Veritas Storage Foundation Tuning Guide

AIX, Linux, and Solaris

5.1 Service Pack 1

Symantec

Veritas Storage Foundation Tuning Guide

The software described in this book is furnished under a license agreement and may be used only in accordance with the terms of the agreement.

Product version: 5.1 SP1

Document version: 5.1SP1.0

Legal Notice

Symantec Corporation
350 Ellis Street
Mountain View, CA 94043

http://www.symantec.com

Technical Support

Symantec Technical Support maintains support centers globally. Technical Support's primary role is to respond to specific queries about product features and functionality. The Technical Support group also creates content for our online Knowledge Base. The Technical Support group works collaboratively with the other functional areas within Symantec to answer your questions in a timely fashion. For example, the Technical Support group works with Product Engineering and Symantec Security Response to provide alerting services and virus definition updates.

Symantec's support offerings include the following:

- A range of support options that give you the flexibility to select the right amount of service for any size organization
- Telephone and/or Web-based support that provides rapid response and up-to-the-minute information
- Upgrade assurance that delivers software upgrades
- Global support purchased on a regional business hours or 24 hours a day, 7 days a week basis
- Premium service offerings that include Account Management Services

For information about Symantec's support offerings, you can visit our Web site at the following URL:

www.symantec.com/business/support/index.jsp

All support services will be delivered in accordance with your support agreement and the then-current enterprise technical support policy.

Contacting Technical Support

Customers with a current support agreement may access Technical Support information at the following URL:

www.symantec.com/business/support/contact_techsupp_static.jsp

Before contacting Technical Support, make sure you have satisfied the system requirements that are listed in your product documentation. Also, you should be at the computer on which the problem occurred, in case it is necessary to replicate the problem.

When you contact Technical Support, please have the following information available:

- Product release level

- Hardware information
- Available memory, disk space, and NIC information
- Operating system
- Version and patch level
- Network topology
- Router, gateway, and IP address information
- Problem description:
  - Error messages and log files
  - Troubleshooting that was performed before contacting Symantec
  - Recent software configuration changes and network changes

Licensing and registration

If your Symantec product requires registration or a license key, access our technical support Web page at the following URL:

www.symantec.com/business/support/

Customer service

Customer service information is available at the following URL:

www.symantec.com/business/support/

Customer Service is available to assist with non-technical questions, such as the following types of issues:

- Questions regarding product licensing or serialization
- Product registration updates, such as address or name changes
- General product information (features, language availability, local dealers)
- Latest information about product updates and upgrades
- Information about upgrade assurance and support contracts
- Information about the Symantec Buying Programs
- Advice about Symantec's technical support options
- Nontechnical presales questions
- Issues that are related to CD-ROMs or manuals

Support agreement resources

If you want to contact Symantec regarding an existing support agreement, please contact the support agreement administration team for your region as follows:

Asia-Pacific and Japan	customercare_apac@symantec.com
Europe, Middle-East, and Africa	semea@symantec.com
North America and Latin America	supportsolutions@symantec.com

Documentation

Your feedback on product documentation is important to us. Send suggestions for improvements and reports on errors or omissions. Include the title and document version (located on the second page), and chapter and section titles of the text on which you are reporting. Send feedback to:

docs@symantec.com

About Symantec Connect

Symantec Connect is the peer-to-peer technical community site for Symantec's enterprise customers. Participants can connect and share information with other product users, including creating forum posts, articles, videos, downloads, blogs and suggesting ideas, as well as interact with Symantec product teams and Technical Support. Content is rated by the community, and members receive reward points for their contributions.

http://www.symantec.com/connect/storage-management

Contents

Chapter 1

Introduction

This chapter includes the following topics:

About tuning Veritas Storage Foundation

Veritas Storage Foundation (SF) is widely used in a range of environments where performance plays a critical role. SF has a number of tunable parameters and configuration options that are meant to enable customization of the stack for the particular environment and workload in which SF is used. This guide helps administrators understand how some of these options affect performance, and provides guidelines for tuning the options.

Chapter 2

Tuning for transaction-processing workloads

This chapter includes the following topics:

About tuning transaction-processing workloads

The Veritas Storage Foundation (SF) stack is frequently used in online transaction-processing (OLTP) environments as the storage stack on top of which mission-critical databases run. This chapter discusses some of the features of the SF stack that are geared toward an OLTP workload, best-practice guidelines for setting up the SF stack for OLTP, and tuning the stack for this workload.

Online transaction-processing workload description

An online transaction-processing (OLTP) workload is a workload in which a number of users connected to a database server concurrently generate queries and updates to the database. You can find examples of this kind of workload in many places, such as online travel reservation systems and online banking. An OLTP workload is characterized by a large number of concurrent transactions, each often accessing only a small amount of data. Performance in OLTP environments is measured primarily by throughput, which is the number of transactions completed per second, with higher throughput being better, and response time, which is the average time taken to complete transactions, with a lower response time being better.

The main component of the OLTP stack is the database, which might have a file system and volume manager below it to manage the storage. The file system, when present, provides space in the form of files for the database to store its tables and other structures. The file system layer is not strictly necessary; databases can usually run directly on raw volumes or disks. Some of the functionality provided by file systems, such as space management, cache management, and recovery management, are also implemented by databases, optimized for database usage. Hence, when running on top of a file system, the database server might use the underlying file system functionality in a very limited manner. Most of the complexity and most of the tuning required in an OLTP stack is typically at the database layer. Some functionality provided by file systems might be redundant or might even degrade performance in an OLTP environment. The following are notable examples of redundant file system functionality:

- File caching
  This is normally an important function provided by file systems that improves performance for many applications, but it is often redundant in an OLTP environment because the database maintains its own cache. Therefore, caching in the file system might add overhead without providing a performance benefit.
- File-level locking
  This is normally required in a file system to ensure data consistency when there are concurrent accesses to a file. However, databases maintain their own fine-grained locks to ensure consistency, and the concurrent accesses that they issue do not need to be serialized. In an OLTP workload, the database typically generates many non-overlapping concurrent reads and writes to the same underlying file; the normal file system locking typically serializes these requests and degrades performance.

The main reason for having a file system in the OLTP stack is that the file system greatly simplifies the administrator's task of space and overall management of the database compared to running the database on top of raw disks or volumes.

Ideally, this convenience should not come at the cost of performance and the file system should provide performance close to that of raw disks and volumes. Veritas File System (VxFS) has features specially designed to meet this goal, including direct I/O to avoid redundant caching in the file system, concurrent I/O to avoid performance degradation due to locking in the file system, and contiguous allocation of large files.

I/O requests generated by the database server can generally be divided into the following categories:

- Write requests to the recovery log. Any changes to the database data or metadata are first recorded by the database server in the recovery log, which is also called the redo log. The recovery log is read during recovery to restore the database to a consistent state, but is not read during normal operation. The write requests to the recovery log are sequential and are typically synchronous. That is, the database server waits for each request to complete before issuing a new one.
- Reads to the database tables. Database blocks are read from database tables into the database memory cache when they are needed by executing transactions. Since an OLTP environment typically has many concurrent transactions, each usually accessing a small amount of data, the read access pattern seen by the file system and volume manager is characterized by a high degree of concurrency, random accesses, and mostly small requests. The service times for read requests usually have a direct impact on the response time of transactions as seen by users.
- Writes to the database tables. Once changes to the database have been recorded in the recovery log, the database server typically holds the modified data in its cache until the server needs to reuse the cache block for other data, at which point the database flushes the modified blocks from its cache to the database tables. These writes are mostly asynchronous; the database server initiates the writes when the server detects pressure on its memory cache, but does not have to wait for the write to complete. The database usually flushes many blocks concurrently, and the write access pattern seen by the file system and volume manager layer is characterized by a high degree of concurrency, random accesses, and mostly small requests. Each of the files used to store the database tables can receive concurrent, non-overlapping writes.

Online transaction-processing workload implications for Veritas Storage Foundation tuning

The database layer takes responsibility for much of the functionality in an online transaction-processing (OLTP) environment, delegating only a few responsibilities to the lower layers of the storage stack. Hence, it is likely that much of the tuning

effort in an OLTP environment centers on the database layer. A number of factors play a role in database performance, including the database schema, query design, and database server configuration, such as the amount of memory given to the database for its cache. A detailed discussion of database performance tuning is beyond the scope of this guide; instead, you should consult the appropriate tuning guide for the database. This document focuses on a relatively small number of guidelines for configuring and tuning the Veritas Storage Foundation (SF) stack to provide good I/O performance for an OLTP workload. The focus here is the case where the file system (VxFS, in this case) is part of the OLTP stack.

The following list includes some of the important implications of the OLTP environment characteristics for SF tuning:

- When VxFS is part of the OLTP stack, meaning that the database not running on raw volumes, the preferred mode of access is usually some form of direct I/O, where caching in the file system is avoided. Many of the VxFS tunable parameters are relevant with buffered I/O, but not with direct I/O. For example, the tunable parameters related to read-ahead and write-behind do not play a role with direct I/O.
- An OLTP environment tends to have a relatively small number of large files; space allocation happens infrequently and in large chunks. Hence, the OLTP workload does not stress the VxFS metadata caches and the intent log; typically, these do not require tuning with an OLTP workload.
- The I/O request stream from the database layer contains a mix of sequential, synchronous writes for the recovery log and small, random reads and writes for the database tables. The volume layout, multi-pathing configuration, and tuning of other features should be based on this access pattern.

Best practices for tuning Veritas Storage Foundation in online transaction-processing environments

This section outlines some best practice guidelines for using the Veritas Storage Foundation (SF) stack in online transaction-processing (OLTP) environments.

Separate volume and file system for redo logs

When provisioning the storage for an online transaction-processing (OLTP) database, you should dedicate a small volume and file system for placing the redo logs, separate from the volumes and file systems used for storing the rest of the database structures. This separation allows accesses to the redo log to be tuned separately from other database accesses and makes it easier to monitor the redo

log performance at the file system, volume, and disk level.The ability to tune the redo log accesses separately is advantageous for the following reasons:

- The redo log is written to with sequential, synchronous writes; this is very different from the access pattern for the rest of the database.
- Having low response times for redo log writes is usually more crucial for good OLTP performance than having low response times for other writes.

The terms redo log volume and data volume are used when the storage layout conforms to the above recommendation. The term redo log volume is used to refer to the volume on which the redo logs are placed. The term data volumes is used for the volumes on which the rest of the database tables are placed.

Data volumes striped across multiple spindles

A data volume should be striped across multiple spindles (physical disks) to improve performance. Striping can be done either in the disk array when the LUNs are created, or in Veritas Volume Manager (VxVM) when the volume is created. In general, striping can improve performance in two ways:

- A large read and write request can be serviced by multiple disks in parallel, making the combined bandwidth of the disks available for a single request.
- Multiple small read and write requests received concurrently can be serviced concurrently and independently by different disks, allowing more requests to be completed per second.

An online transaction-processing (OLTP) data volume typically has the latter access pattern: small, concurrent requests to the database tables. A striped layout for the data volume can reduce response times and increase throughput by allowing I/O requests to be serviced concurrently.

For striped volumes, the stripe unit size parameter determines how much data is stored contiguously on a disk or column in the stripe before moving on to the next disk or column. For an OLTP workload, a moderate stripe unit size works better than a very large stripe unit size since the individual request sizes are typically small. The default stripe unit size for striped VxVM volumes is 64k and is a good choice for striped data volumes of an OLTP database. At the minimum, the stripe unit size should be at least twice the database page size to avoid multiple phsyical disk I/Os per database I/O request.

Mirroring data and redo log volumes

For critical databases, Symantec recommends that you protect the data by using a mirrored layout for data and redo log volumes. Mirroring allows continuous operation and prevents data loss even if there is failure of a single disk. Mirroring

can be done in the disk array when the LUNs are created or in Veritas Volume Manager (VxVM) when the volumes are created. For data volumes, the striping recommendation above can be combined with mirroring by using the stripe-mirror and mirrored stripe layouts of VxVM. This creates volumes that can sustain high throughput and while also providing protection against failures. For redo log volumes, simple mirroring is sufficient.

Mirroring in VxVM is usually used with the Dirty Region Logging (DRL) feature, which reduces the time required to synchronize mirrored volumes after a system crash. The DRL feature usually adds a small performance overhead due to the need for tracking dirty regions of the volume that would need synchronization after a system crash. The sequential DRL feature of VxVM greatly reduces the overhead of dirty region tracking and its use is appropriate when the volume is known to have sequential write access pattern. For redo log volumes, sequential DRL should be used.

In the case of Oracle databases, using the Veritas Oracle Disk Manager (ODM) extension optimizes accesses to mirrored volumes. The Veritas ODM extension works with the Oracle Resilvering feature, where the database identifies the regions of a mirrored volume that need to be resynchronized; the DRL overhead is eliminated in this case.

Parity-based redundancy schemes such as RAID-5 are sometimes used to provide protection against disk failures, similar to protection offered by mirroring. However, the small, random write pattern that is a characteristic of an online transaction-processing (OLTP) workload is generally not well-suited for parity-based schemes from a performance standpoint. Hence, Symantec does not recommend these redundancy schemes for OLTP workloads.

Balanced load on the I/O paths

In a typical online transaction-processing (OLTP) setup, the database server uses storage exported by a disk array over a storage area network (SAN). The I/O path in such a setup involves many elements, such as an HBA, FC switch and port, array controller and port, and physical disks. When planning the storage layout, ensure that the layout allows the I/O load to be spread evenly among the elements on the I/O path. For example, if the disk array has two controllers, each controller should have roughly the same I/O load. In the planning phase, you cannot determine the exact load distribution that will result from a layout; however, simple reasoning about the layout can help avoid bottlenecks that can require disruptive reconfigurations later. The load balancing considerations vary depending on the type of array, such as active-active or active-passive, where mirroring and striping are done, such as in a disk array or the host volume manager, and other factors.

As a simple example, consider a data volume created as a large Veritas Volume Manager (VxVM) volume on a single, large LUN on an active-passive disk array. In this case, since this is an active-passive disk array, the active load will completely be on one controller while the other controller is unused. Instead, if two smaller LUNs were provisioned on the array, and a striped VxVM volume were created over these LUNs, the I/O load could be better distributed across the array controllers by assigning ownership of each LUN to a different controller.

When the database is up and running, a monitoring utility such as `iostat` can be used as a simple check to determine whether the load is more or less evenly distributed among the elements on the I/O path. While a host-based utility such as `iostat` does not give information on each element in the I/O path, load imbalances on the I/O path usually show up as an imbalance in throughput and response times on active devices seen by `iostat`. You can get more detailed information about bottlenecks by using SAN monitoring tools.

Mount options for file systems

When you configure databases other than Oracle to run on Veritas File System (VxFS), Symantec recommends that you mount the VxFS file systems using the `-o cio` mount option, which enables concurrent I/O. The `cio` mount option causes direct I/O to be used, thereby avoiding redundant caching of data in the file system, and avoids lock contention for most concurrent accesses issued by the database server.

In the case of Oracle databases, Symantec recommends that you use the Veritas Oracle Disk Manager (ODM) extension. The steps for setting up Oracle databases with the Veritas ODM extension are outlined in the section on tuning for Oracle databases. The Veritas ODM extension provides the benefits of concurrent I/O and more.

Monitoring performance

Performance data collected from monitoring utilities forms the basis of performance tuning. The statistics gathering framework for tuning the Veritas Storage Foundation (SF) stack in an online transaction-processing (OLTP) environment should, at a minimum, include the output of the following utilities:

- `vxfsstat`: For VxFS information counters.
- `vxstat`: For I/O statistics at the volume layer.
- `vxdmpadm`: For I/O statistics at the multi-pathing layer using the `vxdmpadm iostat` command.
- `iostat`: For I/O statistics at the disk device layer.

- `vmstat`: For memory and CPU utilization and paging statistics.

For routine monitoring, a monitoring interval of about 30 seconds is recommended for these utilities. An interval that is too small can result in performance overhead due to monitoring; an interval that is too large may not give sufficiently detailed information. While it is critically important to have statistics from periods of peak load, having the same statistics also from periods of low load is often useful in identifying bottlenecks. For example, when the average I/O service time is seen to be much higher at peak load compared to low load, this indicates a storage bottleneck; the I/O service time at peak load by itself is usually insufficient to draw this conclusion.

Databases also provide performance tuning features and tools that can help optimize the performance of the storage stack. In some cases, these features can help in identifying ways to reduce the load on the storage subsystem by tuning at the database layer, such as by increasing the database cache size. These tuning features might also help in identifying the specific kind of I/Os, for example redo log writes, which should be the focus of tuning to improve overall performance. The recommended interval for statistics collection for these features or tools might be different from that for the SF or operating system utilities listed above; consult the documentation of the database for guidance.

An OLTP request stream consists of 3 major groups of requests: redo log writes, reads on database tables, and writes on database tables. Low service times for redo log writes is usually the most crucial for good overall performance. Between reads and writes on database tables, overall performance is usually more sensitive to read service times. In some cases, database tuning features can help identify the specific kind of requests that should be targeted for tuning.

General tuning recommendations for an online transaction-processing workload

Some tuning recommendations for an online transaction-processing (OLTP) workload are listed in this section. These generally apply to all databases; exceptions are noted where appropriate.

Tuning Veritas File System for an online transaction-processing workload

Veritas File System (VxFS) tuning differs greatly based on which I/O mode is in effect. For an online transaction-processing (OLTP) workload, administrators typically enable Concurrent I/O or direct I/O explicitly using VxFS mount-time options; in the case of Oracle databases, administrators typically enable the Veritas

ODM extension. However, databases can, based on database-specific configuration options, enable Concurrent I/O or direct I/O for their accesses using platform-specific mechanisms. Hence, it is incorrect to assume that the default buffered I/O mode is in effect in VxFS when the administrator has not explicitly enabled Concurrent I/O, direct I/O, or the Veritas ODM extension.

See "About I/O modes" on page 64.

As a first step to tuning VxFS, you should determine which I/O mode is in effect. The `vxfsstat -v` command displays counters that help in identifying the I/O mode being used:

- When buffered I/O is being used, each read request causes one of the counters `vxi_read_seq` or `vxi_read_rand` to be incremented. The `vxi_read_seq` counter is incremented when VxFS detects the read as a sequential read, and the `vxi_read_rand` counter is incremented for reads classified as random reads. Similarly, each write request causes one of the counters `vxi_write_seq` or `vxi_write_rand` to be incremented. If the output of the `vxfsstat` command shows non-zero values for these counters, the value indicates that buffered I/O is being used.
  One example of when buffered I/O better than direct I/O and concurrent I/O is with read-only database operations, which benefit from file system read-aheads and buffering from multiple clients that issue localized read requests.
- When direct I/O is being used, each read request increments the counter `vxi_read_dio`. Similarly, each write request increments the counter `vxi_write_dio`. If these counters are seen to have non-zero values, this indicates that direct I/O is being used. Concurrent I/O is a form of direct I/O and these same counters are also incremented in the case Concurrent I/O.
- The path for read and write requests when the Veritas ODM extension, for Oracle databases, is being used is slightly different from the default direct I/O path. Even though reads and writes through the Veritas ODM library do not cause the caching of data in the page cache, the reads and writes do not cause the `vxi_read_dio` and `vxi_write_dio` counters to get incremented.
  See "Tuning recommendations for transaction-processing workloads in Oracle databases" on page 25.

Note: The counters reported by the `vxfsstat` command are for the VxFS kernel module, not for each file system. If you have multiple VxFS file systems mounted, these counters will show aggregate activity for all file systems. This should be kept in mind when interpreting the data.

Tuning Veritas Volume Manager for an online transaction-processing workload

For an online transaction-processing (OLTP) workload, the performance impact of dirty region logging (DRL) and instant snapshots on transaction throughput and response times is generally low. These features incur performance overhead mainly for writes. In the case of redo log writes in an OLTP workload, configuring sequential DRL, as recommended in the best practice guidelines, keeps DRL overhead very low. In the case of data volume writes, overhead incurred by these writes for DRL and instant snapshots have a low impact on overall performance because of the asynchronous nature of data volume writes. Still, tuning these features can further reduce their overhead and prove beneficial in some cases.

The following table lists some aspects that can be tuned:

Region size for DRL	For DRL, the choice of region size is a tradeoff between the performance of normal writes during normal operation and the time taken to recover. A larger region size reduces the overhead of DRL, but can also increase the time it takes to synchronize volume mirrors after a crash.
Number of dirty regions allowed in the DRL	Increasing the `voldrl_max_drtregs`, `voldrl_volumemax_drtregs`, and `voldrl_volumemax_drtreg_20` parameters increases the maximum number of dirty regions that may exist at any time. This can increase performance during normal operation, but can increase recovery time.
Number of dirty regions in sequential DRL	For redo log volumes that are mirrored and configured with sequential DRL, increasing the `voldrl_max_seq_dirty` parameter can further reduce the overhead of DRL.
Region size for instant snapshots	Symantec recommends the default region size of 64 KB for an OLTP workload, since writes are mostly small. See "Instant snapshots" on page 98.
Memory for caching region maps	Increasing the `volpagemod_max_memsz` parameter increases the amount of memory used to cache region maps in memory and can improve performance. See "Dirty Region Logging for mirrored volumes" on page 94.

Dynamic multi-pathing tuning

The default settings of the dynamic multi-pathing (DMP) tunable parameters usually work quite well for an online transaction-processing (OLTP) workload. Symantec recommends the `minimumq` I/O load balancing policy, which is the

default, for an OLTP workload for all array types. Other aspects of DMP tuning are often dependent more on the nature of the storage environment, such as the total number of paths and the frequency of errors, than on the workload characteristics.

See "Summary of Dynamic Multi-Pathing tuning" on page 126.

Tuning recommendations for transaction-processing workloads in Oracle databases

Detailed guidance for tuning Oracle is beyond the scope of this tuning guide. See the appropriate performance tuning guide for Oracle for detailed information. This document highlights a few initialization parameters for Oracle that have the most impact on the SF stack; the rest of the section covers setting up and using the Veritas Oracle Disk Manager (ODM) extension.

Oracle initialization parameters

The amount of memory devoted to an Oracle database instance is among the most important tuning decisions for good online transaction-processing (OLTP) performance. The memory allocated to the database is used for a number of internal memory pools. Oracle supports various options for memory management through its initialization parameters. In one of the simplest options, an administrator can choose automated memory management and only specify the total amount of memory to be used by the database instance, and then Oracle assigns the memory appropriately to its different pools.

