File System for Backup

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

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

If a complete backup of a snapshot file system is made through a utility such as `volcopy` and is later restored, it will be necessary to `fsck` the restored file system because the snapshot file system is only consistent and not clean. The file system may have some extended inode operations that must be completed,

---

though there should be no other changes. Since the snapshot file system is not writable, it cannot be fully fscked. However, the fsck -n command can be used to report any inconsistencies.

Creating a Snapshot File System

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

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

The snapshot file system must be created large enough to hold any blocks on the snapped file system that may be written to while the snapshot file system exists.

Note: If a snapshot file system runs out of blocks to hold copied data, it will be disabled and all further access to the snapshot file system will fail.

During a period of low activity when the system is relatively inactive (for example, on nights and weekends), the snapshot only needs to contain 2 to 6 percent of the blocks of the snapped file system. During a period of higher activity, the snapshot of an "average" file system might require 15 percent of the blocks of the snapped file system, though most file systems do not experience this much turnover of data over an entire day. These percentages tend to be lower for larger file systems and higher for smaller ones. The system administrator should manage the blocks allocated to the snapshot based on file system usage, duration of backups, etc.

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

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

Making a Backup

Here are some typical examples of making a backup of a 300,000 block file system named /home (which exists on disk /dev/dsk/c0t0d0s7) using a snapshot file system on /dev/dsk/c0t1d0s1 with a snapshot mount point of /backup/home:

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

```
# mount -F vxfs -o snapof=/dev/dsk/c0t0d0s7, \
snapsize=100000 /dev/dsk/c0t1d0s1 /backup/home

# cd /backup

# find home -ctime -7 -depth -print | \
cpio -oc > /dev/rmt/c0s0

# umount /backup/home
```

- To back up entire file system using volcopy:

```
# mount -F vxfs -o snapof=/dev/dsk/c0t0d0s7, \
snapsize=100000 /dev/dsk/c0t1d0s1 /backup/home

# volcopy -F vxfs home /dev/rdsk/c0t1d0s1 \
c0t1d0s1 /dev/rmt/c0s0 tape77

# umount /backup/home
```

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

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

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

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

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

```
# mount -o snapof=/dev/dsk/c0t0d0s7,snapsize=100000 \
/dev/dsk/c0t1d0s1 /backup/home

# vxdump f - /dev/rdsk/c0t1d0s1 | \
dd bs=128k > /dev/rmt/c0s0
```

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

Performance of Snapshot File Systems

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

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

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

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

Performance and Tuning 5

Introduction

For any file system, the ability to provide peak performance is important. Adjusting the available VERITAS File System options provides a way to optimize system performance. This chapter describes tools that an administrator can use to optimize VxFS. To optimize an application for use with VxFS, see Chapter 6, "Application Interface."

The following topics are covered in this chapter:

- Choosing a Block Size

- Choosing an Intent Log Size

- Choosing Mount Options

    - `log`

    - `delaylog`

    - `tmplog`

    - `nolog`

    - `nodatainlog`

    - `blkclear`

    - `mincache`

    - `convosync`

    - `vxldlog`

- • Combining mount Options
- Kernel Tuneables
  - • Internal Inode Table Size
  - • VxVM Maximum I/O Size
- Monitoring Free Space
  - • Monitoring Fragmentation
- I/O Tuning
  - • Tuning VxFS I/O Parameters
  - • Tuneable VxFS I/O Parameters

Choosing a Block Size

Note: The block size is chosen when a file system is created and cannot be changed later.

The standard ufs file system defaults to an 8K block size with a 1K fragment size. This means that space is allocated to small files (up to 8K) in 1K increments. Allocations for larger files are done in 8K increments, except for the last block which may be a fragment. Since many files are small, the fragment facility saves a large amount of space when compared to allocating space 8K at a time.

The unit of allocation in VxFS is a block. There are no fragments, since storage is allocated in extents that consist of one or more blocks. For the most efficient space utilization, the smallest block size available on the system should be used. Typically, this provides the best performance as well. The smallest block size available is 1K, which is also the default block size for VxFS file systems created on the system. Unless there are special concerns, there should never be a need to specify a block size when creating file systems.

For large file systems, with relatively few files, the system administrator may wish to experiment with larger block sizes. The trade off of specifying larger block sizes is a decrease in the amount of space used to hold the free extent bitmaps for each allocation unit, an increase in the maximum extent size, and a decrease in the number of extents used per file versus an increase in the

amount of space wasted at the end of files that are not a multiple of the block size. Larger block sizes use less disk space in file system overhead, but consume more space for files that are not a multiple of the block size.

Overall file system performance may be improved or degraded by changing the block size. For most applications, it is recommended that the default values for the system be used. Some applications may benefit from a larger block size. The easiest way to judge which block sizes will provide the greatest system efficiency is to try representative system loads against various sizes and pick the fastest.

Choosing an Intent Log Size

Note: The intent log size is chosen when a file system is created and cannot be subsequently changed.

The mkfs utility uses a default intent log size of 1024 blocks. The default size is sufficient for most workloads. If the system is used as an NFS server or for intensive synchronous write workloads, performance may be improved using a larger log size.

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

Note: When a larger intent log size is chosen, recovery time will be proportionately longer and the file system may consume more system resources (such as memory) during normal operation.

Note: When QuickLog is used, the log is chosen by the system administrator at creation time and is easily changed at any time during use. For more information on log creation, log manipulation, and load balancing, see Chapter 9, "VERITAS QuickLog."

Choosing Mount Options

In addition to the standard mount mode (log mode), VxFS provides blkclear, delaylog, tmplog, nolog, 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 are capable of significantly improving 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, rename, etc.) may show over 100 percent improvement. The best way to select a mode is to test representative system loads against the logging modes and compare the performance results.

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

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

log

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

delaylog

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

tmplog

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

nolog

Same as `tmplog`.

nodatainlog

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

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

A `nodatainlog` mode file system should be 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 should be approximately 10 percent slower than a standard mode VxFS file system, depending on the workload.

mincache

The `mincache` mode has five suboptions:

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

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

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

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

Since the `mincache=direct`, `mincache=unbuffered`, and `mincache=dsync` modes change non-synchronous I/O to synchronous I/O, there can be a substantial degradation in throughput for small to medium size files for most applications. Since the `VX_DIRECT` and `VX_UNBUFFERED` advisories do not allow any caching of data, applications that would normally

benefit from caching for reads will usually experience less degradation with the `mincache=dsync` mode. `mincache=direct` and `mincache=unbuffered` require significantly less CPU time than buffered I/O.

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

convosync

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

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

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

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

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

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

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

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

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

vxldlog

`vxldlog` 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 `vxldlog=vxlog1`. If `vxldlog=` is specified with no QuickLog device, the QuickLog driver chooses an appropriate log device automatically.

For more information, see Chapter 9, "VERITAS QuickLog."

Combining `mount` **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.

▼ 5

Kernel Tuneables

This section describes the kernel tuneables in VxFS.

Internal Inode Table Size

Inodes are cached in a "per file system table," known as the *inode table*. Each file system type has a tuneable to determine the number of entries in its inode table. For the VxFS file system, the tuneable is `vxfs_ninode`. For the `ufs` file system, the tuneable is `ufs_ninode`.

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

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

It may be necessary to tune the `dnlc` (directory name lookup cache) size to keep the value within an acceptable range relative to `vxfs_ninode`. It must be within 80% of `vxfs_ninode` to avoid spurious `ENFILE` errors or excessive CPU consumption, but must be more than 50% of `vxfs_ninode` to maintain good performance. On Solaris 2.6, the variable `ncsize` determines the size of `dnlc`. The default value of `ncsize` is based on the kernel variable `maxusers`. It is computed at system boot time. This value can be changed by making an entry in the `/etc/system` file:

```
set ncsize = new_value
```

The new `ncsize` is effective after the system has been rebooted.

VxVM Maximum I/O Size

If the file system is being used in conjunction with the VERITAS Volume Manager, then the Volume Manager by default breaks up I/O requests larger than 256K. If you are using striping, for optimal performance, the file system issues I/O requests that are full stripes. If the stripe size is larger than 256K, those requests are broken up.

To avoid undesirable I/O breakup, the vol_maxio parameter should be increased. To increase the value of vol_maxio, add an entry to /etc/system 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 sectors and is stored as a 16-bit number, so it cannot be set to a value larger than 65535. The value of vol_maxio determines the largest amount of memory that an I/O request can lock, so it should not be set to more than approximately 20 percent of memory.

Monitoring Free Space

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

Monitoring Fragmentation

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

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

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

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

An unfragmented file system will have the following characteristics:

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

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

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

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

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

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

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

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

I/O Tuning

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

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

Tuning VxFS I/O Parameters

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

If the VxFS file system is being used with the VERITAS Volume Manager, the file system queries the Volume Manager to find out the geometry of the underlying volume and automatically sets the I/O parameters. The Volume Manager is queried by mkfs when the file system is created to automatically align the file system to the volume geometry. Then the mount command queries the Volume Manager 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 the Volume Manager, 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 the Volume Manager. While the file system is mounted, any I/O parameters can be changed using the vxtunefs command which can have tuneables specified on the command line or can read them from the /etc/vx/tunefstab file. For more details, see the vxtunefs(1M) and tunefstab(4) manual pages. The vxtunefs command can be used to print the current values of the I/O parameters.