Note: When Veritas File System (VxFS) is accessed using the Veritas Oracle Disk Manager (ODM) extension, or in the direct I/O or Concurrent I/O mode, caching of data in the file system is disabled. This allows more aggressive use of memory for the database cache and generally leads to better performance. However, if too little memory is configured for the database cache, performance might degrade compared to using the buffered I/O mode to access the file system. While the above forms of non-buffered I/O have the potential for better performance than buffered I/O, properly sizing the database cache is crucial for this to happen.

The nature of I/O to the underlying layers--file system and volume manager--can be configured using the initialization parameters `disk_asynch_io` and `filesystemio_options`, according to the guidance provided by Oracle. Common values for these parameters are `disk_asynch_io=true` and `filesystemio_options=setall`.

Configuring the Veritas Oracle Disk Manager extension

Symantec recommends using the Veritas Oracle Disk Manager (ODM) extension when using Veritas Storage Foundation (SF) with Oracle databases. The Veritas ODM extension improves performance for Oracle databases on the SF stack in a number of ways:

- it supports asynchronous I/O
- it supports direct I/O
- it reduces locking overhead
- it reduces number of system calls
- it optimizes file opening and identification
- it supports contiguous allocation
- it eliminates Dirty Region Logging (DRL) overhead of Veritas Volume Manager (VxVM) mirroring

These benefits and the steps for configuring the Veritas ODM extension are explained in the Veritas Storage Foundation guide for Oracle Databases. Configuring Veritas ODM requires replacing the default ODM library in the Oracle distribution with the Veritas ODM library. The Veritas Storage Foundation guide for Oracle Databases also lists the steps to verify that the Veritas ODM extension is correctly configured.

Cached Oracle Disk Manager

While Veritas Oracle Disk Manager (ODM) by default does not cache file data in the file system layer, the Cached ODM feature allows some caching of file data. With Veritas ODM, the Oracle SGA must be sized large enough, otherwise performance can suffer. Cached ODM can improve performance in cases where the Oracle SGA size is limited by other considerations, such as if there is more than one instance on the same server. To enable Cached ODM, set the tunable parameter `odm_cache_enable=1` by using the `vxtunefs` command after mounting the Veritas File System (VxFS) file system. The Cached ODM feature can be configured in two ways:

- to turn caching on or off on a per-file basis
- to set caching advisories based on file type and I/O type

See the *Veritas Storage Foundation: Storage and Availability Management for Oracle Databases* document.

Summary of tuning recommendations for online transaction-processing workload

The following list summarizes the recommendations for databases other than Oracle:

- Create a dedicated volume and file system for the database recovery log.
- Stripe data volumes so that the bandwidth of multiple physical disks is available to I/O on the data volumes.
- Configure sequential dirty region logging (DRL) for the recovery log volume, if the volume is mirrored in Veritas Volume Manager (VxVM).
- Plan the storage configuration so as to balance load on the elements on the I/O path.
- Mount VxFS using the `mount -o cio` command to enable Concurrent I/O. This avoids caching in the file system and reduces locking overhead.
- Use database configuration parameters to allocate a sufficiently large database buffer cache. Using Concurrent I/O or direct I/O for VxFS allows memory to be allocated more aggressively to the database cache. Good performance with these I/O modes depends on the database cache being tuned appropriately.
- If the DRL or instant snapshot features of VxVM are being used, tune these features to reduce performance overhead.
- Collect performance statistics at peak and low loads. Maintaining low service times for the recovery log writes is the highest priority, followed by maintaining low service times for data volume reads.

Chapter 3

Tuning for NFS file-serving workloads

This chapter includes the following topics:

About tuning NFS file-serving workloads

One important workload class where the Veritas Storage Foundation (SF) stack is deployed is NFS file serving. An NFS file server receives NFS client requests over the network and issues operations on the underlying exported file system. The NFS client caches files in the client kernel. Read and write requests coming in to the NFS server are the result of client cache misses in the case of reads and cache flushes in the case of write. The NFS server process is typically a multi-threaded kernel daemon with each thread serving one client request at a time. Each client request that is received over the network is handled by a server thread that maps the request to operations on the server file system and then performs the operations. The following list describes some additional characteristics of an NFS server workload:

- A large fraction of the requests may not require access to file data; rather, these requests require access to file metadata. Examples of such metadata requests are GETATTR, SETATTR, LOOKUP and ACCESS.
- Read and write requests transfer data to and from clients. Request sizes and the ratio of reads to writes varies depending on the kind of files being served.
- Transfer sizes for individual data requests (reads and writes) are limited by the NFS protocol. The limit is 8k for NFSv2. For NFSv3, the size limit varies, but is often 32k.

- Writes are synchronous at the server in NFSv2. That is, the server is required to flush the written data from file system caches to persistent storage. In NFSv3, individual writes need not be synchronous, but the client typically requests the server to flush writes out to disk very soon after the write. NFS servers do not typically hold a large amount of dirty data in their cache.

Tuning recommendations for NFS file-serving workloads

The NFS workload characteristics have the following implications for tuning the Veritas Storage Foundation (SF) stack:

- Since metadata requests are an important part of the request mix, tuning Veritas File System (VxFS) metadata caches is beneficial in many cases. Prominent among these are the VxFS inode cache and the buffer cache.
- Since read and write requests sizes are limited by the NFS protocol, VxFS tunable parameters such as `discovered_direct_iosz` that take effect with large requests typically are not a factor.
- Since NFS servers typically do not hold a lot of dirty data in their cache, VxFS tunable parameters that control write flushing are not expected to play a significant role.

Tuning NFS server daemon threads

On Solaris, AIX and Linux, the NFS server is a multi-threaded kernel daemon. When a request is received from an NFS client, a thread in the NFS server is assigned the request and issues appropriate operations on the underlying file system. The number of threads that are started in the NFS server can be tuned; the default values are usually too low for demanding environments. For dedicated NFS file servers, Symantec recommends that the number of NFS server threads be set to a high value, such as 128, in environments where a large number of clients are being served.

Tuning the maximum number of NFS server threads on Solaris

On the Solaris 10 operating system, specify the maximum number of threads in the NFS server by setting the value of `NFSD_SERVERS` in the `/etc/default/nfs` file. Restart the NFS service for a change to the `NFSD_SERVERS` parameter to take effect:

```
# svcadm restart svc:/network/nfs/server
```

The actual number of NFS servers started varies based on the demand and can be observed in the output of commands such as `prstat`.

Tuning the number of NFS server threads on Linux

On Linux, specify the number of NFS threads by editing the `/etc/sysconfig/nfs` file.

To specify the number of NFS threads

1. Edit the `/etc/sysconfig/nfs`:

   ```
   # vi /etc/sysconfig/nfs
   ```

2. Change the `RPCNFSDCOUNT=` value to the number of NFS threads that you want.
3. Save the `/etc/sysconfig/nfs`.
4. Restart the NFS server:

   ```
   # /etc/rc.d/init.d/nfs restart
   ```

Tuning the maximum number of NFS server threads on AIX

On AIX 6.1, the number of NFS threads that are started varies dynamically based on load, but is subject to a maximum. The default value of this limit is usually high enough that you do not need to tune explicitly.

The `nfso` command manages NFS tuning parameters. The `nfso -h` *`nfs_max_threads`* command gets information on the maximum number of threads and how AIX handles thread creation. The `nfso` command can also change the maximum number of threads if the default limit is not suitable.

Tuning the main memory caches

NFS file-serving can benefit from the tuning of metadata caches, which is where file and file system metadata are cached, as well as the tuning of the page cache, which is where file data is cached. NFS file-serving typically results in the heavy use of the metadata caches. The operating system page cache is also heavily used during file serving; if the server is dedicated to NFS file-serving, most of the memory that is not used by kernel structures, including VxFS metadata caches, typically fill up with file data cached in the page cache.

There are dependencies between the memory caches that must be kept in mind while tuning them. Since these considerations are somewhat different for each

operating system, steps for tuning main memory caches for a file-serving workload are listed separately for Solaris, Linux and AIX.

Tuning memory caches on Solaris

Use the following utilities to monitor the main memory caches for an NFS file server workload on Solaris:

- `vxfsstat`

  This utility gives information on the Veritas File System (VxFS) buffer cache, inode cache, and DNLC (with the `-bi` option).

- `vmstat`

  This utility gives information on memory usage and page scanning activity.

The following list provides suggestions for tuning the main memory caches for an NFS file server workload on Solaris:

- Increasing the size of the VxFS buffer cache or the VxFS inode cache should be performed only when `vmstat` output indicates that there is enough free memory to accommodate the size increase.
- Tune the VxFS buffer cache based on the `vxfsstat` command's output.

  See "Tuning the Veritas File System buffer cache" on page 51.

  If the current size has not reached the maximum, the buffer cache need not be tuned. If the current size has reached the maximum and if the hit rate or recycle age is low, increase the buffer cache size limit. Changing the maximum size of the buffer cache requires a reboot for the new size to take effect.
- Tune the VxFS inode cache based on `vxfsstat` output.

  See "Tuning the Veritas File System inode cache" on page 54.

  Changing the maximum size of the VxFS inode cache requires a reboot for the new size to take effect.
- Tuning the inode cache so that the recycle age is high should also ensure that the page cache is utilized well. On Solaris 10, for a dedicated NFS file-serving workload, the `vmstat` command often shows large values for free memory. This happens even when most of the memory is being used by the page cache to cache file data, meaning that the memory is not actually free. This occurs because of the way that Solaris 10 manages and reports on page cache usage.

  See "Page cache monitoring and tuning on Solaris" on page 60.

  The free memory reported by the `vmstat` command includes pages that are actually free (the freelist) as well as pages that are on the cachelist, which are pages that contain valid cached file pages. As a result, the actual page cache usage is somewhat difficult to determine. The breakup of memory usage into

freelist pages and cachelist pages can be seen using a debugger, such as by using the following command:

```
# echo "::memstat" | mdb -k
```

Ideally, most of the memory should be in the cachelist, because this ensures that memory is being used to cache file pages rather than not being utilized. Tuning the inode cache appropriately is typically good enough to ensure that the page cache is also well utilized.

- Since much of the memory is likely to be in cachelist pages, and since Solaris easily reclaims this memory when the freelist is depleted, you typically do not see expensive page scanning for a dedicated NFS file server workload. The `sr` field of the `vmstat` command's output should be close to 0.

Tuning main memory caches on Linux

Use the following information when tuning the main memory caches for an NFS file server workload on Linux:

- You need the output from the following monitoring utilities to properly tune the main memory caches:
  - The `vxfsstat` command with the `-bi` option gives information on VxFS buffer cache and inode cache.
  - The `/proc/meminfo` command shows overall memory usage.
  - The `/proc/zoneinfo` command gives memory usage information at the memory zone level.
  - The `vmstat` command gives memory and CPU utilization information.
  - The `sar` command with the `-B` option gives page scanning and reclamation information.
  - The `/proc/slabinfo` gives information on kernel slab caches
  - The `top` command gives CPU utilization of user processes, kernel threads and daemons.
- Tune the VxFS buffer cache based on the `vxfsstat` command's output.
  See "Tuning the Veritas File System buffer cache" on page 51.
  If the current size has not reached the maximum, the buffer cache need not be tuned. If the current size has reached the maximum and if the hit rate or recycle age is low, increase the buffer cache size limit. Changing the maximum size of the buffer cache requires a reboot or module reload for the new size to take effect.

- Check the `vxfsstat` command's output to see if the VxFS inode cache needs to be tuned.
  See "Tuning the Veritas File System inode cache" on page 54.
  Changing the maximum size of the inode cache requires a reboot or module reload for the new size to take effect.
- Check the `/proc/slabinfo` command's output to see if the operating system's dentry cache size matches the VxFS inode cache size. Ideally, the caches should be about the same size. Under memory pressure, Linux can shrink the dentry cache in an attempt to reclaim memory, but Linux provides a tunable parameter, `vfs_cache_pressure`, to control this behavior
  See "Tuning the Linux dentry cache" on page 58.
  For an NFS file server workload, this tunable parameter can be set to a value lower than 100 to favor the reclamation of page cache pages rather than shrinking the dentry cache.
- Check memory usage reported by the `/proc/meminfo` command to see if the page cache is sized well. If the inode cache is too small, most of the memory might remain free and hence be underutilized. Tuning the inode cache properly usually results in a drop in free memory and an increase in memory used for the page cache.
  See "Tuning the Linux dentry cache" on page 58.
  On a well-tuned system, you commonly see free memory levels drop low, which results in page scanning and reclamation activity. You should check that the page reclamation is happening efficiently. This can be seen directly from the output of the `sar -B` command, which gives information on the number of pages scanned, the number of pages freed, and the efficiency of page scanning. You should also monitor the CPU utilization of `kswapd`, the kernel daemon responsible for page scanning, to see that the utilization is within reasonable bounds.

The following example output is of tuning a RHEL5 server under an NFS file-serving workload. The server has 4 dual-core processors and 32GB memory. The number of NFS server threads was set to 256 prior to this exercise, but other tunable parameters are at their default setting. The statistics were collected by starting all the monitoring utilities from a shell script; so the actual command lines are not shown. With the server under heavy load, the output of `cat /proc/meminfo` is show below:

```
MemTotal:     32958952 kB
MemFree:      16827880 kB
Buffers:          2012 kB
Cached:        9786216 kB
SwapCached:          0 kB
```

```
Active:        5799208 kB
Inactive:      4040608 kB
HighTotal:           0 kB
HighFree:            0 kB
LowTotal:     32958952 kB
LowFree:      16827880 kB
SwapTotal:    34996216 kB
SwapFree:     34994740 kB
Dirty:          614656 kB
Writeback:           0 kB
AnonPages:       51448 kB
Mapped:          11732 kB
Slab:          1036272 kB
PageTables:       4420 kB
NFS_Unstable:        0 kB
Bounce:              0 kB
CommitLimit:  51475692 kB
Committed_AS:   329276 kB
VmallocTotal: 34359738367 kB
VmallocUsed:    392620 kB
VmallocChunk: 34359345639 kB
HugePages_Total:     0
HugePages_Free:      0
HugePages_Rsvd:      0
Hugepagesize:     2048 kB
```

The following output is of the `vxfsstat -bi` command in the same interval:

```
13:22:59.336 Sat 25 Sep 2010 01:22:59 PM IST -- delta (30.030 sec sample)

Lookup, DNLC & Directory Cache Statistics
        0 maximum entries in dnlc
   130048 total lookups             0.00% fast lookup
        0 total dnlc lookup         0.00% dnlc hit rate
        0 total enter               0.00  hit per enter
        0 total dircache setup      0.00  calls per setup
   145053 total directory scan      0.00% fast directory scan

inode cache statistics
   818978 inodes current    818980 peak               818977 maximum
   950026 lookups            70.99% hit rate
        0 inodes alloced          0 freed
        0 sec recycle age
      600 sec free age
```

```
buffer cache statistics
  3382872 Kbyte current    6860608 maximum
  1779834 lookups            99.99% hit rate
     3571 sec recycle age [not limited by maximum]
```

The VxFS buffer cache size is well below its maximum size. The buffer cache hit rate and recycle age are seen to be very high. There is no need to tune the buffer cache in this example. The inode cache, on the other hand, is seen to be under heavy pressure: the size of the inode cache has reached the maximum possible, the hit rate is low, and the recycle age is extremely low (0, in fact). The counters for DNLC are 0 because on Linux, VxFS does not maintain its own DNLC.

From the `/proc/meminfo` command's output, you see that almost 16 GB memory is free. This indicates that the inode cache that is too small; the inode cache is itself under heavy pressure and it is not large enough to allow the page cache to grow to utilize all of the free memory. The following `/proc/meminfo` output shows how the situation changes when the VxFS inode cache maximum size is increased drastically to about 2 million:

```
MemTotal:     32958952 kB
MemFree:        155612 kB
Buffers:          1080 kB
Cached:       23022680 kB
SwapCached:          0 kB
Active:        9177028 kB
Inactive:     13899596 kB
HighTotal:           0 kB
HighFree:            0 kB
LowTotal:     32958952 kB
LowFree:        155612 kB
SwapTotal:    34996216 kB
SwapFree:     34994724 kB
Dirty:          622628 kB
Writeback:           0 kB
AnonPages:       51664 kB
Mapped:          11588 kB
Slab:          1994912 kB
PageTables:       4540 kB
NFS_Unstable:        0 kB
Bounce:              0 kB
CommitLimit:  51475692 kB
Committed_AS:   327848 kB
VmallocTotal: 34359738367 kB
```

```
VmallocUsed:    429556 kB
VmallocChunk: 34359305131 kB
HugePages_Total:     0
HugePages_Free:      0
HugePages_Rsvd:      0
Hugepagesize:     2048 kB
```

The following output is from the `vxfsstat -bi` command after increasing the inode cache size:

```
01:37:16.156 Sat 04 Sep 2010 01:37:16 AM IST -- delta (30.007 sec sample)

Lookup, DNLC & Directory Cache Statistics
        0 maximum entries in dnlc
    77769 total lookups             0.00% fast lookup
        0 total dnlc lookup         0.00% dnlc hit rate
        0 total enter               0.00  hit per enter
        0 total dircache setup      0.00  calls per setup
    93616 total directory scan      0.00% fast directory scan

inode cache statistics
  2047144 inodes current   2047444 peak              2047442 maximum
   942090 lookups            97.40% hit rate
        0 inodes alloced         50 freed
      487 sec recycle age [not limited by maximum]
      600 sec free age

buffer cache statistics
  3363560 Kbyte current    6860608 maximum
  1133917 lookups            99.99% hit rate
     3586 sec recycle age [not limited by maximum]
```

As can be seen, this single change has altered the memory utilization dramatically. As can be seen from the `vxfsstat` command's output, the inode cache is still at its maximum size, but the recycle age is very high now. The hit rate has improved to about 97%. The side effect of the inode cache tuning change can be seen in the output of `/proc/meminfo`: MemFree has dropped to 150 MB and there is an increase in the amount of cached data. The larger inode cache has allowed the page cache to grow. With free memory so low, you must check for page reclamation activity.

The following `sar -B` command's output shows page scanning and reclamation:

```
pgpgin/s pgpgout/s   fault/s  majflt/s  pgfree/s pgscank/s pgscand/s pgsteal/s    %vmeff
50845.20  96110.23    253.30      0.40  61814.40  22174.93      0.00  22137.43     99.83
```

```
47935.87  96316.37    208.97       0.03  59121.33  20937.60       0.00  20911.40      99.87
48298.27  94385.97    217.40       0.00  58679.20  20908.80       0.00  20882.10      99.87
```

The above output shows that pages are now being scanned and reclaimed. However, the efficiency of the scanning is very high (%vmeff, 99%+), so the operating system is not having difficulty reclaiming pages.

Finally, check that the Linux dentry cache and the VxFS inode cache are at about the same size. The number of entries in the Linux dentry cache can be seen in the output of the `/proc/slabinfo` command:

```
slabinfo - version: 2.1
# name            <active_objs> <num_objs> <objsize> <objperslab> <pagesperslab> :
dentry_cache       1574508       1576206    216       18           1 :

tunables <limit> <batchcount> <sharedfactor> :
tunables  120     60           8 :

slabdata <active_slabs> <num_slabs> <sharedavail>
slabdata  87567          87567       0
```

The output shows that the number of objects in the Linux dentry cache is much smaller than the number of inodes in the VxFS inode cache that were seen in the `vxfsstat` command's output. This usually is due to the kernel shrinking the dentry cache under memory pressure. To ensure that the two caches are sized to work well with each other, the `vfs_cache_pressure` tunable parameter was set to a value of 30 to reduce the shrinking of the dentry cache. The following output is of the `/proc/slabinfo` command after this change:

```
slabinfo - version: 2.1
# name            <active_objs> <num_objs> <objsize> <objperslab> <pagesperslab> :
dentry_cache       1933662       1971396    216       18           1 :

tunables <limit> <batchcount> <sharedfactor> :
tunables  120     60           8 :

slabdata <active_slabs> <num_slabs> <sharedavail>
slabdata  109522         109522      360
```

The number of objects in the dentry cache (1.97 million) is now close to the number of inodes in the inode cache (2.04 million).

Tuning main memory caches on AIX

Use the following utilities to monitor the main memory caches for an NFS file server workload on AIX:

- `vxfsstat`

  This utility gives information on the Veritas File System (VxFS) buffer cache, inode cache, and DNLC (with the `-bi` option).

- `vmstat`

  This utility gives information on memory usage and page scanning activity.

The following list provides suggestions for tuning the main memory caches for an NFS file server workload on AIX:

- Tune the VxFS buffer cache based on the `vxfsstat` command's output.

  See "Tuning the Veritas File System buffer cache" on page 51.

  If the current size has not reached the maximum, the buffer cache need not be tuned. If the current size has reached the maximum and if the hit rate or recycle age is low, increase the buffer cache size limit. Changing the maximum size of the buffer cache requires a reboot for the new size to take effect.

- Tune the VxFS inode cache based on `vxfsstat` output.

  See "Tuning the Veritas File System inode cache" on page 54.

  If the amount of free memory reported by the `vmstat` command is very low, increasing the VxFS inode cache size limit is not recommended since it can trigger page scanning and reclamation, which can degrade performance. If the VxFS inode cache is under pressure, but `vmstat` output shows a large amount of free memory, you can increase the size limit of the VxFS inode cache. If the inode cache size limit is increased, monitor the system to ensure that the tuning change does not cause heavy page scanning and reclamation activity, as seen in the pages scanned and pages reclaimed fields of `vmstat` output.

- The `numperm` and `numclient` fields, which are displayed by the `vmstat -v` command, show the memory used to cache file data pages. On a well-tuned NFS file server, a significant amount of memory should be devoted to caching file data pages; however, free memory should not be so low that the system is pushed into a state where page scanning and reclamation overhead becomes significant.

Tuning for mirrored Veritas Volume Manager volumes and snapshots

An NFS server that has been tuned well, meaning that the number of NFS server threads has been tuned to a high value, and is under high load can generate numerous concurrent requests for the Veritas Volume Manager (VxVM) layer

because of the large number of NFS server threads concurrently issuing I/O requests. Since these I/Os are for requests from different NFS clients, there may not be a lot of locality among these concurrent requests. As a result, the NFS server workload may see higher overhead compared to other workloads in some of the features provided by VxVM, such as dirty region logging (DRL) and instant snapshots. The reference section on VxVM tuning has more information on tuning these features to improve performance. This section gives a brief summary of the tuning considerations as they apply to an NFS workload.

See "About tuning Veritas Volume Manager" on page 89.