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

Tuneable VxFS I/O Parameters

The tuneable VxFS I/O parameters are:

read_pref_io

> The preferred read request size. The file system uses this in conjunction with the read_nstream value to determine how much data to read ahead. The default value is 64K.

read_nstream

> The desired number of parallel read requests of size read_pref_io to have outstanding at one time. The file system uses the product of read_nstream multiplied by read_pref_io to determine its read ahead size. The default value for read_nstream is 1.

read_unit_io

> This is a less preferred request size. Currently, the file system does not use this tuneable.

write_pref_io

> The preferred write request size. The file system uses this in conjunction with the write_nstream value to determine how to do flush behind on writes. The default value is 64K.

write_nstream

> The desired number of parallel write requests of size write_pref_io to have outstanding at one time. The file system uses the product of write_nstream multiplied by write_pref_io to determine when to do flush behind on writes. The default value for write_nstream is 1.

write_unit_io

> This is a less preferred request size. Currently, the file system does not use this tuneable.

pref_strength

> Indicates to the file system how large a performance gain might be made by adhering to the preferred I/O sizes. The file system does not use this tuneable.

buf_breakup_size

> Tells the file system how large an I/O it can issue without a driver breaking up the request. The file system does not use this tuneable.

max_direct_iosz

> The maximum size of a direct I/O request that will be issued by the file system. If a larger I/O request comes in, then it is broken up into max_direct_iosz chunks. This parameter defines how much memory an I/O request can lock at once, so it should not be set to more than 20 percent of memory.

discovered_direct_iosz

> Any file I/O requests larger than the discovered_direct_iosz are handled as discovered direct I/O. A discovered direct I/O is unbuffered similar to direct I/O, but it does not require a synchronous commit of the inode when the file is extended or blocks are allocated. For larger I/O requests, the CPU time for copying the data into the page cache and the cost of using memory to buffer the I/O data becomes more expensive than the cost of doing the disk I/O. For these I/O requests, using discovered direct I/O is more efficient than regular I/O. The default value of this parameter is 256K.

default_indir_size

> On VxFS, files can have up to 10 direct extents of variable size stored in the inode. Once these extents are used up, the file must use indirect extents which are a fixed size that is set when the file first uses indirect extents. These indirect extents are 8K by default. The file system does not use larger indirect extents because it must fail a write and return ENOSPC if there are no extents available that are the indirect extent size. For file systems with a lot of large files, the 8K indirect extent size is too small. The files that get into indirect extents use a lot of smaller extents instead of a few larger ones. By using this parameter, the default indirect extent size can be increased so large that files in indirects use fewer larger extents.

> The tuneable default_indir_size should be used carefully. If it is set too large, then writes will fail when they are unable to allocate extents of the indirect extent size to a file. In general, the fewer and the larger the files on a file system, the larger the default_indir_size can be set.

This parameter should generally be set to some multiple of the `read_pref_io` parameter.

`default_indir_size` is not applicable on Version 4 disk layouts.

max_diskq

Limits the maximum disk queue generated by a single file. When the file system is flushing data for a file and the number of pages being flushed exceeds `max_diskq`, processes will block until the amount of data being flushed decreases. Although this doesn't limit the actual disk queue, it prevents flushing processes from making the system unresponsive. The default value is 1 MB.

qio_cache_enable

Enables or disables caching on Quick I/O files. The default behavior is to disable caching. To enable caching, set `qio_cache_enable` to 1.

On systems with large memories, the database cannot always use all of the memory as a cache. By enabling file system caching as a second level cache, performance may be improved.

If the database is performing sequential scans of tables, the scans may run faster by enabling file system caching so the file system will perform aggressive read-ahead on the files.

max_seqio_extent_size

Increases or decreases the maximum size of an extent. When the file system is following its default allocation policy for sequential writes to a file, it allocates an initial extent which is large enough for the first write to the file. When additional extents are allocated, they are progressively larger (the algorithm tries to double the size of the file with each new extent) so each extent can hold several writes worth of data. This is done to reduce the total number of extents in anticipation of continued sequential writes. When the file stops being written, any unused space is freed for other files to use.

Normally this allocation stops increasing the size of extents at 2048 blocks which prevents one file from holding too much unused space. `max_seqio_extent_size` is measured in file system blocks.

`initial_extent_size`

> Changes the default initial extent size. VxFS determines, based on the first write to a new file, the size of the first extent to be allocated to the file. Normally the first extent is the smallest power of 2 that is larger than the size of the first write. If that power of 2 is less than 8K, the first extent allocated is 8K. After the initial extent, the file system increases the size of subsequent extents (see `max_seqio_extent_size`) with each allocation.
>
> Since most applications write to files using a buffer size of 8K or less, the increasing extents start doubling from a small initial extent. `initial_extent_size` can change the default initial extent size to be larger, so the doubling policy will start from a much larger initial size and the file system will not allocate a set of small extents at the start of file.
>
> This parameter should only be used on file systems that will have a very large average file size. On these file systems it will result in fewer extents per file and less fragmentation. `initial_extent_size` is measured in file system blocks.

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

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

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

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

5

Application Interface   6 ▼

Introduction

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

This chapter describes how the application writer can optimize applications for
use with the VxFS. To optimize VxFS for use with applications, see Chapter 5,
"Performance and Tuning."

The following topics are covered in this chapter:

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

▽ 6

Cache Advisories

The VxFS file system 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) man 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 transfer size must be a multiple of the sector size.

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

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

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

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

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

Unbuffered I/O

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

Discovered Direct I/O

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

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

For information on how to set the discovered_direct_iosz, see "I/O Tuning" in Chapter 5.

Data Synchronous I/O

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

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

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

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

Other Advisories

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

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

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

Extent Information

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

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

1. Use VX_GETEXT to read the current settings.

2. Modify the values to be changed.

3. Call VX_SETEXT to set the values.

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

Space Reservation

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

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

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

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

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

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

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

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

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

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

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

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

Fixed Extent Sizes

The VxFS file system 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 subpage I/O, it can request a fixed extent size to match its I/O size and avoid wasting extra space.

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

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

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

Freeze and Thaw

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

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

Note: While a file system is frozen by the VX_FREEZE ioctl, all logging performed by the QuickLog device is suspended. Logging of the file system is resumed once the VX_THAW ioctl is issued. For more information on the effects of these ioctls on QuickLog, see Chapter 9, "VERITAS QuickLog."

Get I/O Parameters ioctl

VxFS provides the VX_GET_IOPARAMETERS ioctl to get the recommended I/O sizes to use on a file system. This ioctl can be used by the application to make decisions about the I/O sizes issued to VxFS for a file or file device. For more details on this ioctl, refer to the vxfsio(7) manual page. For a discussion on various I/O parameters, refer to Chapter 5, "Performance and Tuning," and the vxtunefs(1M) manual page.

6

Quotas 7 ⩯

Introduction

The VERITAS File System supports BSD-style user quotas. The quota system limits the use of two principal resources of a file system: files and data blocks. For each of these resources, users may be assigned quotas.

The following topics are covered in this chapter:

- Quota Limits
- Quotas File on VxFS
- Quota commands
- `quotacheck` With VxFS
- Using Quotas

Quota Limits

Quota limits for individual users can be set up for file and data block usage on a file system. A user quota consists of limits for these resources. The following limits can be set for each resource:

- The *hard limit* is an absolute limit that cannot be exceeded under any circumstances.
- The *soft limit* (which is 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, and a default value of 7 days is set by VxFS. There are separate time limits for files and blocks.

An example of the use of soft limits is when the user needs to run applications that might generate large temporary files. In cases like these, quota limit violations may be allowed for a limited duration. However, if the user continuously exceeds the soft limit, no further allocations are allowed after the expiration of the time limit.

The system administrator is responsible for assigning hard and soft limits to the users, as well as setting associated time limits. Although file and data block limits can be set individually for each user, the time limits apply to the file system as a whole. Quota information associated with user IDs are stored in a quotas file, as described in Chapter 2, "Disk Layout."

Quotas File on VxFS

A *quotas* file (named `quotas`) must exist in the root directory of a file system for any of the quotas commands to work. This is a BSD quotas implementation requirement, and is also applicable to VxFS quotas. The `quotas` file in the root directory is referred to as the *external quotas* file. VxFS also maintains an *internal quotas* file for its internal use.

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

Quota commands

> **Note:** Most of the quotas commands in VxFS (as with `ufs`) are similar to BSD quotas commands. However, the `quotacheck` command is an exception—VxFS does not support an equivalent command. This is discussed in more detail in "quotacheck With VxFS."

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

VxFS supports the following quota-related commands:

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

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

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

`quotacheck` **With VxFS**

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

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

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

Using Quotas

To use the quotas functionality on a file system, quotas need to be turned on. Quotas can be turned on either at mount time or any time after a file system is mounted.

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

To turn on quotas for a VxFS file system, use the following commands:

    # vxquotaon /*mount_point*

Quotas can also be turned on for a file system at mount time by giving an option to the `mount` command:

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

`vxedquota` is a quota editor. User quotas can be set up with the `vxedquota` command by the superuser:

    # vxedquota *username*

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

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

Time limits can be modified using the command:

    # vxedquota -t

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

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

    # vxquota -v *username*

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

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

    # vxquotaoff /*mount_point*

Quick I/O for Databases 8 ▼

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 separately licensable product available from VERITAS as a Quick I/O Database accelerator option, or as part of a VERITAS Edition integrated product suite.

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

Operating systems such as Solaris 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.

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 which 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 similar performance as 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's a good idea 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.

Note: If the `qio` module is loaded after the VxFS file system is mounted, the file system must be unmounted and mounted again to be accessible by 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. The `qiomkfile` command has five options:

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

You can specify file size in terms of bytes, kilobytes, megabytes, gigabytes, or sectors (512 bytes) by adding a `k`, `K`, `m`, `M`, `g`, `G`, `s` or `S` suffix. The default is bytes. Unlike the VxFS `setext` command, which requires superuser privileges, any user who has read/write permissions can run `qiomkfile` to create the files.

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

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

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

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

```
$ ls -al

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

or:

```
$ ls -lL

crw-r----- 1  oracle    dba 43,  0    Aug 22 13:46 dbfile
-rw-r--r-- 1  oracle    dba 10485760  Aug 22 13:46 .dbfile
```

If you specify the -a option, an absolute pathname (see "Using Absolute or
Relative Pathnames") is used so /database/dbfile points to
/database/.dbfile::cdev:vxfs:. To check the results, enter:

```
$ ls -al

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

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

Accessing Regular VxFS Files Through Symbolic Links

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

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

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

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

Using Absolute or Relative Pathnames

It's usually better to use relative pathnames instead of absolute pathnames when creating symbolic links to access regular files as Quick I/O files. Using relative pathnames 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 pathnames.

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

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

Preallocating Files Using the `setext` **Command**

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

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

See the `setext`(1) man 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 package (`VRTSqio`) 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 `VRTSqio` 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

Databases can use a maximum of only 4 GB of memory for their databases 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, the 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 uses a direct-write, copy-behind technique. After the direct I/O is scheduled, and while it is waiting for the completion of the I/O, the file system updates its buffer to reflect the changes written to disk.

Cached Quick I/O also helps sequential table scan because of the read-ahead algorithm used in the VERITAS File System. For most queries that require sequential table scans, Cached Quick I/O can significantly reduce 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 and off. See the `vxtunefs`(1M) and `qioadmin`(1) online manual 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. Setting the `qio_cache_enable` parameter.

2. Enabling the Cached Quick I/O feature for specific Quick I/O files.

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

You enable Cached Quick I/O for a file system by setting the `qio_cache_enable` flag with 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. This command enables caching for all the Quick I/O files on this file system.

Note: This enables Cached Quick I/O for all files in the 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 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 *tunetable* 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
```

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 cache advisory for some files is ON.

To check on the current cache advisory settings for a file, use the -P option:

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

  names.dbf,OFF
```

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

```
$ vxtunefs -p /database01

  qio_cache_enable = 1
```

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

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

Tuning Cached Quick I/O

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

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

Quick I/O Statistics

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

Quick I/O Summary

To increase database performance on a VERITAS File System using Quick I/O:

1. Make sure that the `qio` module is loaded. You can add the following line to the file `/etc/system` to load `qio` whenever the system reboots.

    ```
    forceload: drv/qio
    ```

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

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

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

▼ 8

VERITAS QuickLog   9 ▼

Introduction

VERITAS QuickLog™ is an optionally licensable product included as part of the VERITAS File Server Edition for enhanced file system performance.

- VERITAS QuickLog Overview
- QuickLog Installation
- 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

Although QuickLog improves file system performance, VxFS does not require QuickLog to operate effectively. QuickLog is transparent to the end user and requires minimum intervention or training for a system administrator.

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 physical disk). 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 by the fact that the file system is performing other operations (inode updates, reading and writing data) that requires 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.

Figure 12 shows a logical view of QuickLog and how it interfaces with the operating system.

QuickLog Installation

The VERITAS QuickLog driver package (`VRTSqlog`) is dynamically loadable and unloadable using the `pkgadd` and `pkgrm` utilities. `pkgadd` modifies `/etc/devlink.tab` and creates the 32 QuickLog device nodes used to attach QuickLog volumes to the QuickLog driver. Each QuickLog device can attach a maximum of four volumes.

Figure 12   QuickLog Logical View
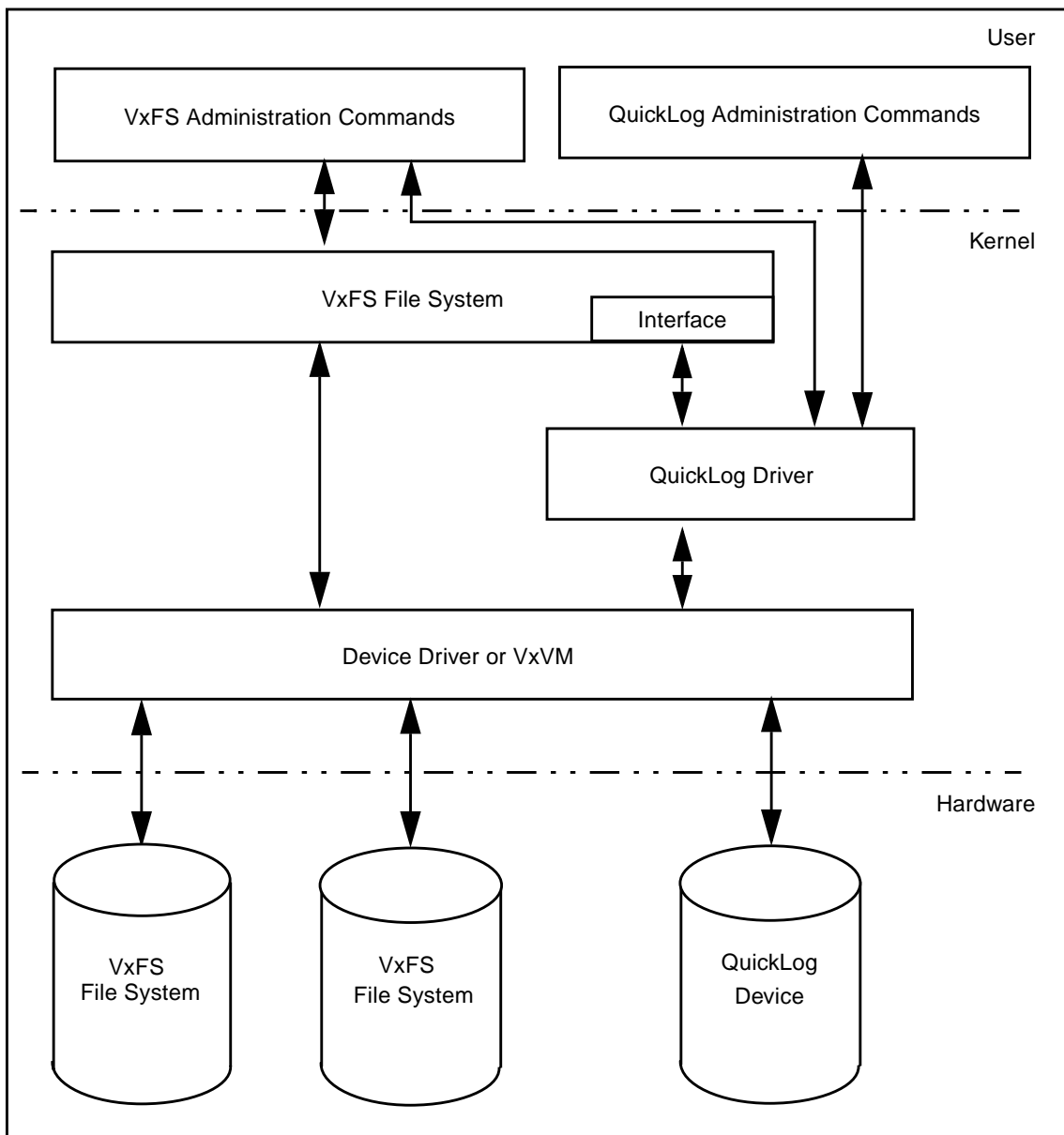
QuickLog Setup

VERITAS QuickLog supports:

- up to 32 QuickLog devices
- up to 32 VxFS file systems per QuickLog device
- from one to four QuickLog volumes per QuickLog device (see "QuickLog Load Balancing" for details)
- communication between QuickLog and VxFS through an integrated interface

QuickLog cannot be enabled for the root file system.

Creating a QuickLog Device

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

1. Create a VxVM volume using the command vxassist:

   # vxassist -g *diskgroup* make *qlog_volume* *size* [*vxvm_disk*]

or a raw disk partition using the format command.

If the QuickLog volume is a VxVM volume, it must reside in the same disk group as the file system to be logged. Each QuickLog volume should reside on a separate physical disk.

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

2. Build a QuickLog volume using the command vxld_mklog:

   # vxld_mklog -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 should be spread out across the one to four QuickLog volumes to be attached (see "QuickLog Load Balancing" for details).

The command `vxld_mklog` both writes out the QuickLog volume layout to the volume *qlog_volume* and attaches the QuickLog volume to the specified QuickLog device. Accepted QuickLog device names are `vxlog1` through `vxlog32`.

Removing a QuickLog Device

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

    # vxld_rmlog *qlog_volume*

`vxld_rmlog` 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 `vxld_rmlog` (see "`Disabling a QuickLog Device`" 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 VERITAS file system by QuickLog: the QuickLog utility `vxld_mntfs` and a VxFS special mount option.

The `-o vxldlog=` 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 VERITAS File System using `vxld_mntfs`:

```
# vxld_mntfs [qlog_device] /mount-point
```

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

```
# mount -F -o remount,vxldlog=[qlog_device] fsname /mount-point
```

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.