When both DRL and instant snapshot functionality are needed for a volume in VxVM, the same version 20 DCO volume is used to keep track of the DRL dirty regions and region maps for instant snapshots. However, from a performance standpoint, the region size considerations for these two features are different. For an NFS workload, a large DRL region size is essential for reducing the negative performance impact of DRL on NFS server performance. However, very large region sizes are not optimal for instant snapshots. For an NFS workload, Symantec recommends that you do not use these two features of VxVM together. One alternative is to use mirroring capabilities of disk arrays and use the instant snapshot feature of VxVM. Another option is to use VxVM mirroring and DRL, but use other Storage Foundation snapshot features, such as Storage Checkpoints, that can be used to achieve the same goals as VxVM instant snapshots.

When the DRL feature is used without instant snapshots for an NFS workload, you should have a large region size for good performance of the NFS server. With DRL in a version 20 DCO, the region size should be chosen larger than the default of 64 KB. For traditional DRL, the default regions size is typically large enough that explicit tuning might not be required.

When you need VxVM instant snapshots, you should use full-sized instant snapshots over space-optimized instant snapshots (SO snapshots) for an NFS workload where performance is important. The default region size is generally appropriate for an NFS workload.

You should tune other parameters, such as `voldrl_max_drtregs`, `voldrl_volumemax_drtregs`, `voldrl_volumemax_drtregs_20`, and `volpagemod_max_memsz`, according to the guidelines in the reference section on VxVM tuning.

See "About tuning Veritas Volume Manager" on page 89.

Chapter 4

Tuning reference for Veritas File System

This chapter includes the following topics:

- About Veritas File System space allocation
- Online resizing and defragmentation

About tuning Veritas File System

Veritas File System (VxFS) is an enterprise-class file system that has provided robust, reliable, high-performance storage to demanding applications for over 2 decades. A number of characteristics combine to provide robustness and good performance in VxFS. The following list provides a brief overview of VxFS:

- When a VxFS file system is created on disk storage using the `mkfs` command, VxFS creates a number of structures on the device to keep track of free space, files created, and blocks allocated to files. The structures created vary with the layout version; VxFS releases 5.0 through 5.1SP1 use disk layout Version 7 by default, but also support some of the older versions. The intent log, one of the key structures enabling reliable operation in VxFS, is created on disk at the time of file system creation.
  See "Tuning the intent log" on page 45.
- The VxFS kernel module implements the file system functionality and must be loaded before a VxFS file system residing on disk can be mounted and accessed. When the VxFS module is loaded, the module creates a number of structures in kernel memory. The most important among these structures are memory caches used by VxFS. In most cases, VxFS maintains its own caches rather than relying on the file system caches provided by the operating system. VxFS caches are designed to scale well and provide good performance on large multiprocessor systems.
  See "About the Veritas File System caches" on page 49.
- When a file system is mounted and accessed, changes to file data and file system metadata are usually made in the memory caches first and flushed to disk later. As a result, the on-disk state can be slightly out-of-date with the latest state of the file system. When a file system is cleanly unmounted, all changes are written to disk before the file system shuts down, bringing the on-disk state up-to-date. However, during a crash, not all changes make it to disk. VxFS uses intent logging to ensure that the key file system structures can be recovered quickly in the event of a crash.
- VxFS supports traditional UNIX applications that use the buffered read/write behavior, and also provides interfaces meant for efficient I/O for databases.
  See "Tuning read-ahead" on page 66.
  See "Read flush-behind in Veritas File System" on page 72.
  See "Tuning Veritas File System buffered writes" on page 74.
  See "About I/O modes" on page 64.

- VxFS uses extent-based allocation to be enable efficient space allocation to files. It supports online resizing and defragmentation, which allow space allocation to remain efficient over time.
  See "About Veritas File System space allocation" on page 85.

Monitoring Veritas File System operation

Veritas File System (VxFS) maintains a number of counters internally as it encounters events of interest. These are called VxFS info counters. Examples are counters that maintain the number of lookups and hits in the VxFS metadata caches. The `vxfsstat` utility reports the values of these counters. Examining the output of `vxfsstat` can give valuable insight into what tuning needs to be performed. Use these counters, along with operating system monitoring utilities, such as `vmstat` and `iostat`, to guide the process of tuning VxFS.

Creating file systems

You create a VxFS file system by using the `mkfs` command.

For information about the options that are available at the time that you create the file system, see the `mkfs_vxfs`(1M) manual page.

These options specify layout parameters for the file system being created. The following options are particularly relevant from a performance standpoint:

- File system block size
  The block size of a file system is the smallest unit in which Veritas File System (VxFS) allocates space for files in that file system. By default, VxFS chooses the block size based on the size of the file system being created, but the block size can be specified explicitly at the time that you create the file system. The block size cannot be changed after the file system has been created. The impact of block size on performance is typically small in VxFS, since VxFS is an extent-based file system that allocates space in contiguous regions called extents, which consist of multiple blocks. The default block size picked by VxFS based on the size of the file system being created is appropriate for most systems.
- Intent log size
  The intent log of a file system is an on-disk structure where VxFS records structural changes to the file system before applying the changes to the actual file system structures. By default, VxFS chooses the intent log size based on the size of the file system that is being created. The intent log size can be explicitly specified when creating the file system; it can also be changed after

the file system has been created. Some workloads result in heavier use of the intent log and can get better performance with a larger intent log size.
See "Tuning the intent log" on page 45.

Mounting file systems

Veritas File System (VxFS) allows you to specify a number of options with the `mount` command that can be used to control some aspects of operation for the file system being mounted.

For information about the mount options, see the `mount_vxfs`(1M) manual page and the *Veritas File System Administrator's Guide*.

The following options are commonly used to manage performance:

- Enabling direct I/O
  The default mode for handling I/O in VxFS is called buffered I/O, where file data is cached in the file system data cache. However, for some applications, performance improves with direct I/O, which is a mode in which caching in the file system is avoided and data is moved directly between storage devices and application buffers. VxFS mount options can be used to specify that direct I/O, rather than the default buffered I/O, should be used for the file system being mounted. You can use the `mincache` and `convosync` mount options to achieve this; in the most common case, the mount option `mincache=direct,convosync=direct` is used to enable direct I/O for a file system.
  See "Direct I/O" on page 81.
- Enabling Concurrent I/O
  The Concurrent I/O mode is similar to direct I/O except that it uses a relaxed form of locking that improves performance for some database workloads. The default locking in VxFS provides concurrency control for read and write requests that complies with the POSIX standard, but limits performance for some database workloads where the database frequently issues concurrent, non-overlapping writes to the file system; these writes would be serialized by the default locking mechanism, although they can be allowed to proceed in parallel without violating consistency. The Concurrent I/O mode, where the locking performed by VxFS allows concurrent writes to the same file, can be enabled with the `-o cio` mount option.
- Intent log mode
  The `log`, `delaylog` and `tmplog` mount options control how VxFS writes metadata changes to the intent log. The default mode is `delaylog` and is appropriate in most cases.

See "Tuning the intent log" on page 45.

- Data in intent log
  The `datainlog` and `nodatainlog` mount options affect how small, synchronous writes are handled. With the `datainlog` mount option, which is the default, VxFS performs an optimization where it writes data from small, synchronous writes to the intent log; this optimization is disabled with the `nodatainlog` mount option.
  See "Synchronous buffered writes" on page 75.

Tuning the intent log

Veritas File System (VxFS) uses intent logging or journaling to enable fast recovery and better file system integrity. A high-level file system operation such as creating a file can require updates to multiple VxFS on-disk metadata structures. A system crash can result in an inconsistent state for the file system, where some of the file system metadata structures have been updated to reflect the changes made by high level operations and other structures have not. On file systems that do not use intent logging, recovering from a crash involves a complete scan and verification of the metadata structures, known as a file system consistency check or fsck, which is a time consuming process. In an intent logging file system, any changes that are required to on-disk metadata structures as a result of system calls are first recorded in an intent log, which is an on-disk region written sequentially. The actual metadata structures are updated only after the changes have been safely recorded in the intent log. While recovering from a crash, the file system can examine the intent log to determine which metadata structures were being updated, complete any partial updates, and recover to a consistent state. This is much faster than scanning and verifying the entire file system.

In VxFS, a full file system consistency check (full fsck) is needed in some cases, most commonly when sectors on underlying disk devices might have been damaged. In most cases, recovering from the intent log is sufficient to ensure integrity of the file system.

Deciding which mode of intent log operation to use

Veritas File System (VxFS) provides a mount time option to select the mode for intent log operation. You choose the mode by using the `mount -o` *`logmode`* command, where *logmode* is either `log`, `delaylog`, or `tmplog`. The default mode is `delaylog`.

With intent logging, the writes to the intent log simply record changes that will be applied to other on-disk structures; VxFS issues separate writes to update the actual on-disk structures. For the intent logging mechanism to work, writes to

the intent log should happen before the corresponding writes to the other file system structures. However, it is not strictly necessary for the intent log writes to be written out synchronously at the end of the system call that is causing the changes. The different modes for intent log updates differ in when entries are written out to the intent log, and hence provide different guarantees regarding the persistence of effects of system calls.

In VxFS, when the file system is mounted in `log` mode, the intent log is updated synchronously at the end of the system calls, ensuring that the effects of system calls are persistent upon their completion. When the file system is mounted in `delaylog` mode, the intent log is written mostly asynchronously and the persistence of the effects of the system call is not guaranteed upon completion of the call. In both modes, VxFS is able to recover to a consistent state after a crash by replaying the intent log.

The `delaylog` mode is generally able to provide better performance since `delaylog` usually does not require a synchronous write to the intent log as part of system call. This behavior reduces the completion time of system calls as seen by applications. VxFS worker threads flush the changes to the intent log asynchronously within a few seconds, typically within 3 seconds. This behavior still provides better persistence guarantees than traditional UNIX file systems where persistence of the effects of system calls can be delayed by as much as 30 seconds. For most applications, the default `delaylog` mode is the appropriate mode.

The `tmplog` mode is only recommended for temporary file systems. The `tmplog` mode delays log flushing, as with the `delaylog` mode. In addition, some other changes allow this mode to give better performance, but with weaker persistence guarantees.

Intent log size

Veritas File System (VxFS) uses the on-disk intent log as a circular buffer. VxFS writes the intent log sequentially until it reaches the end of the disk region marked for the log. At this point, VxFS wraps around and starts writing from the beginning of the disk region marked for the log, overwriting the old intent log entries. Before old entries in the intent log can be overwritten, the changes to file system structures corresponding to those entries must have been flushed out to disk. New writes to the intent log must pause if this flushing is not complete. In most cases, by the time the log wraps around, VxFS will have flushed out changes corresponding to the older log entries. However, if the intent log is small and the workload is metadata-intensive, the log can wrap around frequently and in some cases force operations to pause.

The intent log size can be specified during file system creation using the `-o logsize=n` option with the `mkfs` command, where *n* is the number of file system blocks to be used for the intent log. The default log size depends on the size of the file system being created. For small file systems, VxFS allocates less space for the intent log; for file systems greater than 512 GB, VxFS allocates 256 MB for the intent log. 256 MB is the maximum size that VxFS allocates for the intent log. If necessary, intent log size can be changed after the file system has been created.

For information on the minimum, maximum and default intent log size, see the `mkfs_vxfs`(1M).

From a performance standpoint, a larger intent log is better because the log wraps around less frequently. The space required for the intent log might be a concern when the file system is small; however, the maximum size of the intent log is 256 MB, which is a small amount of storage space by current standards. A larger intent log might result in a slightly longer recovery time because a larger number of log entries could need to be replayed during recovery. A larger log size also increases the memory requirement during recovery, which is roughly twice the intent log size; again, given the large amount of memory on current systems, the memory requirement is usually not a concern, even with the maximum intent log size.

For a system in use, the output of the `vxfsstat -v` command can help in determining if the intent log space is affecting performance. The following counters can indicate if performance is affected:

- vxi_tranleft_asyncflush
- vxi_tranleft_syncflush
- vxi_tranleft_delay
- vxi_tran_retry
- vxi_bdwrite_loglow
- vxi_brelse_loglow

VxFS increments these counters whenever low intent log space causes VxFS to take some action. If these counters are all zero, then intent log space is never an issue in VxFS operation for your workload. This is the ideal case. If these counters have a high value and if the intent log space is not at its maximum, then increase the intent log space.

The VxFS info counters can help identify when the intent log size allocated during file system creation is not sufficiently large for your workload. In this case, you can resize the intent log using the `fsadm -o logsize=size` command, where *size* is the new desired size of the intent log.

See the `fsadm_vxfs`(1M) manual page.

About the datainlog and nodatainlog mount options

The primary purpose of the Veritas File System (VxFS) intent log is to record changes to file system metadata; file data is not normally written to the intent log. However, VxFS does use an optimization where for some small, synchronous writes, file data is written to the intent log synchronously and the actual file blocks on disk are updated asynchronously.

See "Synchronous buffered writes" on page 75.

The `mount -o datainlog` option enables this optimization and the `mount -o nodatainlog` option disables it. The default is to use the `datainlog` optimization, meaning that unless the `nodatainlog` mount option is explicitly used, VxFS will write data from some small, synchronous writes to the intent log. For most workloads, the default `datainlog` mode performs better than or as well as `nodatainlog`. However, as is the case with most optimizations, this is not always true; for some workloads `nodatainlog` can perform better.

The default `datainlog` option can increase the amount of data written to the intent log; this should be another consideration in deciding intent log size and placement.

Placing the intent log on a separate device

In Veritas File System (VxFS), you can place the intent log on a separate device using the multi-volume file system feature. The multi-volume file system feature is available when VxFS is used with Veritas Volume Manager (VxVM).

For information about multi-volume file systems, see the *Veritas File System Administrator's Guide*.

You gain the following benefits for placing the intent log on a separate device:

- The intent log has a sequential access pattern, and placing it on a dedicated disk avoids the unnecessary disk head movement that would result if it shared the device with other file system data and metadata.
- When using VxVM mirroring, sequential dirty region logging (DRL) can be used to reduce the overhead of DRL.

When the storage for the file system comes from a disk array, placing the intent log on a separate device might not yield much performance benefit, because the array write cache would normally hide disk write latencies from the file system. However, in cases where the disk array cache is under heavy pressure, there might be some performance gain.

Intent log placement can be specified using the `fsadm -o logvol=`*`vol`* command, where *vol* is the volume in the volume set where the intent log is to be placed.

See the fsadm_vxfs(1M) manual page.

About the Veritas File System caches

Veritas File System (VxFS), as with most file systems, uses main memory caches to improve performance by reducing accesses to storage devices. Some of the important caches used by VxFS are as follows:

page cache	On Solaris, Linux, and AIX, VxFS file data pages are cached in the operating system page cache that is integrated with the virtual memory management system of the operating system. The page cache is typically shared by the different file systems running on the machine. For example, on a Solaris server, the page cache is used to store file data pages for all VxFS file systems as well as all UFS file systems mounted on the server.
buffer cache and inode cache	These are metadata caches that VxFS maintains on its own; they are distinct from similar metadata caches maintained by the operating system or other file system types on the same system. These VxFS metadata caches are shared by all VxFS file systems that are mounted on the system. For example, all VxFS file systems mounted on a Solaris server share the same VxFS buffer cache; however, this buffer cache is not shared with any UFS file systems that are mounted on the server.
Directory Name Lookup Cache	On Solaris and AIX, VxFS maintains a separate Directory Name Lookup Cache (DNLC) of its own to speed up filename lookups in directories; the VxFS DNLC is shared by all VxFS file systems mounted on the system and is not shared with other file system types. On Linux, VxFS shares the operating system's directory cache--the dentry cache--with other file systems.

Since all of these caches must divide up a finite amount of main memory among themselves, tuning one cache can have implications for the others.

About the Veritas File System metadata caches

Veritas File System (VxFS) maintains the following private metadata caches: the buffer cache, the inode cache, and, on Solaris and AIX, the Directory Name Lookup Cache (DNLC). These caches share the following characteristics:

- An important parameter governing the operation of the cache is its maximum size. The maximum size of the cache can be specified using a tunable parameter. If the size is not specified using the tunable parameter, VxFS decides the maximum size of the cache based on the total amount of main memory on the

system. The maximum size is decided at VxFS module initialization, typically at boot time.

- Actual entries in the cache are created and freed based on demand. Under heavy usage, VxFS grows the cache until the cache reaches the maximum size that was decided at initialization. Once that limit is reached, more memory is not allocated for the cache; instead, old buffers are reused after evicting existing entities. This dynamic allocation of memory for cache entries means that the actual memory used by the cache might be much less than the maximum. The actual and maximum sizes for the cache can be seen using the `vxfsstat` utility. Observing `vxfsstat` output at regular intervals over a period of time while the system is in use gives a clear picture of how much the workload is stressing the cache.
- In many cases, no explicit tuning of the cache may be required. Since VxFS, by default, chooses the size limit for its metadata caches based on the amount of memory on the system, high-end systems automatically get larger caches. However, since the appropriate size for the cache also depends on the nature of the workload, in some cases tuning is required.
- Most commonly, the tuning that is required involves increasing the maximum size of the cache when the cache is consistently at its maximum size and under pressure. Under these conditions, after the maximum size is increased, the actual size of the cache also usually increases. The tradeoff here is that the cache will be more effective, but there will be a reduction in the free memory available on the system that may affect other aspects of system operation.
- In some cases, usually on systems with a memory bottleneck, an administrator might want to reduce the memory allocated to a VxFS metadata cache. In this case, the administrator can set the maximum size of the cache to a value less than the actual size of the cache as seen from `vxfsstat` output. This reduces the effectiveness of the cache and should only be done if you expect that the memory freed will benefit overall operation of the system by allowing another cache to grow or by making the memory available to applications.

The `vxfsstat` utility with the `-bi` option prints relevant statistics for all VxFS metadata caches. Typically, this is combined with the `-t` option of `vxfsstat` to get statistics at regular intervals while the system is in peak use. For example:

```
# vxfsstat -bi -t 30 /mount1
```

The first set of values in this case is the absolute sample representing stats since the last reboot; the remaining samples give relevant information that can be used to analyze the cache usage during the period of observation. Usually, it is more convenient to use the `-v` option of vxfsstat instead of the `-bi` option; this gives

all of the event counters that VxFS maintains, not just stats for the metadata caches.

See the `vxfsstat`(1M) manual page.

Tuning the Veritas File System buffer cache

The buffer cache is the lowest-level metadata cache in Veritas File System (VxFS). Disk blocks containing file system metadata are read and written through the buffer cache; other higher-level VxFS metadata caches, such as the inode cache, also read from and write to the buffer cache rather than issuing disk I/O directly. Tuning the buffer cache can be very effective in improving performance for some workloads by reducing metadata-related disk I/O.

For tuning the buffer cache, it is important to know the maximum size and the current size of the buffer cache. The following fields in the `vxfsstat -v` output give this information:

vxi_bcache_maxkbyte	Gives the maximum size of the buffer cache. If you have explicitly specified the value for the maximum size of the buffer cache, then this counter should be the close to that value in most cases. There are some sanity checks performed by VxFS based on which it might adjust or ignore the value you specified.
vxi_bcache_curkbyte	Gives the current size of the buffer cache. This can be less than or equal to `vxi_bcache_maxkbyte`.

Setting the maximum buffer cache size on Solaris

The tunable parameter `vx_bc_bufhwm` is used to specify the maximum size of the buffer cache on Solaris. The value of this parameter is interpreted as the desired maximum size of the buffer cache in kilobytes. For example, to set the maximum size of the buffer cache to 2 GB (2*1024*1024 kilobytes), add the following line in the `/etc/system` file:

```
set vxfs:vx_bc_bufhwm= 2097152
```

This value takes effect after the next system reboot.

Setting the maximum buffer cache size on Linux

The VxFS module parameter `vxfs_mbuf` is used to specify the maximum size of the buffer cache on Linux. The value of this parameter is interpreted as the desired

maximum size of the buffer cache in bytes. For example, to set the maximum size of the buffer cache to 2 GB, add the following line in the `/etc/modprobe.conf` file:

```
options vxfs vxfs_mbuf= 2147483648
```

This value takes effect after the next system reboot or after the Veritas File System (VxFS) module is unloaded and reloaded using the `modprobe` command.

Setting the maximum buffer cache size on AIX

The tunable parameter `vx_bc_bufhwm` is used to specify the maximum size of the buffer cache on AIX. The value of this parameter is interpreted as the desired maximum size of the buffer cache in kilobytes. For example, to set the maximum size of the buffer cache to 2 GB (2*1024*1024 kilobytes), add the following line in the `/etc/vx/vxfssystem` file:

```
vx_bc_bufhwm 2097152
```

This value takes effect after the next system reboot or when the VxFS kernel extension is reloaded.

When to tune the buffer cache

The output of the `vxfsstat` command can give insight into whether the buffer cache tuning for your setup and workload is likely to be beneficial. Look for the following counters, which are output when you use the `vxfsstat` command with the `-v` option:

vxi_bcache_curkbyte	The current size of the buffer cache in kilobytes.
vxi_bcache_maxkbyte	The maximum size in kilobytes to which the buffer cache can grow.
vxi_bcache_recycleage	The average number of seconds that a buffer in the buffer cache remains free before it is reused to store a different disk block.
vxi_bc_lookups	The number of lookups to the buffer cache.
vxi_bc_hits	The number of lookups to the buffer cache that found the block in the cache. vxi_bc_hits*100/vxi_bc_lookups gives the buffer cache hit rate as a percentage.

The current and maximum buffer cache size, hit rate, and recycle age are also printed in more readable form with the `-b` option of `vxfsstat`.

Ideally, the buffer cache hit rate should be in the high 90s and the recycle age should be high. You will commonly see a 99% hit rate or more in the case of the buffer cache, and a recycle age of more than 500. The recycle age is a good indicator of the pressure on the buffer cache. If the value is less than 100, it indicates that buffers in the buffer cache are being reclaimed too frequently.

The most frequent tuning that is required for the buffer cache involves increasing its maximum size. If the current size of the buffer cache is equal to the maximum size, meaning that the buffer cache has grown to its maximum size, and recycle age is very small, then you should consider increasing the maximum size of the buffer cache. In this scenario, when you increase the maximum size of the buffer cache, you are likely to see the size of the buffer cache (`vxi_bcache_curkbyte`), the buffer cache hit rate, and recycle age go up. This reduces the disk I/O pertaining to metadata. The performance benefit from this varies; on systems where storage is the bottleneck, there can be a good gain in performance. However, increasing the buffer cache size reduces the free memory available in the system. Depending on the nature of the workload and pressure on the memory, this can cause problems in the form of additional paging or swapping to disk.