To ensure that QuickLog is enabled for a specific VERITAS file system after every system reboot, add "`vxldlog=`" 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 mountto fsck point     type  pass at boot options
#

/dev/vx/dsk/vol1 /dev/vx/rdsk/vol1 /vol1 vxfs 1 no \
vxldlog=
```

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

Disabling a QuickLog Device

To disable logging by QuickLog without unmounting a VERITAS File System, use the `vxld_umntfs` command:

```
# vxld_umntfs /mount-point
```

Make sure to disable QuickLog devices for all mounted and logged VERITAS File Systems and detach all QuickLog volumes before unloading the QuickLog driver (see `vxld_umntall`(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 32 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:

1. Detach a QuickLog volume from the QuickLog device by using `vxld_umntlog`.

   # vxld_umntlog *qlog_volume*

2. Remove the QuickLog volume from the QuickLog configuration by using `vxld_rmlog`.

   # vxld_rmlog *qlog_volume*

Before the QuickLog volume is detached, `vxld_umntlog` 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 `vxld_mklog`:

   # vxld_mklog -g *diskgroup* vxlog[x] *qlog_volume*

If a QuickLog volume already exists, attach the volume by using `vxld_mntlog`:

   # vxld_mntlog vxlog[x] *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 VERITAS 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 I/O read and write operations

- the average number of bytes per second for I/O read and write operations

- the average service time for I/O read and write operations

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

QuickLog Recovery

Note: No user intervention is required.

During the boot sequence, the QuickLog start-up script `/etc/init.d/vxld-startup` searches the QuickLog configuration file `/etc/vxld/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 VERITAS File System(s) 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 `vxld_logck`(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 VERITAS File Systems logged by 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 `vxld_mntall`, which reattaches all recovered QuickLog volumes. The QuickLog volumes must be reattached before you can remount VERITAS File Systems to log with QuickLog.

$\overline{\underline{\underline{\blacktriangledown}}}$ 9

Kernel Messages   A

Introduction

This appendix contains a listing of diagnostic or error messages generated by the VERITAS File System kernel. Each message is accompanied by an explanation 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 the File System
  - Recovering a Disabled File System
- Kernel Messages
  - Global Message IDs

▼ A

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 doesn't 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 the 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 operation still work, most simply return EIO. The only thing that can be done when the file system is disabled is to do a umount and run a full fsck.

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

```
# fsck -F vxfs -o full -y /dev/rdsk/c1t0d0s1
```

The file system usually becomes disabled because of disk errors. Disk failures that disabled 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 Explanation sub-section for each message describes the problem, the Action sub-section suggests possible solutions.

Global Message IDs

Each time a VxFS kernel message is displayed on the system console, it is displayed along with a monatomically increasing message ID, shown in the msgcnt field. This ID guarantees that the sequence of events is known in order to help analyze file system problems.

Each message is also written to an internal kernel buffer and can be viewed 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 is 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: 001

```
NOTICE: msgcnt x: vxfs: mesg 001: vx_nospace - mount_point file system
full (n block extent)
```

▼ Explanation

The file system is out of space.

Often, there is plenty of space and one runaway process used up all the remaining free space. In other cases, the available free space becomes fragmented and unusable for some files.

A

▼ Action

Monitor the free space in the file system and prevent it from becoming full. If a runaway process has used up all the space, stop that process, find the files created by the process, and remove them. If the file system is out of space, remove files, defragment, or expand the file system.

To remove files, use the `find` command to locate the files that are to be removed. To get the most space with the least amount of work, remove large files or file trees that are no longer needed. To defragment or expand the file system, use `fsadm` (see the `fsadm`(1M) manual page).

Message: 002

```
WARNING: msgcnt x: vxfs: mesg 002: vx_snap_strategy - mount_point file
system write attempt to read-only file system
```

```
WARNING: msgcnt x: vxfs: mesg 002: vx_snap_copyblk - mount_point file
system write attempt to read-only file system
```

▼ Explanation

The kernel tried to write to a read-only file system. This is an unlikely problem, but if it occurs, the file system is disabled.

▼ Action

The file system was not written, so no action is required. Report this as a bug to your customer support organization.

Message: 003, 004, 005

```
WARNING: msgcnt x: vxfs: mesg 003: vx_mapbad - mount_point file system
free extent bitmap in au aun marked bad
```

```
WARNING: msgcnt x: vxfs: mesg 004: vx_mapbad - mount_point file system
free inode bitmap in au aun marked bad
```

```
WARNING: msgcnt x: vxfs: mesg 005: vx_mapbad - mount_point file system
inode extended operation bitmap in au aun marked bad
```

▼ Explanation

If there is an I/O failure while writing a bitmap, the map is marked bad. The kernel considers the maps to be invalid, so does not do any more resource allocation from maps. This situation can cause the file system to report "out of space" or "out of inode" error messages even though df may report an adequate amount of free space.

This error may also occur due to bitmap inconsistencies. If a bitmap fails a consistency check, or blocks are freed that are already free in the bitmap, the file system has been corrupted. This may have occurred because a user or process wrote directly to the device or used fsdb to change the file system.

The VX_FULLFSCK flag is set. If the map that failed was a free extent bitmap, and the VX_FULLFSCK flag can't be set, then the file system is disabled.

▼ Action

Check the console log for I/O errors. If the problem is a disk failure, replace the disk. If the problem is not related to an I/O failure, find out how the disk became corrupted. If no user or process was writing to the device, report the problem to your customer support organization. Unmount the file system and use fsck to run a full structural check.

Message: 006, 007

```
WARNING: msgcnt x: vxfs: mesg 006: vx_sumupd - mount_point file system
summary update in au aun failed
```

```
WARNING: msgcnt x: vxfs: mesg 007: vx_sumupd - mount_point file system
summary update in inode au iaun failed
```

▼ Explanation

An I/O error occurred while writing the allocation unit or inode allocation unit bitmap summary to disk. This sets the VX_FULLFSCK flag on the file system. If the VX_FULLFSCK flag can't be set, the file system is disabled.

▼ Action

Check the console log for I/O errors. If the problem was caused by a disk failure, replace the disk before the file system is mounted for write access, and use fsck to run a full structural check.

≡ A

Message: 008, 009

```
WARNING: msgcnt x: vxfs: mesg 008: vx_direrr – mount_point file system
inode inumber block blkno error errno
```

```
WARNING: msgcnt x: vxfs: mesg 009: vx_direrr – mount_point file system
inode inumber immediate directory error errno
```

▼ Explanation

A directory operation failed in an unexpected manner. The mount point, inode, and block number identify the failing directory. If the inode is an immediate directory, the directory entries are stored in the inode, so no block number is reported. If the error is ENOENT or ENOTDIR, an inconsistency was detected in the directory block. This inconsistency could be a bad free count, a corrupted hash chain, or any similar directory structure error. If the error is EIO or ENXIO, an I/O failure occurred while reading or writing the disk block.

The VX_FULLFSCK flag is set in the super-block so that fsck will do a full structural check the next time it is run.

▼ Action

Check the console log for I/O errors. If the problem was caused by a disk failure, replace the disk before the file system is mounted for write access. Unmount the file system and use fsck to run a full structural check.

Message: 010

```
WARNING: msgcnt x: vxfs: mesg 010: vx_ialloc – mount_point file system
inode inumber not free
```

▼ Explanation

When the kernel allocates an inode from the free inode bitmap, it checks the mode and link count of the inode. If either is non-zero, the free inode bitmap or the inode list is corrupted.

The VX_FULLFSCK flag is set in the super-block so that fsck will do a full structural check the next time it is run.

▼ Action

Unmount the file system and use fsck to run a full structural check.

Message: 011

```
NOTICE: msgcnt x: vxfs: mesg 011: vx_noinode – mount_point file system
out of inodes
```

▼ Explanation

The file system is out of inodes.

▼ Action

Monitor the free inodes in the file system. If the file system is getting full, create more inodes either by removing files or by expanding the file system. File system resizing is described in Chapter 1, "The VERITAS File System," and in the fsadm(1M) online manual page.

Message: 012

```
WARNING: msgcnt x: vxfs: mesg 012: vx_iget – mount_point file system
invalid inode number inumber
```

▼ Explanation

When the kernel tries to read an inode, it checks the inode number against the valid range. If the inode number is out of range, the data structure that referenced the inode number is incorrect and must be fixed.

The VX_FULLFSCK flag is set in the super-block so that fsck will do a full structural check the next time it is run.

▼ Action

Unmount the file system and use fsck to run a full structural check.

Message: 013

```
WARNING: msgcnt x: vxfs: mesg 013: vx_iposition – mount_point file
system inode inumber invalid inode list extent
```

▼ Explanation

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.

≡ A

The `VX_FULLFSCK` flag is set in the super-block and the file system is disabled.

▼ Action

Unmount the file system and use `fsck` to run a full structural check.

Message: 014

```
WARNING: msgcnt x: vxfs: mesg 014: vx_iget - inode table overflow
```

▼ Explanation

All the system in-memory inodes are busy and an attempt was made to use a new inode.

▼ Action

Look at the processes that are running and determine which processes are using inodes. If it appears there are runaway processes, they might be tying up the inodes. If the system load appears normal, increase the `vxfs_ninode` parameter in the kernel (see "Internal Inode Table Size" in Chapter 5, "Performance and Tuning").

Message: 015

```
WARNING: msgcnt x: vxfs: mesg 015: vx_ibadinactive - mount_point file
system can't mark inode inumber bad
```

```
WARNING: msgcnt x: vxfs: mesg 015: vx_ilisterr - mount_point file
system can't mark inode inumber bad
```

▼ Explanation

An attempt to mark an inode bad on disk, and the super-block update to set the `VX_FULLFSCK` flag, failed. This indicates that a catastrophic disk error may have occurred since both an inode list block and the super-block had I/O failures. The file system is disabled to preserve file system integrity.

▼ Action

Unmount the file system and use `fsck` to run a full structural check. Check the console log for I/O errors. If the disk failed, replace it before remounting the file system.

Message: 016

```
WARNING: msgcnt x: vxfs: mesg 016: vx_ilisterr - mount_point file
system error reading inode inumber
```

▼ Explanation

An I/O error occurred while reading the inode list. The VX_FULLFSCK flag
is set.

▼ Action

Check the console log for I/O errors. If the problem was caused by a disk
failure, replace the disk before the file system is mounted for write access.
Unmount the file system and use fsck to run a full structural check.

Message: 017

```
WARNING: msgcnt x: vxfs: mesg 017: vx_attr_getblk - mount_point file
system inode inumber marked bad
```

```
WARNING: msgcnt x: vxfs: mesg 017: vx_attr_iget - mount_point file
system inode inumber marked bad
```

```
WARNING: msgcnt x: vxfs: mesg 017: vx_attr_indadd - mount_point file
system inode inumber marked bad
```

```
WARNING: msgcnt x: vxfs: mesg 017: vx_attr_indtrunc - mount_point file
system inode inumber marked bad
```

```
WARNING: msgcnt x: vxfs: mesg 017: vx_attr_iremove - mount_point file
system inode inumber marked bad
```

```
WARNING: msgcnt x: vxfs: mesg 017: vx_bmap - mount_point file system
inode inumber marked bad
```

```
WARNING: msgcnt x: vxfs: mesg 017: vx_bmap_indirect_ext4 - mount_point
file system inode inumber marked bad
```

```
WARNING: msgcnt x: vxfs: mesg 017: vx_delbuf_flush - mount_point file
system inode inumber marked bad
```

```
WARNING: msgcnt x: vxfs: mesg 017: vx_dio_iovec - mount_point file
system inode inumber marked bad
```

```
WARNING: msgcnt x: vxfs: mesg 017: vx_dirbread - mount_point file
system inode inumber marked bad
```

```
WARNING: msgcnt x: vxfs: mesg 017: vx_dircreate - mount_point file
system inode inumber marked bad
```

```
WARNING: msgcnt x: vxfs: mesg 017: vx_dirlook - mount_point file system
inode inumber marked bad
```

```
WARNING: msgcnt x: vxfs: mesg 017: vx_doextop_iau - mount_point file
system inode inumber marked bad
```

WARNING: msgcnt *x*: vxfs: mesg 017: vx_doextop_now – *mount_point* file system inode *inumber* marked bad

WARNING: msgcnt *x*: vxfs: mesg 017: vx_do_getpage – *mount_point* file system inode *inumber* marked bad

WARNING: msgcnt *x*: vxfs: mesg 017: vx_enter_ext4 – *mount_point* file system inode *inumber* marked bad

WARNING: msgcnt *x*: vxfs: mesg 017: vx_exttrunc – *mount_point* file system inode *inumber* marked bad

WARNING: msgcnt *x*: vxfs: mesg 017: vx_get_alloc – *mount_point* file system inode *inumber* marked bad

WARNING: msgcnt *x*: vxfs: mesg 017: vx_ilisterr – *mount_point* file system inode *inumber* marked bad

WARNING: msgcnt *x*: vxfs: mesg 017: vx_ilock – *mount_point* file system inode *inumber* marked bad

WARNING: msgcnt *x*: vxfs: mesg 017: vx_indtrunc – *mount_point* file system inode *inumber* marked bad

WARNING: msgcnt *x*: vxfs: mesg 017: vx_iread – *mount_point* file system inode *inumber* marked bad

WARNING: msgcnt *x*: vxfs: mesg 017: vx_iremove – *mount_point* file system inode *inumber* marked bad

WARNING: msgcnt *x*: vxfs: mesg 017: vx_iremove_attr – *mount_point* file system inode *inumber* marked bad

WARNING: msgcnt *x*: vxfs: mesg 017: vx_logwrite_flush – *mount_point* file system inode *inumber* marked bad

WARNING: msgcnt *x*: vxfs: mesg 017: vx_oltmount_iget – *mount_point* file system inode *inumber* marked bad

WARNING: msgcnt *x*: vxfs: mesg 017: vx_overlay_bmap – *mount_point* file system inode *inumber* marked bad

WARNING: msgcnt *x*: vxfs: mesg 017: vx_readnomap – *mount_point* file system inode *inumber* marked bad

WARNING: msgcnt *x*: vxfs: mesg 017: vx_reorg_trunc – *mount_point* file system inode *inumber* marked bad

WARNING: msgcnt *x*: vxfs: mesg 017: vx_stablestore – *mount_point* file system inode *inumber* marked bad

WARNING: msgcnt *x*: vxfs: mesg 017: vx_tranitimes – *mount_point* file system inode *inumber* marked bad

WARNING: msgcnt *x*: vxfs: mesg 017: vx_trunc – *mount_point* file system inode *inumber* marked bad

WARNING: msgcnt *x*: vxfs: mesg 017: vx_write_alloc2 – *mount_point* file system inode *inumber* marked bad

```
WARNING: msgcnt x: vxfs: mesg 017: vx_write_default - mount_point file
system inode inumber marked bad

WARNING: msgcnt x: vxfs: mesg 017: vx_zero_alloc - mount_point file
system inode inumber marked bad
```

▼ Explanation

When inode information is no longer dependable, the kernel marks it bad on disk. The most common reason for marking an inode bad is a disk I/O failure. If there is an I/O failure in the inode list, on a directory block, or an indirect address extent, the integrity of the data in the inode, or the data the kernel tried to write to the inode list, is questionable. In these cases, the disk driver prints an error message and one or more inodes are marked bad.

The kernel also marks an inode bad if it finds a bad extent address, invalid inode fields, or corruption in directory data blocks during a validation check. A validation check failure indicates the file system has been corrupted. This usually occurs because a user or process has written directly to the device or used fsdb to change the file system.

The VX_FULLFSCK flag is set in the super-block so fsck will do a full structural check the next time it is run.

▼ Action

Check the console log for I/O errors. If the problem is a disk failure, replace the disk. If the problem is not related to an I/O failure, find out how the disk became corrupted. If no user or process is writing to the device, report the problem to your customer support organization. In either case, unmount the file system and use fsck to run a full structural check.

Message: 019

```
WARNING: msgcnt x: vxfs: mesg 019: vx_log_add - mount_point file system
log overflow
```

▼ Explanation

Log ID overflow. When the log ID reaches VX_MAXLOGID (approximately one billion by default), a flag is set so the file system resets the log ID at the next opportunity. If the log ID has not been reset, when the log ID reaches VX_DISLOGID (approximately VX_MAXLOGID plus 500 million by default), the file system is disabled. Since a log reset will occur at the next 60 second sync interval, this should never happen.

▼ Action

Unmount the file system and use `fsck` to run a full structural check.

Message: 020

```
WARNING: msgcnt x: vxfs: mesg 020: vx_logerr - mount_point file system
log error errno
```

▼ Explanation

Intent log failed. The kernel will try to set the `VX_FULLFSCK` and `VX_LOGBAD` flags in the super-block to prevent running a log replay. If the super-block can't be updated, the file system is disabled.

▼ Action

Unmount the file system and use `fsck` to run a full structural check. Check the console log for I/O errors. If the disk failed, replace it before remounting the file system.

Message: 021

```
WARNING: msgcnt x: vxfs: mesg 021: vx_fs_init - mount_point file system
validation failure
```

▼ Explanation

When a VERITAS File System is mounted, the structure is read from disk. If the file system is marked clean, the structure is correct and the first block of the intent log is cleared.

If there is any I/O problem or the structure is inconsistent, the kernel sets the `VX_FULLFSCK` flag and the mount fails.

If the error isn't related to an I/O failure, this may have occurred because a user or process has written directly to the device or used `fsdb` to change the file system.

▼ Action

Check the console log for I/O errors. If the problem is a disk failure, replace the disk. If the problem is not related to an I/O failure, find out how the disk became corrupted. If no user or process is writing to the device, report the problem to your customer support organization. In either case, unmount the file system and use `fsck` to run a full structural check.

Message: 022

```
WARNING: msgcnt x: vxfs: mesg 022: vx_mountroot - root file system
remount failed
```

▼ Explanation

The remount of the root file system failed. The system will not be usable if the root file system can't be remounted for read/write access.

When a VERITAS root file system is first mounted, it is mounted for read-only access. After `fsck` is run, the file system is remounted for read/write access. The remount fails if `fsck` completed a resize operation or modified a file that was opened before the `fsck` was run. It also fails if an I/O error occurred during the remount.

Usually, the system halts or reboots automatically.

▼ Action

Reboot the system. The system either remounts the root cleanly or runs a full structural `fsck` and remounts cleanly. If the remount succeeds, no further action is necessary.

Check the console log for I/O errors. If the disk has failed, replace it before the file system is mounted for write access.

If the system won't come up and a full structural `fsck` hasn't been run, reboot the system on a backup root and manually run a full structural `fsck`. If the problem persists after the full structural `fsck` and there are no I/O errors, contact your customer support organization.

Message: 023

```
WARNING: msgcnt x: vxfs: mesg 023: vx_unmountroot - root file system
is busy and can't be unmounted cleanly
```

▼ Explanation

There were active files in the file system and they caused the unmount to fail.

When the system is halted, the root file system is unmounted. This happens occasionally when a process is hung and it can't be killed before unmounting the root.

▼ Action

fsck will run when the system is rebooted. It should clean up the file system. No other action is necessary.

If the problem occurs every time the system is halted, determine the cause and contact your customer support organization.

Message: 024

```
WARNING: msgcnt x: vxfs: mesg 024: vx_cutwait - mount_point file system
current usage table update error
```

▼ Explanation

Update to the current usage table (CUT) failed.

For a Version 2 disk layout, the CUT contains a fileset version number and total number of blocks used by each fileset.

The VX_FULLFSCK flag is set in the super-block. If the super-block can't be written, the file system is disabled.

▼ Action

Unmount the file system and use fsck to run a full structural check.

Message: 025