The need to tune down the maximum size of the buffer cache arises less often. In cases where the size of the buffer cache is consistently less than the maximum, there is no real incentive to reduce the maximum size of the buffer cache to match actual usage. However, there might be scenarios--usually on systems with a memory bottleneck--where memory needs to be carefully apportioned among the various main memory caches, and shrinking the buffer cache to favor some other cache may be beneficial. For example, consider a scenario where the inode cache and the buffer cache are both at their maximum size and the observed statistics suggest that increasing inode cache size is more crucial. An administrator can decide to tune the VxFS system by reducing the maximum buffer cache size and increasing the maximum inode cache size.

Additional considerations for tuning the buffer cache

When the buffer cache size is increased, the primary effect is reducing the disk I/O pertaining to metadata. On systems where storage bandwidth is the bottleneck, this can result in improved performance. Increasing the buffer cache size also reduces the free memory available in the system. In particular, increasing the buffer cache size reduces the effective memory available to the operating system page cache. In systems with a memory bottleneck, this can cause problems in the form of additional paging or swapping to disk.

Tuning the Veritas File System inode cache

The Veritas File System (VxFS) inode cache is a metadata cache dedicated to storing file inodes. The inode for a file in VxFS, and in most UNIX file systems, stores a collection of metadata for the file, including access rights, location of the file blocks on disk, and access and modification times. In VxFS, before any operation can be performed on a file, the inode for the file must be brought into the inode cache if it is not already present there. Inodes are stored persistently on disk, with each inode commonly occupying 256 bytes on disk. The inode size can be chosen to be 512 bytes at the time that you created the file system. The main fields of an inode on disk are access rights, access and modification timestamps, and an index of the block layout for the file. When an inode is read into the inode cache, a number of fields are added to the ones that are present in the disk inode. These include pointers to cached pages for the file and various locks to synchronize accesses. An in-core inode--an inode residing in the inode cache--is significantly larger than an on-disk inode; the actual size varies with the operating system, but is typically around a kilobyte.

The inode cache conceptually sits on top of the buffer cache. When an inode for a file is needed and is not present in the inode cache, a block containing that inode is first read from disk into the buffer cache, if the inode is not already present in the buffer cache. This brings all disk inodes in that disk block into the buffer cache. Inode metadata is typically read in 8 KB chunks. The inode that is needed is copied into the inode cache with the in-core fields added. Thus, an inode can simultaneously exist in in-core form in the inode cache and in on-disk form in the buffer cache. The other combinations are also possible: an inode can exist only in the inode cache, only in the buffer cache as a disk inode, or in neither. As a result, properly tuning the buffer cache can also benefit inode cache operation by eliminating disk reads for inodes.

Tuning the inode cache is important even when the buffer cache has been properly tuned. When an inode needs to be brought into the inode cache, another inode that is currently not in use might need to be evicted. The eviction can require some processing to be performed, such as invalidating any file data pages for the inode that are cached in the page cache. Also, some information that VxFS has about the file, such as sequential read/write patterns on the file, are lost in the eviction and are not available when the file is accessed again. Tuning the inode cache properly can help reduce inefficiencies resulting from these factors.

As in the case of the buffer cache, the size of the inode cache varies because VxFS grows and shrinks the cache based on demand. For tuning the inode cache, it is important to know the actual size and maximum size of the inode cache. There are 3 counters in the `vxfsstat -v` output related to inode cache size that are relevant when tuning the inode cache:

vxi_icache_maxino	The maximum size of the inode cache. If you have explicitly specified the inode cache size, this counter should be close to that value in most cases. VxFS performs some sanity checks that it uses to determine whether to adjust or ignore the value that you specified.
vxi_icache_curino	The current size of the inode cache.
vxi_icache_peakino	The maximum size seen for the inode cache since the last reboot.

On Solaris, Linux and AIX, the tunable parameter `vxfs_ninode` is used to specify the maximum size of the VxFS inode cache. The value of this parameter is interpreted as the maximum number of inodes allowed in the inode cache. The procedure for setting this tunable parameter is different on each operating system.

Setting the maximum inode cache size on Solaris

To set the maximum number of inodes in the inode cache to 1 million on Solaris, add the following line in the file `/etc/system`:

```
set vxfs:vxfs_ninode=1000000
```

This value takes effect after the next system reboot. Each inode occupies more than one kilobyte in memory, so the actual memory required by an inode cache of the above size is more than one gigabyte.

Setting the maximum inode cache size on Linux

To set the maximum number of inodes in the inode cache to 1 million on Linux, add the following line in the file `/etc/modprobe.conf`:

```
options vxfs vxfs_ninode=1000000
```

This value takes effect after the next system reboot or after the VxFS module is reloaded. Each inode occupies more than one kilobyte in memory, so the actual memory required by an inode cache of the above size is more than one gigabyte.

Setting the maximum inode cache size on AIX

To set the maximum number of inodes in the inode cache to 1 million on Linux, add the following line in the file `/etc/vx/vxfssystem`:

```
vxfs_ninode 1000000
```

This value takes effect after the next system reboot or after the VxFS kernel extension is reloaded. Each inode occupies more than one kilobyte in memory, so the actual memory required by an inode cache of the above size is more than one gigabyte.

When to tune the inode cache size

The output of `vxfsstat` can give insight into whether inode cache tuning is likely to be beneficial for your setup and workload. Look for the following counters in the `vxfsstat -v` command's output:

vxi_icache_curino	The current size of the inode cache (the number of inodes in the cache).
vxi_icache_maxino	The maximum size of the inode cache.
vxi_icache_recycleage	The average number of seconds that an inode remains unused before it is reused to store a different inode.
vxi_iget	The number of lookups in the inode cache.
vxi_iget_found	The number of times a lookup found the inode in the inode cache. vxi_iget_found*100/vxi_iget gives the inode cache hit rate as a percentage.

The current inode cache size, maximum inode cache size, hit rate, and recycle age are also printed in more readable form with the `-i` option of the `vxfsstat` command.

Ideally, the inode cache hit rate should be in the high 90s and the recycle age value should be high. If the recycle age is low (less than 100), it indicates that there is pressure on the inode cache, meaning that inodes are getting evicted very frequently.

The most frequent tuning that is required for the inode cache involves increasing its maximum size. If the size of the inode cache is equal to the maximum size and the recycle age is small, you should consider increasing the maximum size of the inode cache. When the maximum inode cache size is increased in this scenario, the actual size of the inode cache is also likely to increase (possibly to the new maximum); the hit rate and recycle age are also likely to go up. In effect, inodes will be evicted less often from the inode cache and disk inodes will need to be read in less often from the buffer cache. The processing overhead that this eliminates can result in significant gains in performance, especially in cases where the CPU usage is high. Improved hit rate in the inode cache will also reduce pressure on the buffer cache. The downside to increasing the inode cache size is that there

will be a reduction in the free memory on the system, which can cause performance problems in some cases.

The need to tune down the maximum size of the inode cache arises less often. The inode cache is an important metadata cache for VxFS operation and in most cases it makes sense to tune it well. But in some cases, usually on systems with a memory bottleneck, the size of the inode cache may need to be reduced to make memory available to another cache or to applications. Reducing the inode cache size will involve setting the value of the tunable parameter vxfs_ninode to a value less than the observed size of the inode cache.

Additional considerations for tuning the inode cache

When a file has its inode cached in the inode cache, it can have data cached in the page cache. Hence, when the number of inodes in the inode cache increases, there might be an increase in the total amount of file data that is cached. As a result, an increase in the inode cache size may result in a big drop in free memory on the system--this is partly because the inode cache is using more memory, and partly because more file data is being cached. In many cases, the drop in free memory due to an expanding page cache is not a problem, because memory that was free is being put to use and the OS is usually able to efficiently reclaim memory when it needs to. However, depending on how the file system, other kernel modules and applications are using memory, there might be situations where the OS is not able to efficiently reclaim memory for new requests, thus leading to performance problems. The overall memory usage on the system, as given by a utility such as `vmstat`, and page reclamation activity should be taken into account before and after tuning the inode cache.

Tuning the Directory Name Lookup Cache

The Directory Name Lookup Cache (DNLC) is a metadata cache used to speed up filename lookups. A lookup operation takes a directory inode and a filename and tries to get the inode for the file, if the filename exists. The lookup operation is one of the most frequent operations in a file system, and a dedicated cache can give significant performance overall benefits. The DNLC caches both filenames that have been found in a directory (positive entries) and filenames that have been confirmed not to exist in the directory (negative entries). DNLC operation is generally closely tied to inode cache operation because the positive entries in the DNLC link to the inode cache.

Tuning the Directory Name Lookup Cache on Solaris and AIX

On Solaris and AIX, Veritas File System (VxFS) maintains its own Directory Name Lookup Cache (DNLC) to improve the performance of lookup operations. All VxFS file systems mounted on a system share the same DNLC, which is distinct from the DNLC maintained by other file system types on the system, such as the native file systems. Since the DNLC and inode cache operation are closely linked, VxFS uses a single tunable parameter to control the maximum size of both the DNLC and inode cache. The `vxfs_ninode` tunable parameter that determines the maximum number of entries in the inode cache also determines the maximum size of the DNLC. Typically, sizing the VxFS inode cache according to the guidelines in the previous section is sufficient to ensure that the DNLC is also sized appropriately.

Tuning the Linux dentry cache

On Linux, VxFS does not maintain its private DNLC; instead, VxFS shares the Linux dentry cache with other file systems. The dentry cache is registered as a slab cache in Linux and is resized dynamically by the Linux kernel based on demand. When there is demand for more entries, the dentry cache grows. When there is memory pressure, Linux automatically shrinks the dentry cache. Since the dentry cache is controlled by the Linux kernel and the VxFS inode cache size is tuned separately, care needs to be taken to ensure that the two are working well together. In many cases, the relative sizes of the two caches can reveal the need for tuning one or the other. The size of the VxFS inode cache can be seen from the output of `vxfsstat`. The size of the Linux dentry cache can be seen in the `proc` pseudo file system by using the following command:

```
# cat /proc/slabinfo
```

One potential problem is that the dentry cache is trying to grow, but the VxFS inode cache has already reached its size limit. In this case, the VxFS inode cache could be limiting the dentry cache. If the active file systems are VxFS, the size of the dentry cache in this case is likely to be about the same as the VxFS inode cache. In this situation, the output of `vxfsstat` usually reveals that the VxFS inode cache is under pressure. Tuning the VxFS inode cache should mitigate the problem.

See "Tuning the Veritas File System inode cache" on page 54.

Another potential problem is that the Linux kernel has shrunk the dentry cache under memory pressure and as a result the VxFS inode cache is not being used optimally. If the active file systems are VxFS, the size of the dentry cache in this case is likely to be less than the VxFS inode cache size. The Linux kernel tunable parameter `vfs_cache_pressure` can potentially be used in this case to instruct the kernel to target the page cache rather than the dentry cache to reclaim

memory; this will ease the shrinking of the dentry cache. See the Linux kernel documentation for information on when it is appropriate to change the `vfs_cache_pressure` parameter.

Page cache monitoring and tuning

On Solaris, Linux, and AIX, physical memory is divided into fixed size units called pages and allocated to application processes or the kernel as needed. Allocated pages fall into one of the following categories:

kernel memory	Allocated to the various kernel modules for storing their data. This includes memory allocated to VxFS metadata caches like inode and buffer cache.
anonymous pages	Allocated to executing processes to store their runtime stack and heap.
page cache	Allocated to store data pages of files in the different file systems that are mounted.
free memory	Pages that are unallocated and readily available to the kernel or user processes as needed.

The pages in the page cache are brought in and flushed out by interactions between the kernel Virtual Memory Management subsystem and the file system modules, such as Veritas File System (VxFS). The page cache can expand to use available memory when file data is being heavily accessed, and can shrink under memory pressure when the operating system is low on free memory. This approach allows flexible and dynamic caching of file data.

When the memory that is readily available for new allocations drops below some threshold, these operating systems initiate page scanning and reclamation. The reclamation typically targets anonymous pages and page cache for pages that have not been recently used and frees them. Page scanning and reclamation can sometimes result in performance degradation, and operating systems typically provide a number of tunable parameters to control which pages are targeted and when.

Effective tuning of VxFS also requires some awareness of the interactions between the memory management of the operating system and the VxFS tuning changes.

See the tuning guidelines of the operating system for more detailed information about operating system page cache management. The tuning recommendations might vary depending on the release of the operating system.

Page cache monitoring and tuning on Solaris

On Solaris 10, there are two structures that are important in understanding page cache operation: the segmap cache and the cachelist. The segmap cache contains file data pages that have been recently accessed using read/write system calls. The segmap cache has a size limit that is decided at startup; when it is full, older file pages get moved out of the segmap cache into the cachelist to make room for newly accessed pages. The size of the cachelist is not pre-determined; it can potentially use up all of the free memory on the system. The segmap cache functions as the part of the page cache that is bounded in size, but is somewhat protected in the sense that the pages there are not readily reclaimed. The cachelist is the part of the page cache that can grow to occupy available memory, but the kernel can easily reclaim these pages if the need arises, without resorting to page scanning. This division usually works well: for many workloads, Solaris 10 is able to take advantage of available memory for the page cache without incurring the overhead of page scanning and reclamation.

The memory reported by utilities such as `vmstat` as free memory is the combination of the memory in free pages (called the freelist) and the memory in cachelist pages. This memory can also be seen in the output of the `kstat -m unix -n` *`system_pages`* command as the `freemem` field. Solaris initiates page scanning and reclamation when `freemem` drops below a threshold.

The Solaris 10 tunable parameter `segmap_percent` is used to specify the size limit of the segmap cache. The tunable parameter `lotsfree` is used to specify the initial threshold for page scanning; when freemem (free + cachelist pages) drops below `lotsfree`, the operating system starts page scanning to reclaim pages. There are a few other parameters that specify thresholds related to intensity of page scanning; these can be found in the documentation on Solaris 10 tuning.

In most cases, the default values of the Solaris 10 tunable parameters for page cache operation work well. Rather than change the values of these parameters, administrators are more likely to have to interpret the memory state of the system properly for making tuning decisions, such as whether it is appropriate to tune a VxFS metadata cache given the memory usage on the system. In many cases, the output of `vmstat` has sufficient information for making such tuning decisions; the amount of free memory and the paging statistics reported by `vmstat` are particularly useful in understanding the state of the system. In some cases though, a more detailed picture of memory usage on the system may be required for making tuning decisions. For example, it might help to know how much of `freemem` is free and how much is in cachelist pages. One way to get this information is with the `::memstat` command in the `mdb` debugger.

Page cache monitoring and tuning on Linux

A good picture of memory allocation on a Linux server can be obtained from the `proc` pseudo file system. The following output is from an idle RHEL 5 server with 32GB of main memory:

```
# cat /proc/meminfo
MemTotal:     32958952 kB
MemFree:      31535244 kB
Buffers:        207860 kB
Cached:         784748 kB
SwapCached:          0 kB
Active:         636976 kB
Inactive:       427380 kB
HighTotal:           0 kB
HighFree:            0 kB
LowTotal:     32958952 kB
LowFree:      31535244 kB
SwapTotal:    34996216 kB
SwapFree:     34996216 kB
Dirty:              116 kB
Writeback:           0 kB
AnonPages:       71660 kB
Mapped:          22680 kB
Slab:           131728 kB
PageTables:       5660 kB
NFS_Unstable:        0 kB
Bounce:              0 kB
CommitLimit:  51475692 kB
Committed_AS:   363692 kB
VmallocTotal: 34359738367 kB
VmallocUsed:    392620 kB
VmallocChunk: 34359345639 kB
HugePages_Total:     0
HugePages_Free:      0
HugePages_Rsvd:      0
Hugepagesize:     2048 kB
```

The `MemFree`, `Cached`, `Active` and `Inactive` entries are especially useful in understanding page cache operation. `MemFree` shows the amount of memory in free pages; on this idle server, most of the memory is free. Under a file I/O intensive load, the page cache size will increase as memory pages are used to cache file data. This will be seen as an increase in the `Cached` field of `meminfo` output and a drop in `MemFree`. Pages in the page cache are placed on two LRU lists: the active list

and the inactive list, which are represented by the `Active` and `Inactive` entries in `meminfo` output. When the number of free pages drops below a threshold, the inactive list is scanned by the operating system to generate free pages. The operating system also refills the inactive list by moving the least used pages from the active list to the inactive list.

The memory picture on the same server on a file I/O intensive workload is shown below:

```
MemTotal:     32958952 kB
MemFree:        156076 kB
Buffers:          1092 kB
Cached:       24010608 kB
SwapCached:          0 kB
Active:        9541864 kB
Inactive:     14538184 kB
HighTotal:           0 kB
HighFree:            0 kB
LowTotal:     32958952 kB
LowFree:        156076 kB
SwapTotal:    34996216 kB
SwapFree:     34994724 kB
Dirty:          632728 kB
Writeback:           0 kB
AnonPages:       66792 kB
Mapped:          13880 kB
Slab:          1805652 kB
PageTables:       5416 kB
NFS_Unstable:        0 kB
Bounce:              0 kB
CommitLimit:  51475692 kB
Committed_AS:   398112 kB
VmallocTotal: 34359738367 kB
VmallocUsed:    429556 kB
VmallocChunk: 34359308711 kB
HugePages_Total:     0
HugePages_Free:      0
HugePages_Rsvd:      0
Hugepagesize:     2048 kB
```

In this example output, the `MemFree` value has dropped drastically and there is a corresponding increase in Cached memory. The `Active` and `Inactive` lists have also grown in size to accommodate the Cached pages. These fields show that the page cache has grown in response to file I/O, which is desirable. However, you

must also to ensure that the low levels of free memory is not resulting in degradation due to page scanning and reclamation. When free memory drops to low levels, such as in the output above, page scanning activity is to be expected. A good way to check whether the operating system is able to reclaim pages efficiently is to look at the output of the `sar` command from the `sysstat` package. The output of `sar -B` at 30 second intervals is shown below for the server under load:

```
pgpgin/s pgpgout/s   fault/s  majflt/s  pgfree/s pgscank/s pgscand/s pgsteal/s    %vmeff
48828.27  96497.33    228.33      0.37  61528.53  21715.20      0.00  21686.87     99.87
45856.85  96162.35    218.09      0.00  58152.85  20275.91      0.00  20255.75     99.90
46303.73  93568.30    208.47      0.00  58147.97  20398.93      0.00  20365.60     99.84
```

The output of `sar` shows sustained scanning (pgscank/s field) as expected. However, in this case, the scanning is seen to be very efficient (the %vmeff field), meaning that the operating system is able to reclaim (pgsteal/s field) almost every page that it scans.

Another check is to look at the CPU utilization of the kernel daemon `kswapd`, which is responsible for scanning pages and reclaiming unused pages.

In those cases where the overhead of page scanning is seen to be high, some of the Linux virtual memory tunable parameters can be used to improve efficiency. Manipulating the tunable parameters requires a good understanding of the workload. A few of the relevant parameters are as follows:

swappiness
: The default value of this parameter is 60. Lower values cause page cache pages to be reclaimed more eagerly. Higher values cause anonymous pages to be reclaimed more eagerly.

vfs_cache_pressure
: The default value of this parameter is 100. Lower values cause page cache pages to be targeted for reclamation during memory pressure. Higher values cause slab caches to be targeted.

Page cache monitoring and tuning on AIX

File pages in AIX are cached in memory by the Virtual Memory Manager (VMM) as permanent storage pages. File pages from VxFS file systems are classified as client pages; client pages are a subset of permanent storage pages. AIX has a number of tunable parameters that govern caching of file pages and page reclamation behavior. The following list contains some of the tunable parameters that are relevant for VxFS operation:

- `minperm`
- `maxperm`

- maxclient
- strict_maxclient
- lru_file_repage
- page_steal_method

See the AIX documentation for more information on these parameters.

In AIX 6.1 and later, most of these parameters are classified as restricted tunable parameters that you should change only on the recommendation of IBM support; restricted parameters are expected not to require tuning in most cases.

The vmstat command can be used to monitor the memory state of the system. Page reclamation activity can be seen from the pages scanned field and pages freed field of vmstat output. If these fields these have high values, it could indicate that page reclamation is having a negative impact on performance; in such cases, generally you should avoid tuning actions that add to memory pressure, such as increasing VxFS inode cache size limit. With the -v option, vmstat gives the actual usage of cache for file pages in the form of numperm and numclient values, for permanent storage pages and client pages, respectively.

Virtual Memory buffer tuning for VxFS on AIX

On AIX, moving data between the page cache and an underlying file system (paging-in and paging-out of data) is done using paging device tables (PDTs) and virtual memory buffer structures. When using Veritas File System (VxFS) on versions of AIX older than AIX 6.1 TL2, the number of PDTs and buffers might need to be tuned. With AIX 6.1 TL2 and later, buffers are allocated dynamically as needed and tuning is usually not required.

See "About tuning Virtual Memory for Veritas File System on AIX" on page 129.

About I/O modes

The default I/O mode in Veritas File System (VxFS) and most other file systems is buffered I/O with delayed writes. For writes in this mode, data is copied to the file data cache as part of the write call, but flushing of the data to disk is generally delayed to a later time. VxFS uses the operating system page cache as the file data cache. For reads, data is read into the file data cache before copying it to the buffers of the application that issued the read request. By default, file systems also perform concurrency control, typically in the form of locking, to ensure that file data is in a consistent state when there are concurrent reads and writes to the file. The default I/O mode generally provides good performance for many applications because of the benefits of caching and because application writes do

not incur the latency of disk access. Since data is only written to volatile memory as part of the write call and not to stable storage, such as disks, the default I/O mode provides only weak persistence guarantees, but for many applications that is adequate. Applications that use the default I/O mode can still use fsync and fdatasync calls to ensure that data has been made persistent at specific points.

Applications that need stronger persistence guarantees can be written to use synchronous rather than delayed writes. File systems like VxFS that are POSIX standard compliant allow applications to specify flags (O_SYNC and O_DSYNC) when a file is opened, to indicate that writes should be synchronous. For synchronous writes, the file system is required to flush data from the write out to disk before signaling the write call as complete. I/Os to files opened with the O_SYNC and O_DSYNC flags are still buffered in VxFS, meaning that data is copied to the page cache, unless applications or administrators prevent this explicitly.