```
WARNING: msgcnt x: vxfs: mesg 025: vx_wsuper - mount_point file system
superblock update failed
```

▼ Explanation

An I/O error occurred while writing the super-block during a resize operation. The file system is disabled.

▼ Action

Unmount the file system and use fsck to run a full structural check. Check the console log for I/O errors. If the problem is a disk failure, replace the disk before the file system is mounted for write access.

Message: 026

```
WARNING: msgcnt x: vxfs: mesg 026: vx_snap_copyblk - mount_point
primary file system read error
```

▼ Explanation

Snapshot file system error.

When the primary file system is written, copies of the original data must be written to the snapshot file system. If a read error occurs on a primary file system during the copy, any snapshot file system that doesn't already have a copy of the data is out of date and must be disabled.

▼ Action

An error message for the primary file system prints. Resolve the error on the primary file system and rerun any backups or other applications that were using the snapshot that failed when the error occurred.

Message: 027

```
WARNING: msgcnt x: vxfs: mesg 027: vx_snap_bpcopy - mount_point
snapshot file system write error
```

▼ Explanation

A write to the snapshot file system failed.

As the primary file system is updated, copies of the original data are read from the primary file system and written to the snapshot file system. If one of these writes fails, the snapshot file system is disabled.

▼ Action

Check the console log for I/O errors. If the disk has failed, replace it. Resolve the error on the disk and rerun any backups or other applications that were using the snapshot that failed when the error occurred.

≡ A

Message: 028

WARNING: msgcnt *x*: vxfs: mesg 028: vx_snap_alloc - *mount_point* snapshot
file system out of space

▼ Explanation

The snapshot file system ran out of space to store changes.

During a snapshot backup, as the primary file system is modified, the
original data is copied to the snapshot file system. This error can occur if
the snapshot file system is left mounted by mistake, if the snapshot file
system was given too little disk space, or the primary file system had an
unexpected burst of activity. The snapshot file system is disabled.

▼ Action

Make sure the snapshot file system was given the correct amount of space.
If it was, determine the activity level on the primary file system. If the
primary file system was unusually busy, rerun the backup. If the primary
file system is no busier than normal, move the backup to a time when the
primary file system is relatively idle or increase the amount of disk space
allocated to the snapshot file system.

Rerun any backups that failed when the error occurred.

Message: 029, 030

WARNING: msgcnt *x*: vxfs: mesg 029: vx_snap_getbp - *mount_point* snapshot
file system block map write error

WARNING: msgcnt *x*: vxfs: mesg 030: vx_snap_getbp - *mount_point* snapshot
file system block map read error

▼ Explanation

During a snapshot backup, each snapshot file system maintains a block map
on disk. The block map tells the snapshot file system where data from the
primary file system is stored in the snapshot file system. If an I/O operation
to the block map fails, the snapshot file system is disabled.

▼ Action

Check the console log for I/O errors. If the disk has failed, replace it.
Resolve the error on the disk and rerun any backups that failed when the
error occurred.

Message: 031

```
WARNING: msgcnt x: vxfs: mesg 031: vx_disable – mount_point file system
disabled
```

▼ Explanation

File system disabled, preceded by a message that specifies the reason. This usually indicates a serious disk problem.

▼ Action

Unmount the file system and use fsck to run a full structural check. If the problem is a disk failure, replace the disk before the file system is mounted for write access.

Message: 032

```
WARNING: msgcnt x: vxfs: mesg 032: vx_disable – mount_point snapshot
file system disabled
```

▼ Explanation

Snapshot file system disabled, preceded by a message that specifies the reason.

▼ Action

Unmount the snapshot file system, correct the problem specified by the message, and rerun any backups that failed due to the error.

Message: 033

```
WARNING: msgcnt x: vxfs: mesg 033: vx_check_badblock – mount_point file
system had an I/O error, setting VX_FULLFSCK
```

▼ Explanation

When the disk driver encounters an I/O error, it sets a flag in the super-block structure. If the flag is set, the kernel will set the VX_FULLFSCK flag as a precautionary measure. Since no other error has set the VX_FULLFSCK flag, the failure probably occurred on a data block.

▼ Action

Unmount the file system and use fsck to run a full structural check. Check the console log for I/O errors. If the problem is a disk failure, replace the disk before the file system is mounted for write access.

A

Message: 034

```
WARNING: msgcnt x: vxfs: mesg 034: vx_resetlog – mount_point file
system can't reset log
```

▼ Explanation

The kernel encountered an error while resetting the log ID on the file system. This happens only if the super-block update or log write encountered a device failure. The file system is disabled to preserve its integrity.

▼ Action

Unmount the file system and use `fsck` to run a full structural check. Check the console log for I/O errors. If the problem is a disk failure, replace the disk before the file system is mounted for write access.

Message: 035

```
WARNING: msgcnt x: vxfs: mesg 035: vx_inactive – mount_point file
system inactive of locked inode inumber
```

▼ Explanation

`VOP_INACTIVE` was called for an inode while the inode was being used. This should never happen, but if it does, the file system is disabled.

▼ Action

Unmount the file system and use `fsck` to run a full structural check. Report as a bug to your customer support organization.

Message: 036

```
WARNING: msgcnt x: vxfs: mesg 036: vx_lctbad – mount_point file system
link count table lctnumber bad
```

▼ Explanation

Update to the link count table (LCT) failed.

For a Version 2 and above disk layout, the LCT contains the link count for all the structural inodes. The `VX_FULLFSCK` flag is set in the super-block. If the super-block can't be written, the file system is disabled.

▼ Action

Unmount the file system and use `fsck` to run a full structural check.

Message: 037

```
WARNING: msgcnt x: vxfs: mesg 037: vx_metaioerr - file system meta
data error
```

▼ Explanation

A read or a write error occurred while accessing file system metadata. The full `fsck` flag on the file system was set. The message specifies whether the disk I/O that failed was a read or a write.

File system metadata includes inodes, directory blocks, and the file system log. If the error was a write error, it is likely that some data was lost. This message should be accompanied by another file system message describing the particular file system metadata affected, as well as a message from the disk driver containing information about the disk I/O error.

▼ Action

Resolve the condition causing the disk error. If the error was the result of a temporary condition (such as accidentally turning off a disk or a loose cable), correct the condition. Check for loose cables, etc. Unmount the file system and use `fsck` to run a full structural check (possibly with loss of data).

In case of an actual disk error, if it was a read error and the disk driver remaps bad sectors on write, it may be fixed when `fsck` is run since `fsck` is likely to rewrite the sector with the read error. In other cases, you replace or reformat the disk drive and restore the file system from backups. Consult the documentation specific to your system for information on how to recover from disk errors. The disk driver should have printed a message that may provide more information.

Message: 038

```
WARNING: msgcnt x: vxfs: mesg 038: vx_dataioerr - file system file
data error
```

▼ Explanation

A read or a write error occurred while accessing file data. The message specifies whether the disk I/O that failed was a read or a write. File data includes data currently in files and free blocks. If the message is printed because of a read or write error to a file, another message that includes the inode number of the file will print. The message may be printed as the result of a read or write error to a free block, since some operations allocate

---

an extent and immediately perform I/O to it. If the I/O fails, the extent is freed and the operation fails. The message is accompanied by a message from the disk driver regarding the disk I/O error.

▼ Action

Resolve the condition causing the disk error. If the error was the result of a temporary condition (such as accidentally turning off a disk or a loose cable), correct the condition. Check for loose cables, etc. If any file data was lost, restore the files from backups. Determine the file names from the inode number (see the `ncheck(1M)` manual page for more information.)

If an actual disk error occurred, make a backup of the file system, replace or reformat the disk drive, and restore the file system from the backup. Consult the documentation specific to your system for information on how to recover from disk errors. The disk driver should have printed a message that may provide more information.

Message: 039

```
WARNING: msgcnt x: vxfs: mesg 039: vx_writesuper - file system super-
block write error
```

▼ Explanation

An attempt to write the file system super block failed due to a disk I/O error. If the file system was being mounted at the time, the mount will fail. If the file system was mounted at the time and the full `fsck` flag was being set, the file system will probably be disabled and Message 031 will also be printed. If the super-block was being written as a result of a `sync` operation, no other action is taken.

▼ Action

Resolve the condition causing the disk error. If the error was the result of a temporary condition (such as accidentally turning off a disk or a loose cable), correct the condition. Check for loose cables, etc. Unmount the file system and use `fsck` to run a full structural check.

If an actual disk error occurred, make a backup of the file system, replace or reformat the disk drive, and restore the file system from backups. Consult the documentation specific to your system for information on how to recover from disk errors. The disk driver should have printed a message that may provide more information.

Message: 040

```
WARNING: msgcnt x: vxfs: mesg 040: vx_dqbad – mount_point file system
quota file update error for id id.
```

▼ Explanation

An update to the user quotas file failed for the user ID.

The quotas file keeps track of the total number of blocks and inodes used by each user, and also contains soft and hard limits for each user ID. The VX_FULLFSCK flag is set in the super-block. If the super-block cannot be written, the file system is disabled.

▼ Action

Unmount the file system and use fsck to run a full structural check. Check the console log for I/O errors. If the disk has a hardware failure, it should be repaired before the file system is mounted for write access.

Message: 041