In addition to the default mode and the synchronous I/O options available in compliance with the POSIX standard, VxFS allows control over the file system's handling of I/Os in the following ways:

- An application can set cache advisories by issuing ioctls on a file descriptor using the `vxfsio` interface. VxFS decides how to handle I/Os issued on a file descriptor based on the cache advisories that are set on it.
- An administrator can use the `convosync`, `mincache`, and `cio` mount options to set I/O handling behavior for a file system at mount-time.

For information about these features of VxFS and options for controlling the handling of I/Os in VxFS, see the *Veritas File System Administrator's Guide*.

In practice, these features are most commonly used to enable direct I/O and Concurrent I/O, which are important alternatives to buffered I/O in VxFS. The information here is limited to the performance and tuning considerations for direct I/O and Concurrent I/O, and how these modes can be enabled.

Direct I/O is a mode supported by VxFS and many other file systems where file data is not cached in the file system, but is moved directly between application buffers and disk devices. Direct I/O has advantages and disadvantages compared to buffered I/O: direct I/O avoids the overheads of data copying and cache management that are incurred with buffered I/O, but it also forgoes the performance benefits of caching that are available with buffered I/O. Direct I/O can provide better performance than buffered I/O for some workloads, usually in cases where there is not much benefit from caching in the file system. Database workloads often fall in this category, because databases maintain their own cache and caching in the file system is often redundant. Direct I/O can be enabled for I/Os on a file descriptor by setting the `VX_DIRECT` cache advisory on that file descriptor using the vxfsio interface. It can be enabled for a whole file system using the `convosync=direct` and `mincache=direct` mount options. When direct

I/O has been enabled, and I/Os meet certain alignment constraints, VxFS avoids copying file data from the I/O into the page cache.

See "Direct I/O" on page 81.

VxFS also has a feature called discovered direct I/O, which applies to large read and write requests. In discovered direct I/O, requests above a certain size are handled similarly to direct I/O, that is, data is transferred directly between application buffers and storage devices without copying to the page cache, even though direct I/O has not been explicitly enabled.

See "Discovered direct I/O" on page 83.

VxFS also supports a special mode especially suited for databases called Concurrent I/O. This mode is similar to direct I/O in that data is moved directly between application buffers and disk devices without caching in the file system. Additionally, Concurrent I/O relaxes the locking that the file system normally performs: normally, a write locks out other reads and writes to the same file, but with Concurrent I/O, writes and reads can proceed concurrently. The Concurrent I/O mode is useful for databases since they often issue non-overlapping writes concurrently to regions of the same large file; Concurrent I/O improves performance by allowing these accesses to proceed concurrently, rather than serializing them at the file system level. Concurrent I/O can be enabled on a file descriptor by setting the `VX_CONCURRENT` cache advisory using the vxfsio interface. Concurrent I/O can be enabled for a whole file system using the `cio` mount option.

Tuning read-ahead

Read-ahead is a technique that Veritas File System (VxFS) uses to improve performance when it detects a regular pattern in the way reads are being issued to a file. In the simplest form of read-ahead, VxFS detects that a file is being read sequentially and in the background starts issuing read-ahead requests. That is, VxFS issues reads for portions of the file that have not yet been requested by the application. These read-ahead requests move data asynchronously from storage devices to the page cache. Ideally, the application would continue to issue sequential reads and subsequent reads would find the data in the page cache, resulting in large performance gains for the application. In the more sophisticated form of read-ahead, VxFS can detect some complex read patterns, such as strided reads, and issue appropriate read-ahead requests in the background. Since read-ahead involves moving data in the background from the storage devices to the page cache, it applies to the buffered I/O modes and not to direct I/O and Concurrent I/O.

Read-ahead can result in large gains in performance because accessing data from memory caches can be orders of magnitude faster than accessing from storage

devices. However, read-ahead can also lead to performance problems in some cases. Some of the potential problems are as follows:

- Read-ahead can generate a flood of read requests to the underlying disk device queues that slows down I/Os for other applications.
- Read-ahead can cause the page cache to be filled up with read-ahead data, causing useful data from other applications to be evicted.
- If the application does not continue the access pattern that triggered read-ahead, work done in the read-ahead is wasted.
- If the application is slow to process the data it is reading, and if there is pressure on the page cache, some of the file data pages that have been brought in by read-ahead can get invalidated before they are requested by the application. Work done in the read-ahead is hence wasted.

VxFS tries to minimize these problems in the way it does read-ahead by adapting intelligently to conditions. As one example, to prevent a lot of wasted work, VxFS starts read-ahead in small size units and then increases the size of read-ahead when the read pattern continues. But, in some cases, you might need to tune read-ahead behavior manually. VxFS provides a number of tunable parameters for tuning read-ahead.

The tunable parameters controlling read-ahead in VxFS can be changed on the fly using the `vxtunefs` utility. This makes it easier to try out different read-ahead settings to find one that is appropriate for your workload. Also, read-ahead parameters can be set at a per mount point granularity. You can have read-ahead tuned differently for each VxFS file system that is mounted.

Setting the type of read-ahead

The value of the tunable parameter `read_ahead`, which is one of the Veritas File System (VxFS) tunable parameters displayed and set using the `vxtunefs` utility, controls the overall nature of read-ahead. The `read_ahead` parameter can have the following values:

0 Disables read-ahead altogether.

1 Enables normal read-ahead. In this setting, VxFS issues read-ahead when it detects simple sequential access pattern. This is the default value.

2 Enables enhanced read-ahead. In this setting, in addition to sequential access patterns, VxFS looks for more complex access patterns, such as strided reads, backward strided read, and accesses from multiple processes or threads where each access is sequential.

Observing read-ahead behavior

When a buffered I/O read request is received, Veritas File System (VxFS) determines whether or not it should trigger read-ahead. VxFS maintains the following counters, which can be observed in the output of the `vxfsstat -v` command, and can be useful in tuning read-ahead:

vxi_read_rand
: This gives the number of read requests that were classified as random reads. These are reads that would not trigger read-ahead activity.

vxi_read_seq
: This gives the number of read requests that were not classified as random reads; these could trigger read-ahead activity. In the case of normal read-ahead, this counter gives the number of sequential read requests. In the case of enhanced read-ahead, this counter gives the number of read requests that VxFS detected as following one of the many patterns that it looks for in the enhanced mode.

These counters are the cumulative values for all VxFS file systems that are in use on the given system. When a single VxFS file system is in use, interpreting these values is straightforward, but in other cases more analysis might be required. The values of these counters depend on the workload and also on what type of read-ahead is in effect. When read-ahead is disabled (tunable parameter `read_ahead=0`), the `vxi_read_seq` counter should also be 0. The values of these counters can be recorded before and after changing the `read_ahead` tunable parameter to see what is the right setting for your workload. A much higher ratio of `vxi_read_seq`/`vxi_read_rand` with `read_ahead=2` compared to `read_ahead=1` indicates that your workload could benefit from enhanced read-ahead.

If the proportion of read requests classified as random requests is high, the other read-ahead tunable parameters might not have much impact on performance. The proportion is high if the following formula has a high value:

```
vxi_read_rand / (vxi_read_seq + vxi_read_rand)
```

Normal read-ahead on Veritas File System

When the tunable `read_ahead` parameter is set to 1, Veritas File System (VxFS) initiates read-ahead when it detects a simple sequential access pattern. Simply stated, a sequential pattern is one where each read continues where the previous one finished. VxFS initiates a small amount of read-ahead when it first detects sequential access, and increases the amount of read-ahead as the sequential pattern continues. The following list describes the characteristics of VxFS operation when normal read-ahead is in effect:

- On every read request, VxFS computes a next expected read offset for the file as the sum of the offset of the current read and the size of the current read (next_expected_offset = offset_of_current_request + size_of_current_request). This value is stored as a field in the in-core inode for the file.
- On every read request, VxFS compares the value of the read offset to the value of next expected offset stored in the inode. If they are the same, VxFS flags this read as a sequential read and increments the vxi_read_seq counter. Otherwise, VxFS increments the `vxi_read_rand` counter.
- VxFS is able to tell whether a sequential read is part of a continuing sequential run.

  For example, consider a large file that is being read sequentially by the application in 32k read requests. The second read request will be flagged by VxFS as a sequential read. In the case of the third read, VxFS flags it as a sequential read and is also able to detect that it is part of a continuing sequential run.
- VxFS schedules read-ahead differently for the first sequential read (initial read-ahead) and for read requests that are part of a continuing sequential run (continuing read-ahead). In initial read-ahead, VxFS tries to schedule a modest amount of read-ahead since there is not much evidence at this point to indicate that the sequential access pattern will continue. If the sequential access pattern continues, there is reason to believe that the access pattern is strongly sequential, and therefore VxFS increases the amount of read-ahead.
- For the first sequential read, VxFS initiates read-ahead of 4 regions of the file starting with the current offset. VxFS uses the tunable parameter `read_pref_io` to decide the size of each region. The size of each read-ahead region is `read_pref_io` or twice the read request size, whichever is smaller (read_ahead_region_size = MIN (read_pref_io, 2 * size_of_request)). If the read-ahead size is not aligned to the page size, the size is rounded up to align it to the page size.
- The 4 regions that are scheduled to be read in the initial read-ahead constitute a read-ahead pipeline. VxFS schedules new read-ahead–if the sequential access pattern continues–when the read offset crosses a region boundary. Thus, continuing read-ahead adds to the pipeline setup in the initial read-ahead.
- For continuing read-ahead, the read-ahead size is doubled each time compared to the previous read-ahead size, until the read-ahead size reaches the maximum read-ahead size that VxFS allows. The maximum read-ahead size is determined by the tunable parameters `read_pref_io` and `read_nstream`. The maximum read-ahead size is calculated as follows:

  ```
  read_pref_io * read_nstream
  ```

- Once the maximum read-ahead size is reached, VxFS continues to issue read-ahead of the maximum size if the sequential access pattern continues. read-ahead is issued not on every read, but whenever the read offset advances past some internal region markers that VxFS maintains.
- There is an exception to the above description when the file has a sequential or random advisory set on it. When the file has a sequential advisory (`VX_SEQ`) set, VxFS starts issuing read-ahead at the maximum read-ahead size, that is, `read_pref_io * read_nstream`, from the first sequential read. When the file has a random advisory (`VX_RANDOM`) set, VxFS does not issue read-ahead.

  For more information on cache advisories, see the `vxfsio`(7) manual page.

There are some common cases where read-ahead is beneficial, but the normal read-ahead setting in VxFS might not initiate read-ahead. Consider, for example, a file that is being read concurrently by two application processes, each of which is reading the file sequentially. It would be beneficial to schedule read-ahead in this case, just as in the case of a sequential read by a single application. However, the concurrent read requests from the two processes can interfere with the sequential access tracking mechanism, changing the values of the next expected offset in such a way that VxFS does not trigger read-ahead at all or triggers read-ahead only for some of the reads. The enhanced read-ahead setting (read_ahead=2) is designed to enable read-ahead for this and some other cases where the normal read-ahead behavior is inadequate.

Important tunable parameters for read-ahead size

The size of read-ahead is controlled by two tunable parameters: `read_pref_io` and `read_nstream`. These parameters can be displayed and set using the `vxtunefs` utility. This implies that they can be set differently for each Veritas File System (VxFS) mount point and can be changed on the fly without requiring a reboot or module reload. These parameters have an important characteristic: when VxFS is created over a Veritas Volume Manager (VxVM) volume, VxFS queries the underlying volume at mount time and sets these tunable parameters based on the geometry of the underlying volume. When VxFS is not created over a VxVM volume, the default value for each of these parameters is static; the default value is pre-determined and not dependent on the disk device characteristics.

These tunable parameters have the following function:

read_pref_io
: This tunable parameter plays a role in determining the size of read-ahead both for initial read-ahead and continuing read-ahead as described in the previous section. The default value of this parameter is 64 KB. When mounted over a striped VxVM volume, the initial value of this parameter is set to the stripe unit size of the underlying volume.

read_nstream — This parameter is used together with `read_pref_io` to determine the maximum size of read-ahead that VxFS will perform. The maximum read-ahead size is `read_pref_io` * `read_nstream`. The default value of `read_nstream` is 1. When mounted over a striped VxVM volume, the initial value of this parameter is set to the stripe width of the underlying volume (the number of columns in the stripe).

Essentially, `read_pref_io` limits the size of initial read-ahead and `read_pref_io` * `read_nstream` gives the maximum read-ahead size.

Enhanced read-ahead in Veritas File System

The enhanced read-ahead mode in Veritas File System (VxFS) is conceptually similar to the normal mode, but enhanced read-ahead mode results in read-ahead getting triggered in some cases where the normal mode would not trigger read-ahead. The enhanced mode is useful in the case of workloads where multiple threads or multiple processes read the same files, with each thread or process reading sequentially. The normal read-ahead mode may not recognize these accesses as sequential and may not trigger read-ahead. The sequential access detection mechanism, described earlier for the normal read-ahead mode, is expanded in the enhanced mode to keep track of per-thread accesses, for a limited number of threads. This allows VxFS to detect that an individual thread is issuing sequential reads; VxFS can then respond by issuing read-ahead. The enhanced read-ahead mode can also detect strided access patterns, although these patterns tend to be less common in enterprise workloads.

How to tune read-ahead

The default setting of normal read-ahead (tunable parameter read_ahead=1) usually works well because most application accesses are either random or sequential; more complex access patterns are relatively rare. One common case where `read_ahead=1` might not be sufficient is when there are multiple application processes reading the same files concurrently and each process is issuing sequential requests. In this case, Symantec recommends that you manually set `read_ahead=2` using the `vxtunefs` command. Of course, `read_ahead=2` is also beneficial when there are more complex access patterns.

Symantec recommends that the values of the `vxfsstat` counters vxi_read_rand and vxi_read_seq be observed at different settings for the `read_ahead` tunable parameter. If setting `read_ahead=2` is seen to increase the ratio of vxi_read_seq / vxi_read_rand, your workload most likely has characteristics that can benefit from enhanced read-ahead in Veritas File System (VxFS).

Symantec recommends that `read_pref_io` be set at a value close to its default value of 64 KB. An important role of this parameter is to limit the size of initial read-ahead. If this parameter is tuned to a very large value, a single accidental sequential access can trigger a large amount of read-ahead, which can result in a lot of wasted work.

For workloads with predominantly sequential accesses on large files, you might need to tune the value of `read_nstream` to get the appropriate maximum read-ahead size. There are two main factors in choosing the value of `read_nstream`:

- The throughput of the underlying storage device: When the underlying volume or LUN is striped, it can sustain higher I/O throughput; in such cases, a high value of `read_nstream` is appropriate. When VxFS is mounted over a VxVM volume, the initial value of `read_nstream` is based on the volume geometry; in such cases, further tuning might not be required. When striping has been done at the disk-array level, `read_nstream` is likely to require tuning.
- Impact on other applications: A larger value of `read_nstream` might slow down I/O for other processes by enabling large amounts of read-ahead.

Summary of read-ahead tuning

Read-ahead can result in large gains in performance for applications using buffered I/O by moving data in the background from disk devices to the page cache before they are requested by the applications. In the default read-ahead setting, VxFS initiates read-ahead on sequential accesses. VxFS counters displayed by the `vxfsstat` command, namely vxi_read_seq and vxi_read_rand, can be observed to determine if the workload has heavily sequential accesses. The parameters `read_pref_io` and `read_nstream` control read-ahead size. Symantec recommends that `read_pref_io` be kept close to its default value of 64 KB to keep initial read-ahead at a modest size. For workloads with predominantly sequential accesses on large files, you might need to tune `read_nstream` manually, especially when striping has been done at the disk-array level rather than in VxVM.

Read flush-behind in Veritas File System

Read flush-behind is a technique that Veritas File System (VxFS) uses to minimize the degradation in system performance that can sometimes result from sequential reads of large files. The degradation can result in the following issues:

- Sequential scans of large files cause a steady influx of file data into the page cache. This can cause the number of free pages on the system to drop, causing the operating system to initiate page scanning and reclamation. The page reclamation activity consumes CPU cycles and can affect performance.

- Since the pages that are part of the problematic scan are recent and since page reclamation algorithms retain recent pages and reclaim older pages, useful data pages of other files and processes get evicted from the page cache.
- Applications that reference the evicted pages see a drop in performance because they must be read from secondary storage.

VxFS addresses the above problems in the following manner:

- VxFS detects sequential runs and tracks how large they are.
- At the completion of a read, VxFS checks if the system is under page cache pressure. If it is, VxFS attempts to free a chunk of pages that are part of the current sequential run for which reads have completed. This creates free pages, easing the pressure on the page cache and reducing the number of other useful pages that might get targeted by the operating system page reclamation.
- Read flush-behind happens in chunks of size `write_pref_io` * `write_nstream`, where `write_pref_io` and `write_nstream` are tunable parameters that also control write flushing in VxFS.
- A downside of this technique is that if the file from which pages are being freed is read again by the application, the data will not be in the page cache and must be read back from the disk storage.

Read flush-behind example

This section describes a scenario that illustrates how read flush-behind works. Read flush-behind happens only when Veritas File System (VxFS) detects page cache pressure. The determination of whether there is page cache pressure is operating system-specific. For this example, let us assume a system under page cache pressure. Let us assume a value of 64 KB for `write_pref_io` and a value of 4 for `write_nstream`. Let us assume a large file of 100 MB is being read sequentially, with each read request being 64 KB in size.

The first read request (offset 0, size 64 KB) is classified by VxFS as a random read. Subsequent requests are classified as sequential requests, with offset 0 being the start of the sequential run. VxFS also maintains a read flush offset to keep track of read flush-behind activity; before any read flushing has happened, this is the same as the start of the sequential run: offset 0.

At the completion of the fourth read request, the current sequential run is of size 256 KB, which is also the read-flush behind size. At this point, VxFS invalidates all pages for the file in the offset range 0 KB to 256 KB. That is to say, a chunk of size equal to the read flush-behind size, and starting at the read flush offset, is invalidated. After the page invalidation is complete, the read flush offset is set to 256 KB, which is the offset in the current sequential run up to where read flushing

has completed. Similarly, at the completion of the eighth request, VxFS invalidates all pages for the file in the offset range of 256 KB through 512 KB. The read flush offset is updated to 512 KB.

As the sequential run continues, VxFS repeats this pattern of invalidating chunks of the read flush-behind size and adjusting the read flush offset accordingly. When VxFS gets a read request that is not sequential, meaning a read request that is a random read, VxFS resets the read flush-behind point to the offset of the random read and gets ready to track a new sequential run.

Tuning read flush-behind

The tunable parameters controlling read flush-behind are `write_pref_io` and `write_nstream`. These are also the tunable parameters controlling write flushing for sequential writes.

Tuning Veritas File System buffered writes

The default behavior of writes in Veritas File System (VxFS) is to copy the written data to the page cache and return; the data is written to storage devices behind the scenes. These writes are called delayed buffered writes. For many applications, the trade-off in this delayed-write model is acceptable: writes give good performance but there is a small risk of data loss in case the system crashes before data is flushed out. Applications that use delayed writes should be able to tolerate or recover from such data loss. The journaling mechanism in VxFS protects the integrity of file system metadata in the event of a crash; it does not protect against the kind of data loss that can occur with buffered writes.

However, the delayed write model is not appropriate in all cases. For some applications, it is enough to follow-up a series of delayed writes with an `fsync`() or `fdatasync`() call to ensure that data is persistent. But, for other applications, even this might not be good enough. The following writes are commonly used by applications as alternatives to delayed writes when better persistence guarantees are needed:

synchronous writes	These writes not only copy data to the page cache, but also flush the data to disk as part of the `write`() call. Subsequent reads of the same file blocks can possibly get the data from the page cache without having to read from storage devices.
direct writes	These writes bypass the page cache and write data directly from application buffers to storage devices. See "Direct I/O" on page 81.

Synchronous buffered writes

Applications that need strong persistence guarantees for writes usually indicate this to the file system using the `O_SYNC` and `O_DSYNC` flags when opening a file. These flags and the requirements that they impose on the file system are described in the POSIX standard. For an `O_SYNC` or `O_DSYNC` write, VxFS copies data from the write to the page cache, but also flushes the data to disk before signaling the write as complete; these writes are therefore called synchronous writes. An `O_SYNC` write also requires all file metadata modified as a result of the write, like block layout changes and modification time update, to be written out synchronously. An `O_DSYNC` write requires modified metadata to be written synchronously only if the metadata is required to access the data that was written; in the common case of existing file blocks getting overwritten, an `O_DSYNC` write does not usually require metadata to be written synchronously.

In some cases, it might be desirable to change the performance characteristics of an application without rewriting the application by changing the way I/Os issued by the application are handled. VxFS supports this through the `convosync=`*`option`* mount option, which changes the way I/O to a file opened with the `O_SYNC` or `O_DSYNC` flag is handled. You can specify the following values for *option*:

- closesync
- delay
- direct
- dsync
- unbuffered

See the *Veritas File System Administrator's Guide*.

These values change the persistence characteristics of writes from what was expected when the application was written and should be used with caution. While a mount option such as `convosync=delay` can improve performance by causing synchronous writes to be handled as delayed writes, the option can very easily lead to correctness issues.

The mount option `convosync=direct` enables direct I/O for all files in a VxFS file system that were opened with the `O_SYNC` or `O_DSYNC` flags. More commonly, the `convosync` and `mincache` mount options are used together (`convosync=direct,mincache=direct`) to enable direct I/O for a VxFS file system; this option enables direct I/O for files opened with the `O_SYNC` or `O_DSYNC` flags, as well as files opened in default mode.

See "Delayed buffered writes" on page 76.

The datainlog optimization

Veritas File System (VxFS), by default, performs an optimization in its handling of some synchronous writes by writing data from these writes, along with modified metadata, to the intent log. These synchronous writes that are written to the intent log are called logged-writes. The VxFS intent log is used primarily for recording metadata changes; except in the case of logged-writes, file data is not written to the intent log. The `-o datainlog` mount option enables the logged-write optimization and the `-o nodatainlog` mount option disables it. The logged-write optimization is enabled by default; that is, unless the `nodatainlog` mount option is explicitly used, VxFS performs the logged-write optimization.