```
WARNING: msgcnt x: vxfs: mesg 041: vx_dqget – mount_point file system
user quota file can't read quota for id id
```

▼ Explanation

A read of the user quotas file failed for the uid.

The quotas file keeps track of the total number of blocks and inodes used by each user, and contains soft and hard limits for each user ID. The VX_FULLFSCK flag is set in the super-block. If the super-block cannot be written, the file system is disabled.

▼ Action

Unmount the file system and use fsck to run a full structural check. Check the console log for I/O errors. If the disk has a hardware failure, it should be repaired before the file system is mounted for write access.

≡ A

Message: 042

```
WARNING: msgcnt x: vxfs: mesg 042: vx_bsdquotaupdate – mount_point file
system user id disk limit reached.
```

▼ Explanation

The hard limit on blocks was reached. Further attempts to allocate blocks for files owned by the user will fail.

▼ Action

Remove some files to free up space.

Message: 043

```
WARNING: msgcnt x: vxfs: mesg 043: vx_bsdquotaupdate – mount_point file
system user id disk quota exceeded too long
```

▼ Explanation

The soft limit on blocks was exceeded continuously for longer than the soft quota time limit. Further attempts to allocate blocks for files will fail.

▼ Action

Remove some files to free up space.

Message: 044

```
WARNING: msgcnt x: vxfs: mesg 044: vx_bsdquotaupdate – mount_point file
system user id disk quota exceeded.
```

▼ Explanation

The soft limit on blocks is exceeded. The soft limit can be exceeded for a certain amount of time before allocations begin to fail. Once the soft quota time limit has expired, further attempts to allocate blocks for files will fail.

▼ Action

Remove some files to free up space.

Message: 045

```
WARNING: msgcnt x: vxfs: mesg 045: vx_bsdiquotaupdate - mount_point
file system user id inode limit reached.
```

▼ Explanation

The hard limit on inodes was exceeded. Further attempts to create files owned by the user will fail.

▼ Action

Remove some files to free inodes.

Message: 046

```
WARNING: msgcnt x: vxfs: mesg 046: vx_bsdiquotaupdate - mount_point
file system user id inode quota exceeded too long
```

▼ Explanation

The soft limit on inodes has been exceeded continuously for longer than the soft quota time limit. Further attempts to create files owned by the user will fail.

▼ Action

Remove some files to free inodes.

Message: 047

```
WARNING: msgcnt x: vxfs: mesg 047: vx_bsdiquotaupdate - warning:
mount_point file system user id inode quota exceeded
```

▼ Explanation

The soft limit on inodes was exceeded. The soft limit can be exceeded for a certain amount of time before attempts to create new files begin to fail. Once the time limit has expired, further attempts to create files owned by the user will fail.

▼ Action

Remove some files to free inodes.

◢ A

Message: 048, 049

```
WARNING: msgcnt x: vxfs: mesg 048: vx_dqread - warning: mount_point
file system external user quota file read failed

WARNING: msgcnt x: vxfs: mesg 049: vx_dqwrite - warning: mount_point
file system external user quota file write failed.
```

▼ Explanation

To maintain reliable usage counts, VxFS maintains the user quotas file as a structural file in the structural fileset. These files are updated as part of the transactions that allocate and free blocks and inodes. For compatibility with the quota administration utilities, VxFS also supports the standard user visible quota files.

When quotas are turned off, synced, or new limits are added, VxFS tries to update the external quota files. When quotas are enabled, VxFS tries to read the quota limits from the external quotas file. If these reads or writes fail, the external quotas file is out of date.

▼ Action

Determine the reason for the failure on the external quotas file and correct it. Recreate the quotas file.

Message: 50

```
WARNING: msgcnt x: vxfs: mesg 050: vx_ldlogwrite - mount_point file
system log write failed
```

▼ Explanation

A write to VERITAS QuickLog log failed. This marks the log bad and sets the full file system check flag in the super block.

▼ Action

No immediate action required. When the file system is unmounted, run a full file system check using fsck before mounting it again.

Message: 51

```
WARNING: msgcnt x: vxfs: mesg 051: vx_ldlog_start - mount_point file
system log start failed
```

▼ Explanation

vx_ldlog_start failed. QuickLog logging is disabled and file system
continues to use its own log.

▼   Action

No corrective action required on the file system. Determine why the log
didn't start and do administrative tasks on QuickLog (see Chapter 9,
"VERITAS QuickLog").

Message: 52

```
WARNING: msgcnt x: vxfs: mesg 052: vx_ldlog_stop - mount_point file
system log stop failed
```

▼ Explanation

QuickLog copies the log back to the file system after stopping logging
activity. If the stop failed, VxFS treats the failure as the log going bad.

▼ Action

No immediate action required. When the file system is unmounted, run a
full file system check using fsck before mounting it again.

Message: 53

```
WARNING: msgcnt x: vxfs: mesg 053: vx_ldlog_suspend - mount_point file
system log suspend failed:
```

▼ Explanation

When the file system is frozen, QuickLog is suspended; it is activated again
on thaw. If this operation fails, the kernel marks the log bad and sets the full
file system check flag in the super block.

▼ Action

No immediate action required. When the file system is unmounted, run a
full file system check using fsck before mounting it again.

Message: 54

```
WARNING: msgcnt x: vxfs: mesg 054: vx_ldlog_resume – mount_point file
system log resume failed:
```

▼ Explanation

When the file system is thawed, QuickLog must be resumed. If this operation fails, the kernel marks the log bad and sets the full file system check flag in the super block.

▼ Action

No immediate action required. When the file system is unmounted, run a full file system check using fsck before mounting it again.

Message: 056

```
WARNING: msgcnt x: vxfs: mesg 056: vx_mapbad – mount_point file system
extent allocation unit state bitmap number number marked bad
```

▼ Explanation

If there is an I/O failure while writing a bitmap, the map is marked bad. The kernel considers the maps to be invalid, so does not do any more resource allocation from maps. This situation can cause the file system to report "out of space" or "out of inode" error messages even though df may report an adequate amount of free space.

This error may also occur due to bitmap inconsistencies. If a bitmap fails a consistency check, or blocks are freed that are already free in the bitmap, the file system has been corrupted. This may have occurred because a user or process wrote directly to the device or used fsdb to change the file system.

The VX_FULLFSCK flag is set. If the VX_FULLFSCK flag can't be set, the file system is disabled.

▼ Action

Check the console log for I/O errors. If the problem is a disk failure, replace the disk. If the problem is not related to an I/O failure, find out how the disk became corrupted. If no user or process was writing to the device, report the problem to your customer support organization. Unmount the file system and use fsck to run a full structural check.

Message: 057

```
WARNING: msgcnt x: vxfs: mesg 057: vx_esum_bad - mount_point file
system extent allocation unit summary number number marked bad
```

▼ Explanation

An I/O error occurred reading or writing an extent allocation unit summary.

The VX_FULLFSCK flag is set. If the VX_FULLFSCK flag can't be set, the file system is disabled.

▼ Action

Check the console log for I/O errors. If the problem is a disk failure, replace the disk. If the problem is not related to an I/O failure, find out how the disk became corrupted. If no user or process was writing to the device, report the problem to your customer support organization. Unmount the file system and use fsck to run a full structural check.

Message: 058

```
WARNING: msgcnt x: vxfs: mesg 058: vx_isum_bad - mount_point file
system inode allocation unit summary number number marked bad
```

▼ Explanation

An I/O error occurred reading or writing an inode allocation unit summary.

The VX_FULLFSCK flag is set. If the VX_FULLFSCK flag can't be set, the file system is disabled.

▼ Action

Check the console log for I/O errors. If the problem is a disk failure, replace the disk. If the problem is not related to an I/O failure, find out how the disk became corrupted. If no user or process was writing to the device, report the problem to your customer support organization. Unmount the file system and use fsck to run a full structural check.

≡ A

Message: 059

```
WARNING: msgcnt x: vxfs: mesg 059: vx_snap_getbitbp - mount_point
snapshot file system bitmap write error
```

▼ Explanation

An I/O error occurred while writing to the snapshot file system bitmap. There is no problem with the snapped file system, but the snapshot file system is disabled.

▼ Action

Check the console log for I/O errors. If the problem is a disk failure, replace the disk. If the problem is not related to an I/O failure, find out how the disk became corrupted. If no user or process was writing to the device, report the problem to your customer support organization. Restart the snapshot on an error free disk partition. Rerun any backups that failed when the error occurred.

Message: 060

```
WARNING: msgcnt x: vxfs: mesg 060: vx_snap_getbitbp - mount_point
snapshot file system bitmap read error
```

▼ Explanation

An I/O error occurred while reading the snapshot file system bitmap. There is no problem with snapped file system, but the snapshot file system is disabled.

▼ Action

Check the console log for I/O errors. If the problem is a disk failure, replace the disk. If the problem is not related to an I/O failure, find out how the disk became corrupted. If no user or process was writing to the device, report the problem to your customer support organization. Restart the snapshot on an error free disk partition. Rerun any backups that failed when the error occurred.

Message: 061

```
WARNING: msgcnt x: vxfs: mesg 061: vx_resize - mount_point file system
remount failed
```

▼ Explanation

During a file system resize, the remount to the new size failed. The VX_FULLFSCK flag is set and the file system is disabled.

▼ Action

Unmount the file system and use fsck to run a full structural check. After the check, the file system shows the new size.

Message: 062

```
NOTICE: msgcnt x: vxfs: mesg 062: vx_attr_creatop - invalid
disposition returned by attribute driver
```

▼ Explanation

A registered extended attribute intervention routine returned an invalid return code to the VxFS driver during extended attribute inheritance.

▼ Action

Determine which vendor supplied the registered extended attribute intervention routine and contact their customer support organization.

Message: 063