The logged-write optimization applies to `O_SYNC` writes that are 8 KB or less. There are a few other internal checks that VxFS performs before handling a write as a logged-write. An `O_SYNC` write requires that the data from the write as well as modified metadata should be flushed to disk before the write is signaled as complete. For a logged-write, VxFS writes both the data and metadata to the intent log as a single synchronous write and then signals the write as complete; this meets the persistence requirements for `O_SYNC` writes. The actual file blocks that are modified by the logged-write are updated asynchronously from the page cache to disk after the write call is signaled as complete. In contrast, for an `O_SYNC` write that is not performed as a logged-write, two synchronous disk writes are typically performed before the write call is signaled as complete--one synchronous write for the file data and another for the file metadata. The logged-write optimization thus results in lower latency for small `O_SYNC` writes. The downside of a logged-write is that the actual file data generally gets written to disk twice--once to the intent log and once to the actual file blocks on disk--but, the second is asynchronous and usually has low impact on performance.

The performance difference between the `datainlog` and `nodatainlog` options will depend on the workload. `O_SYNC` writes larger than 8 KB are not performed as logged-writes even when the `datainlog` option is enabled. If the workload does not have small `O_SYNC` writes, `datainlog` and `nodatainlog` should perform about the same. For most workloads, `datainlog` should perform better than or as well as the `nodatainlog` option. But, as explained above, there are tradeoffs involved in the logged-write optimization, and for some workloads, `nodatainlog` might perform better.

Delayed buffered writes

Delayed buffered writes in Veritas File System (VxFS) copy the written data to the page cache as part of the write call, and mark these pages as dirty to indicate that they need to be flushed out. Most commonly, delayed writes get written out to storage devices when the operating system page flusher periodically requests

the file system to flush out dirty pages, although there are other mechanisms by which delayed writes are written out.

Delayed writes can give good performance for a number of reasons:

- Applications see low latency on writes because there is no disk access involved.
- If the same blocks are written again before they have been flushed out, you save on the number of writes going to the storage devices. In scenarios where storage is the bottleneck, this can improve overall performance.
- Multiple delayed writes can often be combined and written in one large efficient disk write. A single large write is typically more efficient than multiple small writes of the same aggregate size. If each write were being written to disk as it was received, this optimization would not be possible.

In some cases, you might find it useful to change the performance or persistence characteristics of an application without rewriting it by changing the way I/Os issued by it are handled. The `mincache=`*`option`* mount option alters handling of I/O requests for files opened in the default mode and hence changes the way delayed writes are handled. In contrast, the `convosync` option, discussed in the context of synchronous writes, alters the handling of I/O requests for files opened with the `O_SYNC` or `O_DSYNC` flags. You can specify the following values for *option*:

- closesync
- direct
- dsync
- unbuffered
- tmpcache

See the *Veritas File System Administrator's Guide*.

The `mincache` options are used primarily for stronger persistence characteristics for writes compared to delayed writes, although performance is also often a consideration for using `mincache=direct`. The mount option `mincache=direct` is used to enable direct I/O for files opened in the default mode. More commonly, the mount option `convosync=direct,mincache=direct` is used to enable direct I/O for a VxFS file system; this enables direct I/O for files opened with the `O_SYNC` and `O_DSYNC` flags, as well as those opened in default mode.

See "Synchronous buffered writes" on page 75.

Delayed writes can also sometimes result in performance problems because of too many dirty pages in the cache that need to be flushed out. Write throttling, write flush-behind, and I/O flush throttling are mechanisms in VxFS that are intended to minimize these problems.

See "Write throttling" on page 78.

See "Flush-behind for sequential writes" on page 78.

See "Throttling I/O flushes" on page 80.

Write throttling

Veritas File System (VxFS) allows administrators to set a limit on the amount of dirty data that a file can have in the page cache. This per-file limit is specified using the tunable parameter `write_throttle`. When a new write to a file pushes the amount of dirty data in the page cache for that file above the `write_throttle` limit, VxFS initiates flushing of all dirty pages of the file.

The `write_throttle` tunable parameter can be manipulated using the `vxtunefs` utility. The `write_throttle` parameter can be changed on the fly, and can be set differently for each mounted VxFS file system. The default value of the `write_throttle` tunable parameter is 0. This is interpreted to mean that write throttling is disabled; in this case, there is no limit on the amount of dirty data that a file can have in the page cache. When a value is explicitly specified for `write_throttle`, VxFS uses this value as the number of bytes of dirty data that a file can have in the page cache. The default is recommended for this parameter.

The VxFS counter `vxi_write_throttle`, which you can see in the output of the `vxfsstat -v` command, is incremented each time VxFS flushes out a file because the amount of dirty data for the file exceeded the `write_throttle` parameter.

Flush-behind for sequential writes

Veritas File System (VxFS) has a write flush-behind mechanism for sequential writes that is designed to efficiently write data in large chunks to the underlying storage devices. VxFS performs sequential access pattern detection for buffered writes, just like it does for reads. When there are one or more sequential writes on a file forming a sequential run that is large enough, VxFS writes out that region without waiting for the normal flushing mechanism for delayed writes. This behavior is called write flush-behind and is controlled by the tunable parameters `write_pref_io` and `write_nstream`. The flush-behind happens in chunks of size `write_pref_io` * `write_nstream`.

One of the advantages of the delayed write mode is that a delayed buffered write potentially enables many small writes to be clustered and written out as one large efficient write. In the case of sequential writes forming a large sequential run, VxFS already has a large efficient write, so there is less incentive for delaying the write further. Flushing out the sequential region early also helps prevent too much dirty data from accumulating in the page cache.

The mechanism for sequential access pattern detection classifies every buffered write as either a sequential write or a random write. VxFS maintains the following counters that can be observed in `vxfsstat -v` output:

vxi_write_seq	This counter provides the number of write requests classified as sequential writes.
vxi_write_rand	this counter provides the number of write requests classified as random writes.

These counters can be useful in understanding the nature of write requests in the workload and can help in tuning write flush-behind.

The tunable parameters `write_pref_io` and `write_nstream` together determine the write flushing size. The values of these parameters can be displayed and set using the `vxtunefs` command. This implies that the parameters can be changed on the fly without requiring a reboot or a module reload and can be set differently for each mounted VxFS file system. Similar to the tunable parameters `read_pref_io` and `read_nstream`, when VxFS is mounted over a VxVM volume, these parameters are set at mount time by querying the underlying volume for its geometry. The tunable parameters are described as follows:

write_pref_io	When mounted over a striped VxVM volume, the value of this parameter is set to the stripe unit size of the underlying volume . The default value of this parameter is 64 KB.
write_nstream	When mounted over a striped VxVM volume, the value of this parameter is set to the number of columns in each stripe of the underlying volume. The default value of this parameter is 1.

For workloads that access large files sequentially, the values of `write_pref_io` and `write_nstream` should be chosen such that the flush size (write_pref_io * write_nstream) results in efficient I/O. When the file system has been created over a striped VxVM volume, in which the striping was done in VxVM rather than in the disk array, the values of `write_pref_io` and `write_nstream`, obtained by querying the volume at mount time, generally result in efficient I/O. However, when striping has been done in the disk array, `write_nstream` may need to be manually tuned to improve the efficiency of write flushing. Rather than the default of 1, a higher value such as 16 is a good start for `write_nstream` in such cases; with `write_pref_io` at 64 KB, this would result in a write flush size of 1 MB. Since these parameters can be easily changed on the fly, their values can be refined based on observing the effect of different settings.

Throttling I/O flushes

Veritas File System (VxFS) has a mechanism aimed at preventing situations where the flushing of data for a file ties up storage bandwidth and causes performance problems. VxFS has a limit on the amount of data that can be outstanding in disk writes for a file. This per-file limit is specified by the value of the tunable parameter `max_diskq`. VxFS tracks the amount of data being flushed out for each file. If there is a request to flush a file's data to disk, and VxFS detects that the amount of data being flushed out for that file exceeds `max_diskq`, VxFS pauses the requester for a while until some of the outstanding disk writes have completed and the amount of data in outstanding disk writes drops below `max_diskq`. This mechanism is called I/O flush throttling.

Each time I/O flush throttling happens, VxFS increments the counter `vxi_flush_throttle`. Observing the value of this counter in `vxfsstat -v` output can help understand how much flush throttling is happening on a system; this can help in deciding whether the value of the `max_diskq` tunable parameter should be tuned.

The value of the `max_diskq` tunable parameter can be displayed and set using the `vxtunefs` utility. This implies that it can be changed on the fly without a reboot or module reload and it can be set differently for each mounted VxFS file system. The value of this parameter is the I/O flush limit specified in bytes. Thus, the parameter specifies the number of bytes that can be pending in writes for a file.

The default value of `max_diskq` is 1 MB or 16 times the write flush size, whichever is greater. The write flush size is given by `write_pref_io * write_nstream`. When the value of `max_diskq` is explicitly specified, the value is set to the specified value or 4 times the write flush size, whichever is greater.

Tuning Veritas File System buffered I/O on AIX

Buffered I/O involves interactions between Veritas File System (VxFS) and the operating system's virtual memory manager using operating system-specific interfaces. On the AIX platform, VxFS has certain tunable parameters for buffered I/O that are not available on other platforms. These parameters are described in the online manual page for the `vxtunefs` command on AIX.

See the `vxtunefs`(1M) manual page.

The following list provides a brief summary of the buffered I/O parameters:

- Paging-in and paging-out of data: The parameters `drefund_enabled` and `num_pdt`, which are changed using the `vxtunefs -D` command, and the number

of buffer structures, which is changed using the `vxtunefs -b` command, relate to paging-in and paging-out of data for buffered I/O.
See "About tuning Virtual Memory for Veritas File System on AIX" on page 129.

- Operating under memory pressure: The parameters `lowmem_disable`, `read_flush_disable`, and `write_flush_disable`, which are changed using the `vxtunefs -D` command, enable or disable actions taken under memory pressure.
- Flushing of dirty data: The parameters `sync_time`, `chunk_flush_size`, `lm_dirtypage_track_enable`, `fsync_async_flush_enable`, and `bkgrnd_fsync_enable`, which are changed using the `vxtunefs -D` command, govern tracking and flushing of dirty data in the page cache.

Direct I/O

Direct I/O in Veritas File System (VxFS) transfers data directly between application buffers and storage devices without copying file data to the page cache. Direct I/O is an important alternative to buffered I/O; for some workloads, direct I/O can improve performance significantly compared to buffered I/O. Direct I/O also has disadvantages compared to buffered I/O and the choice between the two should be made based on the characteristics of the workload and the environment. Direct I/O is not the default mode and needs to be enabled by an administrator or by the application; however, note that VxFS has a feature called discovered direct I/O, where it automatically handles some large I/O requests without buffering. When direct I/O has been enabled, each I/O must meet some alignment constraints in order for the I/O to be performed as direct I/O; if these constraints are not met, VxFS performs the I/O as buffered I/O, even though direct I/O was requested.

See "Discovered direct I/O" on page 83.

Keep in mind the following information regarding direct I/O:

- Direct I/O can be enabled by an administrator at mount time for a file system using the `convosync` and `mincache` mount options. The mount option `convosync=direct` enables direct I/O for files opened with the `O_SYNC` and `O_DSYNC` flags. The mount option `mincache=direct` enables direct I/O for files opened without these flags. The two mount options are often used together, such as `-o convosync=direct,mincache=direct`, to enable direct I/O for both types of files.
- Direct I/O can be enabled by an application by setting the `VX_DIRECT` cache advisory on a file descriptor using the `vxfsio` interface. This enables direct I/O only for I/Os on that file descriptor.

- Even if direct I/O has been enabled, an I/O is performed as direct I/O only if certain constraints are met. Among these are alignment constraints that can vary with the operating system. In general, the I/O must start on a 512-byte offset and the I/O size must be a multiple of 512 bytes. If these constraints are not met, VxFS performs the I/O as buffered I/O, even if direct I/O has been enabled.
  For more information about these constraints, see the *Veritas File System Administrator's Guide*.
- When a direct I/O read is performed, the counter `vxi_read_dio` maintained by the VxFS module is incremented; for a direct I/O write, the counter `vxi_write_dio` is incremented. These counters can be displayed with the `vxfsstat -v` utility and can be used to verify that direct I/O is being performed. These are module-level counters that keep track of direct I/O requests on any of the mounted VxFS file systems on that system. If there is only one VxFS file system active, these counters directly show activity for that file system. But, when there are multiple VxFS file systems that are active on the same system, these counters should be interpreted carefully. As described in the sections on buffered I/O, VxFS maintains a different set of counters that track buffered I/O reads and writes.
- Writes in the direct I/O mode are signaled as complete only after the data has been written to storage devices; any metadata that is required to retrieve the written data is also flushed out before the `write`() call is signaled as complete. Metadata that is not required to retrieve the data written, like the file modification time, may not be flushed out before the `write`() call completes. The persistence guarantees with direct I/O are thus better than with delayed writes and match the guarantees in the `O_DSYNC` mode. It is usually acceptable to enable direct I/O for applications that are written to use synchronous writes.

The main benefits of direct I/O compared to buffered I/O are as follows:

- Direct I/O avoids the overhead of copying data from application buffers to the page cache.
- Direct I/O avoids the overhead of cache management, which involves interactions with the virtual memory management system of the operating system.
- By not using the page cache, direct I/O reduces memory usage and avoids displacing other useful file data from the page cache.

The main disadvantages of direct I/O are as follows:

- For workloads that re-read the same blocks, data needs to be read from storage devices each time. In contrast, with buffered I/O, it is likely that blocks being

read multiple times will be found in the page cache, thus reducing the number of disk accesses.

- Read-ahead that is available with buffered I/O is not feasible with direct I/O.
- Delayed write optimizations such as combining multiple small writes into a large efficient write are not feasible with direct I/O.

The impact that the advantages and disadvantages are likely to have given the characteristics of a workload should guide the choice of whether direct I/O should be enabled for the workload. Direct I/O is often, though not always, beneficial for database workloads. Since databases maintain their own cache, lack of data caching in the file system is not a disadvantage for many database workloads; rather it is usually the preferred option since it avoids double buffering of the same data. Database reads and writes also generally meet the alignment requirements of direct I/O. For some database workloads, like transaction processing workloads, the Concurrent I/O mode usually works better than plain direct I/O. When direct I/O or Concurrent I/O are used for database workloads, sizing the database cache appropriately becomes particularly important.

See "Concurrent I/O" on page 84.

Discovered direct I/O

The discovered direct I/O feature of Veritas File System (VxFS) causes large I/O requests in the buffered I/O mode to be handled in a manner similar to direct I/O. For these requests, VxFS copies data directly between application buffers and storage devices without copying data to the page cache. This behavior is controlled by the following tunable parameter:

discovered_direct_iosz
: In the buffered I/O mode, read and write requests above this size are performed as discovered direct I/O, that is, without copying data to the page cache. This tunable parameter can be displayed and set using `vxtunefs`; its value can be set differently for each mounted file system and changes take effect immediately without need for a reboot or module reload. The value of this parameter is interpreted as the size in bytes above which VxFS performs discovered direct I/O. The default value is 256 KB.

Metadata flushing with discovered direct I/O happens according to the rules of the buffered I/O mode that is in effect rather than according to the rules of the direct I/O mode. Persistence guarantees for metadata in the default mode, that is, in the buffered I/O mode with delayed writes, are weaker than in the direct I/O mode in VxFS. In this sense, discovered direct I/O is not strictly direct I/O, but this is a detail that does not generally impact tuning considerations. The discovered direct I/O feature allows VxFS to switch between buffered I/O for small requests

and non-buffered I/O for large requests. The rationale for this behavior is that some of the overheads of buffered I/O, especially the overheads of copying and cache management, are amplified for large requests. Also, disks handle large requests more efficiently than smaller ones, so large requests are better candidates for directly issuing to disk.

The parameter `discovered_direct_iosz` can be tuned up or down to increase or decrease, respectively, the request size threshold at which discovered direct I/O takes effect. For workloads where the benefits of caching outweigh the advantages of non-buffered transfers even for large requests, discovered direct I/O can be prevented by increasing `discovered_direct_iosz` to a high value. In general, there are a number of factors that can play a role in determining the best value of `discovered_direct_iosz` for a workload; request sizes in the workload, pattern of re-use of file blocks and page cache pressure are some of these factors. Since changes to `discovered_direct_iosz` can be easily made and reversed, it might be easiest to try a few different values and observe the effect on performance.

Concurrent I/O

Concurrent I/O in Veritas File System (VxFS) is a special type of direct I/O where the normal locking that VxFS performs is relaxed to allow concurrent writes to the same file. In modes other than Concurrent I/O, VxFS acquires a file-level lock in shared mode for a read request and in exclusive mode for a write request; this allows multiple read requests to the file to be serviced concurrently, but a write request prevents other read and write requests from being serviced concurrently. This form of locking is sometimes restrictive for databases which often issue concurrent, non-overlapping writes to the same file; since these writes are non-overlapping allowing them to proceed concurrently would not violate consistency. In Concurrent I/O, both reads and writes acquire the file lock in shared mode, which allows read and write requests to be serviced concurrently. For databases, this ensures that the concurrent writes that they issue are not serialized at the file system level. Concurrent I/O is meant to be used in cases where it is known that the file system will not receive conflicting writes or conflicting reads and writes concurrently; in practice, this mode is used with databases which are known to be well-behaved in this respect. Concurrent I/O is the recommended mode in VxFS for non-Oracle databases running transaction processing workloads; for Oracle databases, the ODM interface, which provides the same benefits and more, is recommended.

Concurrent I/O can be enabled for a whole file system using the `-o cio` mount option. Concurrent I/O can be enabled for a file descriptor by setting the `VX_CONCURRENT` advisory on the file descriptor using the `vxfsio` interface. Other than the difference in locking, Concurrent I/O operation is similar to direct I/O. Concurrent I/O also has the same alignment constraints as direct I/O.

About Veritas File System space allocation

The performance seen by an application can, to a large degree, depend on how well Veritas File System (VxFS) has been able to allocate space for the files used by the application. This section gives an overview of the relevant concepts and discusses administrative actions that can improve space allocation and, as a result, the file system performance.

A VxFS file system is created over a block storage device such as a hard disk. VxFS treats the underlying device as an array of blocks that it can use for storing file data written by applications and for storing its own metadata structures. An example of a metadata structure is the map that VxFS uses to keep track of free space, that is, blocks that have not yet been allocated to a file or metadata structure. The following list contains two basic concepts that are important in understanding VxFS space allocation:

- file system block

  This is the smallest unit of space allocation in VxFS. The file system block size is decided at the time of file system creation and cannot be changed later. The block size for a file system can be specified explicitly at the time of file system creation; permissible block sizes in VxFS are 1 KB, 2 KB, 4 KB and 8 KB. If the block size is not explicitly specified, VxFS picks the block size based on the size of the file system being created. A file system that is 1 terabyte or less is created with a block size of 1 KB; a filesystem greater than 1 terabyte in size is created with a block size of 8 KB.

- extent

  An extent is a contiguous region on the storage device consisting of one or more file system blocks. VxFS is an extent-based file system that allocates space to files in variable-sized extents, rather than in individual blocks. The information of which extents make up a file is part of the file's metadata and is stored in the file's inode and, if necessary, in additional metadata blocks. With extent-based allocation, when the file system is able to allocate space optimally, even a large file may have only one or a few extents allocated to it; in many cases, all the metadata needed to access the file fits in the inode itself. In contrast, file systems that use block-based allocation will need many metadata blocks in addition to the file inode in order to store the block addresses for a large file, and these metadata blocks must be retrieved when accessing the file.

When possible, VxFS tries to allocate space in large extents rather than in multiple smaller extents. Larger extents have the potential to improve performance for the following reasons:

- There are fewer extent addresses to store as part of the metadata for the file; there is less metadata to retrieve when accessing a file.

- A large application I/O request that spans multiple extents must be broken up and issued as smaller requests to the storage device. This will happen less often with large extents compared to smaller extents.
- Larger extents store more data contiguously compared to smaller extents, and therefore tend to improve performance for sequential accesses.

The allocation of space to a file can happen in different ways, but most frequently it happens as part of a write request when an application extends a file. When an application is sequentially extending a file in small chunks, VxFS typically allocates larger extents than is requested, in the anticipation that the allocated space will be used by future requests. If the application does not use the allocated space, VxFS reclaims unused space in a delayed fashion. VxFS starts with a small extent size on the first allocation and then doubles the extent size on each subsequent allocation, till a maximum size is reached. Once the maximum extent size is reached, further allocations continue to happen at the maximum size. This approach results in fewer allocations and larger allocated extent sizes. The following tunable parameters, which can be displayed and set using the `vxtunefs` command, govern this allocation mechanism:

initial_extent_size — This parameter gives the minimum size of the extent allocated on the first write to a file.

max_seqio_extent_size — This parameter gives the maximum extent size allocated.

Increasing the values of these parameters can result in larger allocated extents. However, the default values work well for most environments and are recommended.

Choosing the file system block size

The file system block size can be specified at the time of file system creation. Since Veritas File System (VxFS) uses extent-based allocation, the file system block size plays a less important role in VxFS compared to file systems that use block-based allocation, where a large block size helps reduce the number of block addresses that must be stored as part of the metadata of the file. The main concern in choosing the file system block size in VxFS is the efficiency of space utilization. For a file system that is expected to contain many small files, a small block size results in better utilization of storage space. To see why this is so, consider a hypothetical case where all files in a file system are less than a kilobyte. Since the file system block is the smallest unit of allocation, a large block size of 8 KB for this example would result in more than 7 KB of wasted space in each block. That is more than 87.5% wasted space. A block size of 1 KB results in much better space utilization in this example. The default block size picked by VxFS based on the size of the file system being created usually works quite well.

Online resizing and defragmentation

When a file system has been newly created, the free space is largely contiguous. As a file system is used, repeated allocation and freeing of space in extents of different sizes can result in the free space getting fragmented. When this happens, it becomes difficult for the file system to allocate space in large extents. In this state, the performance of the file system may degrade. The problems related to fragmentation are typically worse when the amount of free space in the file system is low.

There are two important tasks that an administrator can perform periodically to improve file system performance:

- Growing the file system. This increases the amount of free space available for new allocations and enables better allocations. As a general rule, when the amount of free space in a file system drops below 10%, it is a good idea to increase the size of the file system.
- Defragmentation. This reorganizes existing allocations to enable more efficient access.

VxFS supports resizing and defragmentation while the file system is mounted and in use. The `fsadm` command is used for both of these administrative tasks. Typically, an administrator starts by using the `fsadm` command to get a report of the current state of the file system to decide if resizing or defragmentation is necessary.

For information about the command options, guidelines for when these tasks should be performed, and examples that show actual use of the command, see the `fsadm_vxfs`(1M) manual page.

Defragmenting a file system involves reorganizing the data on the storage device by copying data from smaller extents into large extents. This activity typically has a high impact on performance, and should be scheduled at a time when it will cause minimal disruption. The `fsadm` command has an option that limits the time for which reorganization is allowed to run, which is useful in planning this task.

Chapter 5

Tuning reference for Veritas Volume Manager

This chapter includes the following topics:

About tuning Veritas Volume Manager

Veritas Volume Manager (VxVM) provides logical storage devices, called volumes, that appear to applications as physical devices (disks or LUNs), but overcome many of the space, reliability, and performance limitations of actual physical disks.

VxVM relies on certain daemons and kernel threads for its operations. Increasing the number of these threads can in some cases improve performance:

`vxconfigd` The VxVM configuration daemon that reads and updates configuration information stored on disks. The number of threads that the daemon uses can be set using the vxconfigd command. A larger number of threads allows more configuration reads to happen in parallel.

`vxiod` Kernel threads that perform some of the I/O operations in VxVM. The number of threads can be set using the `vxiod` utility. By default, the number of vxiod threads started is 16 for a system with 8 CPUs or less, and 2 times the number of CPUs for a system with more than 8 CPUs, up to a maximum of 64 threads. When the number of threads is set explicitly, it cannot be set to less than 1 or more than 64.

The following table contains some of the tunable parameters that affect general VxVM performance:

vol_maxio This is the maximum size of an I/O request that VxVM handles without breaking up the request. A request greater than vol_maxio in size is broken up and performed synchronously.

On AIX, Linux, and Solaris, the default value is 2048 blocks (1 MB).

For workloads with very large request sizes, vol_maxio can be increased to improve performance.

voliomem_chunk_size VxVM allocates and frees memory in chunks of this size. Increasing the chunk size can reduce the overhead of allocation. The benefit of tuning this parameter is usually small for most workloads.

On AIX, the default value is 64 KB.

On Linux, the default value is 32 KB.

On Solaris, the default value is 64 KB.

Commonly used VxVM layouts

A volume can be thought of as a linear array of blocks. The volume layout determines how these blocks are mapped to storage on underlying disks or LUNs. This section gives an overview of the most commonly used VxVM layouts and their performance characteristics.

See the *Veritas Volume Manager Administrator's Guide* for more information.

The following table contains the most common types of VxVM volumes based on layout:

concat volume	In the simplest and most typical case, a concat volume is built from storage from a single disk or LUN. That is, the underlying storage for the volume comes from a single contiguous region of a disk or LUN. In the more general case, the underlying storage for the volume can come from multiple disks. For example, the first half of the volume can be mapped to one disk and the second half to another; in this case, the volume is formed by concatenating the storage from the two disks. Except in the case where a concat volume is created over a LUN striped in the disk array, this layout is not suitable where high throughput is desired.
striped volume	The volume is divided into chunks called stripe units, which are then distributed across multiple disks or LUNs.
mirrored volume	Two copies of the data are maintained on different disks. While writing, both copies are updated. While reading, either copy can be read.
striped-mirror layout	This layout gives a volume the throughput benefits of striping, combined with the protection given by mirroring. This layout can be thought of as striped across a set of columns, each of which is mirrored.
mirrored-stripe layout	This layout also combines the benefits of striping and mirroring. It can be thought of as a mirrored layout, where each plex is striped across multiple columns.

Disk arrays also offer striping and mirroring features similar to VxVM. There are many different ways of combining striping and mirroring features in a disk array with similar features in VxVM to achieve desired throughput and redundancy characteristics. For example, a striped VxVM volume created over a set of LUNs that are mirrored in the disk array can be used to get performance and reliability similar to a striped-mirror VxVM volume.

Striped layouts

The striped, striped-mirror and mirrored-stripe layouts use striping to improve I/O performance. Striping improves performance in two ways:

- A single large request can be serviced in parallel by multiple disks resulting in better read and write bandwidth. To understand how, consider a volume striped over 4 disks--that is, number of columns is 4--with a 64 KB stripe unit size and an application that is issuing read requests of size 128 KB to the volume. Each read request, assuming the request is aligned on a 128 KB boundary would be serviced by reading 2 stripe units in parallel, each stripe unit from a different disk. In general, in a striped layout, large requests are

able to benefit from the aggregate bandwidth of multiple disks. This improves performance compared to a volume created over a single disk, where performance of large requests is limited by the bandwidth of a single disk.

- Concurrent, small requests can be serviced in parallel by the disks in the striped volume. For most workloads, VxVM sees a high degree of concurrency in requests. That is, the application or higher layer of the storage stack, such as VxFS or a database server, has multiple I/O requests to VxVM outstanding at any given time. Consider again the case of a volume striped over 4 disks with a 64 KB stripe unit size. If the application is issuing multiple 8 KB reads concurrently, 4 such read requests can generally be serviced in parallel. In comparison, when the volume is created over a single disk, the requests are serviced serially, limiting throughput and increasing response times.

There are two main parameters that must be chosen for a striped layout: the number of columns and the stripe unit size. As the number of columns increases, the maximum throughput of a striped layout also increases. The number of columns is usually limited by provisioning constraints. There is also the additional concern of reliability: As the number of columns increases, the probability of disk failure also increases; due to this concern, striping is usually combined with mirroring to improve reliability.

You should choose the stripe unit size based on the workload characteristics. For workloads with small, concurrent requests, a smaller stripe unit size works better. Workloads with large, concurrent requests can benefit from a larger stripe unit size. A stripe unit size that is too small can result in inefficient disk usage, since disk seek overhead calls for a large I/O size for efficiency. On the other hand, a stripe unit size that is too large limits the number of disks that can serve medium and large requests in parallel. The default stripe unit size in VxVM is 64 KB, and generally works well for a range of workloads. Symantec does not recommend reducing the default stripe unit size of 64 KB. As mentioned above, for workloads with very large, concurrent requests, increasing the stripe unit size can improve performance.

Mirrored layouts

The mirrored, striped-mirror and mirrored-stripe layouts store mirrors or copies of each block in the volume, improving reliability in the face of disk failures.

With mirrored volumes, Veritas Volume Manager (VxVM) allows the read policy of the volume to be set to indicate which of the copies should be chosen for read. In the typical case where each plex of the mirrored volume has the same layout, VxVM reads each mirror in a round-robin manner.

The striped-mirror and mirrored-stripe layout offer similar performance and redundancy for normal operation. The stripe-mirror layout is usually preferred

for its ability to limit the impact of a single disk failure to a single column. VxVM gives the flexibility of creating complex layouts by combining and layering these layouts in different ways. For simplicity of analysis, a simple layout is generally preferred.

To understand the performance impact of mirroring, consider two volumes, one a striped layout and other a striped mirror layout, created using the same number of disks. Since VxVM by default allows reads to be serviced from either mirror, read performance is about the same for both layouts. However, the stripe-mirror has higher overhead on writes because two copies need to be written, compared to one for the striped layout. In addition, mirroring is usually used with dirty region logging, which adds a small overhead for write. Combined these two factors reduce the write performance of the striped-mirror volume compared to the striped volume.

Tunable parameters for mirrored volumes

The tunable parameters for Dirty Region Logging, which is generally enabled on mirrored volumes, are discussed separately. In addition the following tunable parameter is relevant for mirrored volumes:

`voliomem_maxpool_sz` This parameter specifies a limit on the memory requested by Veritas Volume Manager (VxVM) for internal purposes. For a mirrored volume, a write request that is greater than this size is broken up and performed in chunks of `voliomem_maxpool_sz`. The default value is 5% of memory on the system, up to a maximum of 128 MB.

Online re-layout

The initial layout for a volume is often made without a good understanding of the workload. Once the volume is in use, it may turn out that the chosen layout does not provide adequate performance. In such cases, the online re-layout feature of Veritas Volume Manager (VxVM) can be used to change to a different layout altogether, such as from concat to striped, or to change the parameters of the same layout, such as increase the number of columns in a striped volume. Online re-layout causes some performance degradation while the re-layout is in progress.

Certain transformations might not be permitted when using online re-layout.

See the *Veritas Volume Manager Administrator's Guide* for more information.

Dirty Region Logging for mirrored volumes

The Dirty Region Logging (DRL) feature of Veritas Volume Manager (VxVM) speeds up recovery of mirrored volumes following a system crash. When the system crashes while writes are in progress to a mirrored volume, the mirrors can become out of sync. One way to recover from such a crash is to resynchronize the entire volume completely, which can take a long time with large volumes. DRL reduces the resynchronization time for mirrored volumes by keeping track of portions of the volume that may be out-of-sync and then syncing only those portions.

The DRL feature works by logically dividing the volume into fixed-size regions and maintaining a map on disk of the status of these regions. A region can be marked as dirty, meaning that the region is potentially not in sync on all mirrors, or clean in the DRL map. When a write is received for a block or blocks within a region, VxVM ensures that the region is marked dirty in the DRL map on disk before the write is allowed to proceed. If the system crashes before the write is complete, the DRL map tells VxVM that the region must be resynchronized as part of the recovery process. Once a write is complete, the corresponding region can be marked clean in the DRL map if there are no other writes active on that region. The recovery time is thus determined by the number of dirty regions in the DRL map and the region size, rather than by the size of the volume.

VxVM uses various optimizations to ensure that the number of writes required to maintain correct DRL bitmap status is minimized. The following list includes some of these optmizations:

- When a write completes, VxVM typically delays marking the corresponding region as clean in the DRL map. It is not incorrect to have a region that is in-sync on all mirrors marked as dirty in the DRL map. However, it is incorrect to mark a region that is out-of-sync as clean. A region marked dirty in the DRL map is potentially, but not necessarily, out-of-sync; a region marked clean is in-sync. In most cases, DRL writes to mark regions as clean can be delayed and combined with other DRL writes.
- If a new write is received for a block and the region corresponding to the block is already marked dirty in the DRL map, there is no need for a DRL update. Except in the case where the workload has highly random requests, it is usually the case that writes show some locality: when write is received for a block, it is likely that other blocks in the same region have been written to recently. As a result many writes to the volume do not require a DRL update. This is another reason why it makes sense to delay marking regions as clean in the DRL map. Also, the larger the region size for the DRL, the higher the probability that a new write will find the region already marked dirty in the DRL. Hence, the larger the region size, the better this optimization works.

- A single write to the DRL map on disk is sufficient to mark multiple regions as dirty. Writes received concurrently by VxVM do not typically make multiple updates to the DRL, even if the writes are to different regions.

As a result of these and other optimizations, VxVM is able to implement DRL functionality with low performance overhead. The DRL overhead can be further reduced by appropriate tuning.

The DRL map is stored persistently on disk and storage must be configured for storing the map. You can store the map in one of two ways:

- Use a traditional DRL that uses a log sub-disk to store the DRL map.
- Use a version 20 DCO volume that can store the DRL map and also region maps for instant snapshots

Performance tuning for the two cases is similar, but minor differences exist.

Tuning traditional Dirty Region Logging

Traditional Dirty Region Logging (DRL) uses a log sub-disk to store the bitmaps necessary to implement DRL functionality. Multiple options exist for configuring the log sub-disk.

See the *Veritas Volume Manager Administrator's Guide* for more information.

In the simplest and most common case, use the `vxassist` command, either at volume creation or later, to associate a dedicated log plex with the mirrored volume and turn on DRL functionality; the log plex contains the log sub-disk that stores the DRL bitmaps.

The region size for the DRL has a large impact on performance and speed of recovery. For traditional DRL, the region size is not specified explicitly. Instead, the size is computed as follows:

- If you specify the log length explicitly at the time that you create the log sub-disk, then Veritas Volume Manager (VxVM) uses the log length to calculate the region size such that the DRL bitmap fits in the log. However, if the region size calculated based on the log length is less than the minimum region size allowed for traditional DRL, then the minimum region size is used. The minimum region size is 1024 sectors (512 KB). Specifying a large log length reduces DRL region size; this favors speedier recovery, but reduces performance.
- If you do not specify the log length at the time of that you create the log sub-disk, VxVM picks a default log length and then calculates the region size such that the DRL map fits in the log. The minimum region size also applies in this case. The default log length picked by VxVM is typically small. For large

volumes, this results in a very large region size which gives better performance, but increases recovery times.

The following VxVM tunable parameters are also relevant for tuning traditional DRL performance:

`voldrl_max_drtregs`	This parameter controls the maximum number of dirty regions that VxVM allows at a time. This is a system parameter; the number of dirty regions in all of the volumes on a system combined are not allowed to exceed `voldrl_max_drtregs`. Increasing the value of this tunable parameter improves performance, although the benefit seen will depend on the nature of the workload. Performance improves because a larger value for this parameter gives VxVM more flexibility in optimizing DRL updates. Cleaning of dirty regions can be delayed, which can help avoid DRL updates when new writes to the same regions are received. The default value of this parameter is 2048. Increasing the value of this parameters can increase the recovery time.
`voldrl_volumemax_drtregs`	This tunable parameter is a per-volume limit on dirty regions for a mirrored volume using traditional DRL. For heavily-used volumes, the value of this parameter can be increased to improve performance. The default value of this parameter is 256. The maximum value is the value of the parameter `voldrl_max_drtregs`.
`voldrl_max_seq_dirty`	This tunable parameter applies to sequential DRL. The default value of this parameter is 3. See "Sequential Dirty Region Logging" on page 97.

Tuning Dirty Region Logging in a version 20 DCO volume

A version 20 DCO volume accommodates the region bitmaps for both DRL and instant snapshots. In the case of version 20 DCO, region size can be specified explicitly. By default, the region size is 64 KB. When using only the DRL functionality, the region size should generally be chosen to be a higher value. When using the same DCO volume for DRL and instant snapshots, there are additional considerations in choosing region size.

See "Using a version 20 DCO volume for both Dirty Region Logging and instant snapshots" on page 102.

The following tunable parameters are relevant for tuning DRL performance with a version 20 DCO volume:

`voldrl_max_drtregs`	The behavior of this tunable parameter is the same for traditional DRL and DRL in a version 20 DCO volume. This parameter controls the maximum number of dirty regions that VxVM allows at a time. This is a system parameter: the number of dirty regions in all of the volumes on a system combined are not allowed to exceed `voldrl_max_drtregs`. Increasing the value of this tunable parameter improves performance, although the benefit seen depends on the nature of the workload. Performance improves because a larger value for this parameter gives VxVM more flexibility in optimizing DRL updates. A larger value allows cleaning of dirty regions to be delayed, which can help avoid DRL updates when new writes to the same regions are received. The default value of this parameter is 2048. Increasing the value of this parameter can increase the recovery time.
`voldrl_volumemax_drtregs_20`	This tunable parameter is a per-volume limit on dirty regions for a mirrored volume using DRL with a version 20 DCO volume. For heavily used volumes, the value of this parameter can be increased to improve performance. The default value of this parameter is 1024. The maximum value is the value of the parameter `voldrl_max_drtregs`.
`voldrl_max_seq_dirty`	This tunable parameter applies to sequential DRL. See "Sequential Dirty Region Logging" on page 97.
`volpagemod_max_memsz`	This is maximum amount of memory that VxVM uses for caching bitmaps and other metadata in version 20 DCO volumes. The default is 6 MB. The valid range is 0 to 50% of the physical memory. Symantec recommends increasing the value of this tunable parameter when you have large volumes and a small region size.

Recovery time considerations for DRL in a version 20 DCO volume are the same as for traditional DRL. Recovery time increases when the region size increases and when the number of dirty regions increases.

Sequential Dirty Region Logging

Veritas Volume Manager (VxVM) provides an optimized form of Dirty Region Logging (DRL), called sequential DRL, for use with volumes where the write pattern

is known to be sequential. Sequential DRL can be enabled with traditional DRL--that is, DRL using a log sub-disk--or with DRL in a version 20 DCO volume. In sequential DRL, when a new region is written for the first time, the region and a few other regions ahead of the region sequentially are marked dirty. At the same time, all other regions are marked clean.

The VxVM tunable parameter `voldrl_max_seq_dirty` specifies how many regions are marked dirty in each DRL update. The default value of `voldrl_max_seq_dirty` is 3. Consider a mirrored volume with a sequential DRL and region size of 512 KB to which an application is writing sequentially in 8 KB requests. In this case, each DRL write dirties 3 regions of 512 KB each. The first write to a region in the volume triggers a DRL update to mark that region and 2 other regions dirty. The next 63 writes to the volume (512 KB region size / 8 KB write size = 64) are to the same region since the writes are sequential. The write after that (65th write) is to a new region, but one that has already been marked dirty by the last DRL update. In this example, the overhead of DRL writes is 1 DRL write for every 192 (512 KB * 3 / 8 KB) writes to the volume, which shows that the sequential DRL overhead can be very low. Since the number of dirty regions at any time is very low, recovery is also fast. The DRL overhead can be further reduced by increasing the region size and by increasing the value of `voldrl_max_seq_dirty`. As with other types of DRL, a larger region size and more dirty regions increase recovery time.

Instant snapshots

An instant snapshot in Veritas Volume Manager (VxVM) is a point-in-time copy of data in a volume. The command to create an instant snapshot typically completes in a matter of seconds. Once the command completes, the instant snapshot is available for use as a point-in-time copy of the original volume. Even if the original volume is updated, reads to the snapshot return data that existed in the original volume at the point in time when the snapshot was created. VxVM implements instant snapshots using a copy-on-write technique. When a block in the original volume is updated for the first time after snapshot creation, VxVM makes a copy of the old data in the block before applying the new update to the block.

There are two types of instant snapshots available in VxVM: full-sized instant snapshots (or simply, full instant snapshots) and space-optimized instant snapshots (SO snapshots). A full instant snapshot requires that space equal to the size of the original volume be dedicated for the snapshot. On the other hand, SO snapshots are typically allocated only a fraction of the space of the original volume.

Full instant snapshots

Since a full-size instant snapshot has space equal to that of the original volume, it can hold the point-in-time copy of every block in the original volume. Data gets populated in the snapshot volume over time due to copy-on-write and background syncing. As long as the data in the snapshot volume has not been completely populated, Veritas Volume Manager (VxVM) serves read requests to the snapshot volume from either the snapshot volume or the original volume, as appropriate. Once the full instant snapshot is completely populated, all reads to the snapshot get serviced from the snapshot volume itself without any request load on the original volume.

VxVM implements instant snapshots on a volume by dividing the volume into fixed-size regions and tracking writes to these regions. On the first write to a region after a snapshot has been created, VxVM copies data from the region in the original volume to the snapshot volume. This copy-on-write mechanism ensures that the snapshot volume retains the data corresponding to the point-in-time when the data was created. The write that triggers a copy-on-write may be much smaller than the region size itself. For example, if the region size is 64 KB and the write issued by the application is 8 KB, VxVM copies a whole 64 KB region to the snapshot volume before applying the 8 KB write to the original volume.

Region size for a full instant snapshot

The region size has a significant impact on performance. A larger region size causes higher latency for the first write to a region after snapshot creation, since that write triggers copy-on-write. However, any subsequent write to the same region, whether to the same block or to a different block in the same region, does not incur the copy-on-write performance penalty. For workloads with mostly random writes, a small region size close to the default of 64 KB works well, especially if the system has a storage bottleneck. For workloads with good write locality, meaning the workload has many writes to the same regions, a region size larger than the default size can give better performance. Generally, the default region size works well for full-size instant snapshots for a range of workloads.

Configuring a version 20 DCO volume for a full instant snapshot

To use the instant snapshot functionality, you must create a version 20 DCO volume and attach the DCO volume to the volume for which you want to use snapshots. The DCO volume contains space for the region maps needed to track write activity. A key parameter when creating a version 20 DCO volume is the region size. By default, the region size is 64 KB, but the size can be a power of 2, with 16 KB as the minimum possible value.

The following tunable parameter has an impact on performance of instant snapshot functionality:

`volpagemod_max_memsz` This is maximum amount of memory that Veritas Volume Manager (VxVM) uses for caching bitmaps and other metadata in version 20 DCO volumes. The default is 6 MB. The valid range is 0 to 50% of the physical memory. Symantec recommends increasing the value of this tunable parameter when you have large volumes and a small region size.

Creation time for full instant snapshot

When you issue the `vxsnap make` command to create a full instant snapshot on a volume, applications might be writing to the volume at the same time. To get consistent metadata, Veritas Volume Manager (VxVM) typically freezes new writes and drains existing writes for short periods during snapshot creation. If the snapshot is taken during a phase where there is heavy write activity on the volume, the `vxsnap make` command generally takes longer to complete. The snapshot creation time increases for a larger region size.

Background syncing for full-sized instant snapshots

In addition to the copy-on-write mechanism that populates data in the snapshot volume, Veritas Volume Manager (VxVM) has a background syncing mechanism to copy regions from the original volume to the snapshot volume. The reading of data from the original volume as part of background syncing causes a performance penalty for accesses to the original volume, over and above the performance penalty of copy-on-write. At the time of snapshot creation, background syncing can be turned off. Whether or not that is desirable depends on the intended use of the snapshot: without background syncing, the data in the snapshot volume might not be fully populated.

Background syncing can be tuned by specifying two parameters:

- I/O size used during background copying of data.
- Delay between successive background copying.

Performance impact of a full instant snapshot on the original volume

A full instant snapshot can have a negative impact on the performance of the original volume, although the impact is usually small. During snapshot creation, there is a brief period while the writes to the volume are frozen, as mentioned

above. After that the performance impact on the original volume is due to the cumulative effect of the following factors:

- Copy-on-write overhead while the snapshot is not fully populated. This is high soon after the snapshot has been created, when almost no regions have been populated in the snapshot; it tapers off as more regions are populated in the full instant snapshot due to copy-on-write and background syncing.
- Background syncing.
- Reads on the snapshot while it is not fully populated. Initially, many of the reads will be serviced from original volume. As more data gets populated in the full instant snapshot, fewer reads need to serviced from the original volume.

Space optimized instant snapshots

A space optimized instant snapshot (SO snapshot) does not require that space equal to the original volume should be reserved for the snapshot. Instead, an SO snapshot uses a cache object mechanism that works with a fraction of the space of the original volume and stores only modified regions within this cache object. There are similarities in the way full-sized instant snapshots and SO snapshots are implemented: both rely on region maps in a version 20 DCO volume to keep track of writes to regions and both employ a copy-on-write mechanism.

There are two main considerations in choosing the region size for an SO snapshot:

- Performance: The region size considerations for an SO snapshot are similar to those for a full instant snapshot from a performance standpoint. A larger region size causes a larger delay for writes that trigger copy-on-write.
- Space utilization: A larger region size causes more waste of space because the whole region, including blocks that have not been modified, must be stored in the cache object; considering that SO snapshots are used with space saving in mind, a large region size works against that principle.