```
WARNING: msgcnt x: vxfs: mesg 063: vx_fset_markbad - mount_point file
system mount_point fileset (index number) marked bad
```

▼ Explanation

An error occurred while reading or writing a fileset structure. VX_FULLFSCK flag is set. If the VX_FULLFSCK flag can't be set, the file system is disabled.

▼ Action

Unmount the file system and use fsck to run a full structural check.

Message: 064

```
WARNING: msgcnt x: vxfs: mesg 064: vx_ivalidate - mount_point file
system inode number version number exceeds fileset's
```

▼ Explanation

During inode validation, a discrepancy was found between the inode
version number and the fileset version number. The inode may be marked
bad, or the fileset version number may be changed, depending on the ratio
of the mismatched version numbers.

VX_FULLFSCK flag is set. If the VX_FULLFSCK flag can't be set, the file
system is disabled.

▼ Action

Check the console log for I/O errors. If the problem is a disk failure, replace
the disk. If the problem is not related to an I/O failure, find out how the
disk became corrupted. If no user or process is writing to the device, report
the problem to your customer support organization. In either case,
unmount the file system and use fsck to run a full structural check.

Message: 066

```
NOTICE: msgcnt x: vxfs: mesg 066: DMAPI mount event - buffer
```

▼ Explanation

An HSM (Hierarchical Storage Management) agent responded to a DMAPI
mount event and returned a message in *buffer*.

▼ Action

Consult the HSM product documentation for the appropriate response to
the message.

Message: 067

```
WARNING: msgcnt x: vxfs: mesg 067: mount of device_path requires HSM
agent
```

▼ Explanation

The file system mount failed because the file system was marked as being under the management of an HSM agent, and no HSM agent was found during the mount.

▼ Action

Restart the HSM agent and try to mount the file system again.

Message: 068

```
WARNING: msgcnt x: vxfs: mesg 068: ncsize parameter is greater than
80% of the vxfs_ninode parameter; increasing the value of
vxfs:vxfs_ninode
```

▼ Explanation

The value auto-tuned for the `vxfs_ninode` parameter is less than 125% of the `ncsize` parameter. This message occurs only if one of the system tuneable parameters—ncsize, vxfs_ninode, maxusers, or max_nprocs—is set manually in the file `/etc/system`.

▼ Action

To prevent this message from occurring, set `vxfs_ninode` to at least 125% of the value of `ncsize`. The best way to do this is to adjust `ncsize` down, rather than adjusting `vxfs_ninode` up. See Chapter 5, "Performance and Tuning," for more information.

Message: 069

```
WARNING: msgcnt x: vxfs: mesg 069: memory usage specified by the
vxfs:vxfs_ninode and vxfs:vx_bc_bufhwm parameters exceeds available
memory; the system may hang under heavy load
```

▼ Explanation

The value of the system tuneable parameters—vxfs_ninode and vx_bc_bufhwm—add up to a value that is more than 66% of the kernel virtual address space or more than 50% of the physical system memory. VxFS inodes require approximately one kilobyte each, so both values can be treated as if they are in units of one kilobyte.

▼ Action

To avoid a system hang, reduce the value of one or both parameters to less than 50% of physical memory or to 66% of kernel virtual memory. See Chapter 5, "Performance and Tuning," for more information.

Message: 070

```
WARNING: msgcnt x: vxfs: mesg 070: checkpoint checkpoint_name removed
from file system mount_point
```

▼ Explanation

The file system ran out of space while updating a checkpoint. The checkpoint was removed to allow the operation to complete.

▼ Action

Increase the size of the file system. If the file system size cannot be increased, remove files to create sufficient space for new checkpoints. Monitor capacity of the file system closely to ensure it does not run out of space. See the *fsadm_vxfs*(1M) man page more information.

Message: 071

```
NOTICE: msgcnt x: vxfs: mesg 071: cleared data I/O error flag in
mount_point file system
```

▼ Explanation

The user data I/O error flag was reset when the file system was mounted. This message indicates e that a read or write error occurred (see Message 038) while the file system was previously mounted.

▼ Action

Informational only, no action required.

Glossary

access control list (ACL)
A list of users or groups who have access privileges to a specified file. A file may have its own ACL or may share an ACL with other files. ACLs allow detailed access permissions for multiple users and groups.

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.

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

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

current usage table
A table containing fileset information, such as the number of blocks currently used by the fileset. Not used in the Version 4 disk layout.

data blocks

Blocks that contain 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

Reorganizing data on disk to keep file data blocks physically adjacent so as 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.

extent

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

extent attributes

The extent allocation policies associated with a file.

external quotas file

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

file system block

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

fileset

A collection of files within a file system.

fixed extent size
> An extent attribute associated with overriding the default allocation policy of the file system.

GB
> Gigabytes.

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*.

I/O clustering
> The grouping of multiple I/O operations to achieve better performance.

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 which also contains metadata associated with that file.

inode allocation unit
> A group of consecutive blocks that contain 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 logging scheme that records pending changes to the file system structure. These changes are recorded in a circular *intent log* file.

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

K
> Kilobytes.

large file
> A file larger than 2 gigabytes. VxFS supports files up to two terabytes in size.

large file system

A file system more than 2 gigabytes in size. VxFS supports file systems up to a terabyte in size.

latency

For file systems, this typically refers to the amount of time it takes a given file system operation to return to the user.

MB

Megabytes.

preallocation

The preallocation of space for a file so that disk blocks will physically be part of a file before they are needed. Enabling an application to preallocate space for a file guarantees that a specified amount of space will be available for that file, even if the file system is otherwise out of space.

primary fileset

A fileset that contains the files that are visible and accessible to users.

qio

The Quick I/O program module.

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 which 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 associated with preallocating space for a file.

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**.

structural fileset

A special fileset that stores the structural elements of the file system in the form of structural files. These files define the structure of the file system and are visible only when using utilities such as the file system debugger.

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.

transaction

An update to the file system structure.

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 derived from the 4.2 Berkeley Fast File System.

I/O

I/O that bypasses the kernel cache to increase I/O performance. This is similar to direct I/O, except when a file is extended; for direct I/O, the inode is written to disk synchronously, for unbuffered I/O, the inode update is delayed. See **buffered I/O** and **direct I/O**.

volume

A virtual disk which represents an addressable range of disk blocks used by applications such as file systems or databases.

`vxfs`

The name of the VERITAS File System type.

VxVM

The VERITAS Volume Manager.

Index

current usage table file, 36
CUT, 45
cylinder groups, 9

D

data blocks, 29, 35
data copy, 90
data integrity
    absolute, 13
data synchronous I/O, 74, 92
data transfer
    direct, 90
default
    allocation policy, 55
    block sizes, 5, 22
    intent log size, 71
defragmentation, 9
    extent, 80
    scheduling, 80
delaylog mount option, 72
device file, 49
direct data transfer, 90
direct I/O, 90
directory
    reorganization, 81
disabled file system
    snapshot, 65
    transactions, 130
discovered direct I/O, 91
disk hardware failure
    recovery from, 8
disk layout
    Version 1, 23
    Version 2, 29
    Version 4, 49
disk space allocation, 5, 22
disk structure
    snapshot, 62
dynamic inode allocation, 30, 40

E

enabling Quick I/O, 114
enhanced data integrity modes, 12
ENOENT, 134
ENOTDIR, 134
expansion, 9 to 10
    file system, 79, 81
extended inode operations map, 27, 44
extension
    Quick I/O, 108
extent, 5, 53
    reorganization, 81
extent allocation, 5
    aligned, 54
    control, 53
    fixed size, 54
extent allocation unit state file, 50
extent allocation unit summary file, 50
extent attributes, 53
extent information, 93
extent size
    fixed, 96
    indirect, 6
extents, 22
external quotas file, 100

F

fast file system recovery, 8
file
    device, 49
    extent allocation unit state, 50
    extent allocation unit summary, 50
    fileset header, 49
    free extent map, 50
    inode allocation unit, 49
    inode list, 49
    label, 49
    log, 50
    object location table, 49
    quotas, 50

ufs, 5
inode table, 78
    internal, 78
inodes, 28, 40
    block based, 5
    dynamic allocation, 30, 40
    lost+found, 27
    root, 27
intent log, 23, 24, 30, 33
    default, 71
    default size, 24, 33
    wrapping, 27, 44
intent logging, 24, 33
internal inode table, 78
    sizes, 78
internal quotas file, 100
ioctl interface, 53

K

kernel asynchronous I/O, 106
kernel tuneables, 78

L

label file, 49
large files, 17
    creating file systems with, 17
    mounting file systems with, 17
largefiles mount option, 17
LCT, 45
link count table, 45
link count table file, 36
load balancing, 125
log failure, 130
log file, 50
log files, 95
log mount option, 72

M

maps
    extended inode operations, 27
    extended node operations, 44
    free extent, 28, 35
    free inode, 27, 44
maximum I/O size, 79
mincache mount option, 72, 74
mkfs, 18, 22
modes
    enhanced data integrity, 12
monitoring fragmentation, 80
mount, 12, 18
mount options
    log, 13
mount options, 71 to 76
    blkclear, 72, 73
    choosing, 71 to 76
    combining, 77
    convosync, 72, 75
    delaylog, 13, 72
    extended, 12
    largefiles, 17
    log, 72
    mincache, 72, 74
    nodatainlog, 72, 73
    nolog, 72, 73
    tmplog, 72, 73
mounting file systems
    option combinations, 77
    with large files, 17
    with QuickLog, 124
msgcnt field, 131
multiple block operations, 5
mv_vxfs, 58

N

naming convention
    Quick I/O, 107
NFS, 11

S