Symantec recommends that you use the default region size (64 KB) or a region size close to that for SO snapshots.

Performance comparison of full-sized and spaced optimized instant snapshots

Copy-on-writes in the case of a space optimized instant snapshot (SO snapshot) require a write to the cache object, and reads from SO snapshot require a read on the cache object. A cache object read or write has a higher cost than read or write

on a normal volume. The main advantage of SO snapshots is the space saving. Full instant snapshots perform better in all of the following cases:

- Snapshot creation time, and the performance impact on the original volume because of writes being frozen during snapshot creation, is lower with full instant snapshots compared to SO snapshots.
- Performance of applications that write to the original volume, is better with full instant snapshot because writing a region to a full snapshot volume (copy-on-write) is more efficient than writing the region to a cache object.
- Performance of applications, such as backups that read from the snapshot volume, tend to be better with full instant snapshots than with SO snapshots, because reading from a cache object has lower performance.
- The performance impact on the original volume when the snapshot is read quickly reduces for full instant snapshots as background syncing populates data in the snapshot, which reduces the number of reads that must be serviced from the original volume. With SO snapshots, any region that is not populated in the cache object as a result of copy-on-write must be read from the original volume.

These benefits are only from a performance standpoint.

For more information about the differences between full-sized instant snapshots and spaced optimized instant snapshots, see the *Veritas Volume Manager Administrator's Guide*.

Using a version 20 DCO volume for both Dirty Region Logging and instant snapshots

One of the conveniences of a version 20 DCO volume is that the same DCO volume can be used for Dirty Region Logging (DRL) and for instant snapshots. A large region size is generally preferable for DRL to reduce the overhead of DRL writes. When a version 20 DCO volume is used just for DRL (no instant snapshots) a large region size is recommended. For instant snapshots, on the other hand, a moderate region size (close to the default of 64 KB) generally works better. If the workload has a large percentage of writes, high degree of concurrency and random nature, DRL overhead can be significant with a small region size. For such workloads, if instant snapshot and DRL are both desired on the same volume, it might be necessary to use a region size that is larger than the default of 64 KB as a compromise between what is good for DRL and what is good for instant snapshots.

Chapter 6

Tuning reference for Dynamic Multi-Pathing

This chapter includes the following topics:

About Dynamic Multi-Pathing in the data center

Dynamic Multi-Pathing (DMP) is an I/O multi-pathing layer that simplifies the task of managing storage performance and availability in complex environments. A typical data center environment has a large number of LUNs configured in disk arrays and made visible to servers over a storage area network; multiple HBAs on each server, a storage area network with redundancy, and multi-controller disk arrays together provide multiple paths to the LUNs that can be used to improve availability and performance. All this makes for a complex environment. The performance implications of configuration decisions are especially difficult to predict in this environment because of accesses to shared disk arrays from multiple servers, variations in applications' load levels, and failure and recovery of components that cause changes in request routing patterns.

The basic task of a multi-pathing layer is to exploit the multiple paths provided by the hardware to improve availability and performance transparently for applications: if an I/O fails on one path due to a fault in a component on the path, the multi-pathing layer can issue the I/O on another path, thus improving availability. Also, different concurrent I/Os can be sent down different paths in parallel to improve performance. DMP provides advanced capabilities that go beyond basic multi-pathing functionality; these capabilities are essential for ensuring good I/O performance and availability in today's complex data center environments. For example, DMP has load balancing policies that can adapt the load distribution to changes in the performance characteristics of the environment, in addition to distributing the I/O load across multiple paths. As another example, DMP in many cases can identify and avoid faulty paths pro-actively, thus avoiding the performance penalty of recovering from errors it, in addition to recovering from an I/O error on a path and issuing the I/O on another path.

Administrators can maintain overall control over operations through policies and tunable parameters while DMP dynamically adapts to the operating conditions within these constraints. The features provided by DMP considerably reduce the complexity with which administrators must deal.

About tuning Dynamic Multi-Pathing

Dynamic Multi-Pathing (DMP) operation, and tuning DMP operation from a performance standpoint, can be thought of in terms of the following areas:

- Device discovery
  Device discovery does the following things:
  - Uniquely identifies LUNs on multiple paths
  - Uniquely identifies LUNs into an enclosure
  - Identifies the model and type of LUN, such as A/A or ALUA, and applies load balancing policies as predefined in the Array Support Libraries (ASLs)
- I/O load balancing
  DMP attempts to improve performance of I/Os issued on a DMP meta-device by balancing the I/Os on the multiple I/O paths for the meta-device. The way the load is balanced on the set of active paths for a meta-device is determined primarily by the DMP I/O policy in effect. There are many different I/O policies in DMP that an administrator can choose depending on the workload and operating environment; the default I/O policy generally works well for a range of workloads and operating environments.
- Error detection and handling

To deliver the higher availability and performance made possible by multiple I/O paths, DMP must handle a variety of errors appropriately. Much of DMP's error handling behavior, such as timeout intervals and the number of retries in different error scenarios, can be modified using tunable parameters.

Warning: In most cases, the default values of these parameters are appropriate and they should be changed with caution.

- Path analysis
  The state of an I/O path (healthy or failed) can change due to errors in various components on the path and the components' subsequent recovery. The more up-to-date the information that DMP has on the state of the I/O paths, the better its I/O scheduling decisions can be. Path analysis in DMP keeps the state of paths reasonably up-to-date to enable better scheduling decisions. The path analysis behavior of DMP can be tuned to favor improved state information at the cost of extra CPU cycles and I/O, or the behavior can be tuned to favor lower CPU and I/O overhead, but with less up-to-date state information on I/O paths. DMP has a number of tunable parameters that determine aspects of path analysis. The default settings of these parameters provides behavior that is generally appropriate for most environments.

DMP can discover Fibre Channel events and monitor for Fibre Channel events. When a Fibre Channel event is received, DMP can, based on the discovered topology, identify the I/O paths that may be impacted by the event. DMP uses SCSI passthru interfaces for fast error handling capability to get detailed error information in the case of I/O errors so that DMP can handle the errors appropriately. DMP can group I/O paths into Subpath Failover Groups (SFGs) and make collective decisions for a whole group. This feature is particularly useful in avoiding multiple I/O errors in cases where a fault affects a whole group of paths.

See the *Dynamic Multi-Pathing Administrator's Guide*.

You can display and change online the tunable parameters that control DMP operation using the `vxdmpadm` command. Use the `vxdmpadm gettune` command to list the parameters and their values, and the `vxdmpadm settune` command to change the parameter values.

```
# vxdmpadm gettune all
             Tunable                Current Value  Default Value
------------------------------    -------------  -------------
dmp_cache_open                          on              on
dmp_daemon_count                        10              10
dmp_delayq_interval                     15              15
dmp_enable_restore                      on              on
```

dmp_fast_recovery	off	on
dmp_health_time	60	60
dmp_log_level	1	1
dmp_low_impact_probe	on	on
dmp_lun_retry_timeout	0	0
dmp_path_age	300	300
dmp_pathswitch_blks_shift	9	9
dmp_probe_idle_lun	on	on

These tunable parameters are system-wide, such that the parameters control how the DMP module operates in general for all storage devices. DMP also allows some aspects of its operation to be customized for an enclosure, array, or array-type by setting attributes for the enclosure, array, or array-type using the `vxdmpadm setattr` command. Where it is possible to do so, these provide fine-grained control over DMP operation.

DMP has a pool of kernel threads for handling error analysis, path restoration, and other administrative tasks. The number of these threads can be controlled using the following tunable parameter:

`dmp_daemon_count`	This tunable parameter specifies the number of kernel threads. The default value is 10, and Symantec recommends that you do not reduce the value of this parameter. For high-end servers with more than 20 CPU cores, Symantec recommends that you set this tunable parameter to half the number of CPU cores in the server.

DMP can be configured to gather statistics for the I/Os that it processes. Statistics gathering within DMP can be enabled using the `vxdmpadm iostat start` command. When statistics gathering is enabled, DMP maintains statistics for I/Os that have completed and maintains timing information for pending I/Os. The idle LUN probing feature and timeout-based I/O throttling feature only take effect when statistics gathering has been enabled. Gathered statistics can be displayed using the `vxdmpadm iostat show` command.

For more information on gathering and displaying I/O statistics, see the *Dynamic Multi-Pathing Administrator's Guide*.

When enabled, statistics gathering can have a small impact on performance. This impact can be controlled without turning off the feature by using the following tunable parameter:

`dmp_stat_interval`	This is the interval at which DMP processes statistics. Increasing the interval reduces the overhead of statistics gathering. The default and minimum value is 1 second. This value can be increased to reduce the performance impact of statistics gathering. However, a larger interval can result in buffer overflows and hence affect the accuracy of statistics.

Dynamic Multi-Pathing device discovery

Device discovery in Dynamic Multi-Pathing (DMP) is the process through which DMP identifies the I/O paths corresponding to each LUN. In device discovery, DMP examines disks discovered by the operating system and identifies the disks that represent different paths to the same LUN. For each LUN, DMP creates a new device in the operating system device tree; this device is the DMP meta-device for the LUN and can be used by higher layers of the storage stack to perform I/O on the LUN. For each I/O on a meta-device, DMP issues the I/O on one of the paths for the meta-device.

Device discovery in DMP is aided by array-specific Array Support Libraries (ASLs). ASLs are generally installed as part of Veritas Storage Foundation (SF).

For information on ensuring that the latest ASLs are installed, see the *Dynamic Multi-Pathing Installation Guide*.

The following tunable parameter optimizes the discovery process:

`dmp_cache_open`	The device discovery layer might need to send multiple requests to each device as part of the discovery process. When `dmp_cache_open` is set to **on**, the first open on a device by the ASL is cached and subsequent accesses can use the cached handle. Setting this parameter to **off** disables caching during device discovery and can slow the discovery process. The default and recommended value of this parameter is **on**.

Dynamic Multi-Pathing I/O load balancing

Dynamic Multi-Pathing (DMP) balances I/Os issued on a meta-device across the multiple I/O paths for the meta-device. A DMP meta-device typically corresponds to a LUN. Some paths might be excluded from the load balancing process as specified below:

- Paths that have been disabled by an administrator and paths discovered by DMP to have failed are not used.
- Based on the array type, some paths may be excluded. For example, on an active-passive array, only the primary paths are used during normal operation; other paths are used only after a failover.
- I/O throttling can cause some paths to be excluded from consideration temporarily.

The way load balancing is performed in DMP is determined by the I/O policy in effect. DMP provides a number of different I/O policies. By default, the `minimumq` I/O policy is used for all meta-devices, but the I/O policy can be set for an enclosure, array, or array-type using the `vxdmpadm setattr` command.

For more information on specifying the I/O policy, see the *Dynamic Multi-Pathing Administrator's Guide*.

Dynamic Multi-Pathing default I/O policy

The default I/O policy in Dynamic Multi-Pathing (DMP) is the `minimumq` policy. With this policy, when a new I/O is received for a meta-device, DMP schedules it on the path for the meta-device that has the minimum number of requests pending on it. Even though it is a simple criterion, the number of pending requests succinctly captures significant information about path performance, and the use of this criterion gives the `minimumq` policy the following beneficial characteristics:

- When the multiple paths for a meta-device are all performing comparably, `minimumq` distributes I/O load evenly across the paths at high load.
- When there is a difference in performance among the paths, better performing paths automatically get a higher proportion of the I/O load. These paths service requests faster and reduce the number of pending requests faster, and hence become eligible for more new I/Os.
- When there is a change in the relative performance of the I/O paths, `minimumq` quickly adapts. If a path that had been performing well degrades in performance, the queue of requests on the path grows and `minimumq` automatically diverts most new I/O requests to other paths. This ability to adapt to changes in path performance is important because, in most environments, the performance delivered by paths changes over time. Errors in storage network components, which can cause re-routing of requests and create hotspots, changes in applications' access patterns and load levels, and accesses from multiple servers to the same arrays can all cause slowing down of some paths compared to others.

While all DMP I/O policies can handle the failure of one or more paths, not all are able to adapt well when the performance characteristics of active paths change significantly; the ability to do this is a distinct advantage that `minimumq` has over many other policies. The `minimumq` policy has been seen to work as well as or better than other policies for a range of workloads and operating conditions. Symantec recommends that you use the `minimumq` I/O policy in DMP.

Optimizing disk array cache usage with the balanced policy

The `balanced` I/O policy in DMP is designed to balance the I/O load across the available paths while optimizing controller cache usage in the underlying disk arrays. DMP is frequently used with high-end, active-active (A-A) disk arrays with large caches in the disk array controllers. With an A-A array, DMP can send an I/O request on any path, to any of the controllers, unlike with an active-passive array where I/Os are sent to the secondary controller only after a failover. However, if an I/O request for a particular block is sent to one controller of an A-A array, and a later I/O request for the same block is sent a different controller, the block will likely reside in the caches of both controllers. The `balanced` I/O policy tries to avoid this cache duplication in the disk array by a careful mapping of blocks to paths. Requests for a particular block are always sent on the same path in this policy, but because different blocks map to different paths, load balancing is still achieved. In case of path failures, the mapping of blocks to paths is re-adjusted. The actual performance benefit from the `balanced` policy depends on many factors, including the cache management techniques of the disk array and nature of the workload. In those cases where the disk array caches are under pressure, the `balanced` I/O policy is a possible option for improving performance through optimized caching.

In the `balanced` I/O policy, each I/O request is mapped to a path based on the starting address of the I/O request. The mapping is based on a partition size, which can be specified when configuring the policy using the `vxdmpadm setattr` command. The mapping can be described as follows:

- The storage space of the LUN (a disk or LUN can be thought of as a linear array of bytes) can be thought of as being divided into partitions, the size of each being the partition size.
- When the starting address of a request is divided by the partition size, you get the partition to which the starting address belongs.
- Each partition is mapped to an I/O path in a rotating fashion. For example, if there are 4 I/O paths, partition 0 is mapped to path 0, partition 1 to path 1, partition 2 to path 2 and partition 3 to path 3; partition 4 is mapped to path 0 again.

- More formally, the I/O path on which an I/O request is to be sent is determined as follows: the starting address of the request is divided by the partition size to give the partition number; the partition number modulo the number of paths gives the path number on which the request is to be sent.

When the partition size for the `balanced` policy is not specified, the default partition size takes effect. The default partition size is the governed by the following tunable parameter:

`dmp_pathswitch_blks_shift` This tunable parameter specifies the default partition size that applies in those cases where the partition size is not specified while configuring the `balanced` I/O policy for an enclosure, array, or array-type using the `vxdmpadm setattr` command. This tunable parameter is only relevant with the `balanced` I/O policy, not with other I/O policies. The value of this parameter is expressed as the integer exponent of a power of 2. The default value of this parameter is 9, which means the default partition size is 29; that is, 512 blocks or 256k. Increasing the value of this parameter by 1, doubles the default partition size. If the value of this parameter is set too high, it can result in uneven load distribution on the paths. If the application I/O activity is localized to an address range of the LUN and the partition size is too large, some of the paths may not get any I/O requests. If the value of this parameter is small relative to the I/O request size, many requests may span partitions, which is not desirable.

When the partition size is specified explicitly in the `vxdmpadm setattr` command, the value is rounded down to the nearest power of 2 and interpreted as the number of blocks in the partition. For example, if the partition size is specified as 1200, the partition size used will be 1024 blocks or 512k. If the partition size is specified as 0, the default partition size, which is based on the value of `dmp_pathswitch_blks_shift`, is used.

Since the `balanced` I/O policy tries to spread the I/O load equally among available I/O paths, it works well when the paths have roughly the same performance; it does not adapt well in those cases where some paths are performing poorly compared to others due to storage network problems or other issues. This is a drawback of the `balanced` policy compared to the default `minimumq` policy. The performance benefit from changing the default I/O policy to the `balanced` I/O policy varies depending on the workload and the cache management techniques of the disk array for which the change is made. In many cases, the performance benefit from the caching optimization might be small, giving `minimumq` an edge

over the `balanced` policy. For these reasons, the `minimumq` policy is the default even for A-A arrays.

Dynamic Multi-Pathing I/O policies

The following I/O policies are available in Dynamic Multi-Pathing (DMP):

- `adaptive`

  In this policy, DMP dynamically calculates recent path performance and assigns priorities to paths based on their performance. Specifically, DMP calculates throughput, or bytes delivered per second. Paths that deliver better performance are assigned a higher priority. I/Os are then routed so that higher priority paths get a greater share of the I/O load. As with the `minimumq` policy, this policy is able to adapt dynamically to changes in the storage environment that affect path performance. The bookkeeping overhead for this policy, which is the overhead involved in calculating path priorities and scheduling I/Os in proportion to priority, tends to be higher than the overhead for `minimumq`.

- `balanced`

  This policy attempts to optimize disk array cache usage while balancing the I/O load on the available paths. The tunable parameter `dmp_pathswitch_blks_shift` is used with this policy.

  See "Optimizing disk array cache usage with the balanced policy" on page 109.

- `minimumq`

  This is the default and recommended I/O policy in DMP.

  See "Dynamic Multi-Pathing default I/O policy" on page 108.

- `priority`

  In this policy, administrators can manually assign priorities to paths and DMP will distribute I/Os among the paths proportionally based on their priority. The priority is an integer value; a higher value indicates that the path should be given a larger share of the I/O load. As an example, if there are two paths with priorities 1 and 2, the path with priority 2 will get two-thirds of the I/Os, while the other one will get a third. This policy may be useful in certain circumstances where administrators want to carefully control the I/O flow on paths. Since the policy is based on a static division of I/O load, it is difficult to employ this policy well in complex environments where unpredictable changes in path performance is common.

- `round-robin`

  In this policy, I/Os are sent down different paths in a randomized fashion so that load is spread evenly across all paths. The overhead of path selection in this policy is less compared to `minimumq`.

The `minimumq` policy needs to sort available paths by queue length; `round-robin` is simpler. This policy works well when the performance of the active I/O paths are about the same, but it does not adapt as well as `minimumq` to changes that may create imbalances in the performance of paths.

- `singleactive`

  In this policy, only one of the available paths is used for I/Os. If the active path fails, one of the other paths is made active. That is, this policy uses multiple paths only for high availability, not for load balancing. Only an outright path failure causes a new path to be chosen as the active path. In cases where the performance delivered by the active path drops, such as because of problems in the storage network, the policy continues to use the same path even though other paths may be capable of delivering better performance.

Dynamic Multi-Pathing I/O throttling

I/O throttling is a mechanism by which Dynamic Multi-Pathing (DMP) temporarily stops issuing I/Os to paths that appear to be either overloaded or underperforming. There is a default I/O throttling mechanism in DMP based on the number of requests queued on a path and it is controlled by the following tunable parameter:

`dmp_queue_depth`	When the number of requests queued on a path reaches the value of this tunable parameter, DMP does not schedule new I/Os on the path until one or more of the pending requests complete. The default value of `dmp_queue_depth` is 32 on Solaris, Linux, and AIX. The default value is appropriate for most environments and tuning this parameter is not recommended.

Administrators can also configure throttling based on request response times. An enclosure, array, or array type can be configured so that I/Os to a path are throttled when the time for which a request has been outstanding on the path exceeds a specified I/O timeout. The timeout is specified when throttling is enabled by using the `vxdmpadm setattr` command. When timeout-based throttling is enabled, DMP stops issuing I/Os on a path if there is at least one request that has been outstanding on the path for the specified timeout. This kind of throttling requires I/O statistics gathering to be enabled. Timeout-based throttling can be useful in cases where one or more paths are responding slowly to I/Os; when configured properly, the throttling mechanism can limit I/Os sent to the underperforming paths.

The following example illustrates how timeout-based throttling works with 2 I/O paths (P1 and P2), with an I/O timeout of 10 seconds and `minimumq` as the I/O policy:

- DMP gets a request, R1, which it issues on path P1. After 8 seconds, DMP receives a request, R2, which it issues on path P2, a request, R3, which it issues on path P1, and a request, R4, which it issues on path P2, following the minimum queue (the `minimumq` I/O policy) logic.
- After another 3 seconds, which is 11 seconds after R1 was issued, DMP receives requests R5 and R6. Since the throttling timeout has been exceeded for path P1, DMP issues both R5 and R6 on path P2, even though P1 would have been the normal choice for one of the requests based on the minimum queue logic.
- After another 2 seconds, R1 completes.
- After another second, DMP receives a request R7. At this point, neither path P1 nor P2 has a request that has been outstanding for 10 seconds, so neither will be throttled. Since P1 has the smaller queue, DMP issues R7 on path P1.

The statistics obtained from DMP or from operating system utilities such as `iostat` can be used to determine whether it is appropriate to enable timeout-based throttling. If the I/O service times on one or a few paths are seen to be much higher than others on a fairly balanced request distribution, timeout-based throttling may be appropriate for the concerned enclosures. The timeout value can be based on the observed service times on healthy paths. A larger value of the I/O timeout generally allows more requests to get queued before throttling happens, essentially delaying the detection of slow paths. A smaller value of the timeout can limit the number of concurrent requests.

Tuning Dynamic Multi-Pathing error handling

The error handling capabilities of Dynamic Multi-Pathing (DMP) are key to its goal of providing the high availability and performance made possible by multiple I/O paths. Aspects of error handling in DMP can be tuned to get the behavior desired in a particular environment. In most cases though, the default settings for the tunable parameters governing error handling work well and they should be changed with caution. The following examples illustrate tuning decisions related to error handling that an administrator can make, along with the relevant tunable parameters:

- Should DMP try to get detailed error information from the HBA interface in order to enable better error analysis (`dmp_fast_recovery`)?
- How many times should DMP retry an I/O on a path when the I/O returns an error but error analysis indicates that the path is not faulty (`dmp_retry_count`)?
- How long should DMP wait after sending a SCSI command before timing out (`dmp_scsi_timeout`)?

- Should DMP detect and avoid intermittently failing paths (`dmp_health_time` and `dmp_path_age`)?
- How long should DMP wait after a failover, such as with an active-passive array, before commencing I/Os (`dmp_delayq_interval`)?
- How should DMP behave when all paths to a LUN have failed, which in some cases indicates an array undergoing maintenance (`dmp_lun_retry_timeout`)?

Dynamic Multi-Pathing SCSI bypass

A key to intelligent error analysis in Dynamic Multi-Pathing (DMP) is its SCSI bypass capability. For normal application I/O, DMP effectively sits on top of the SCSI layer. DMP performs path selection and then issues the I/O to the SCSI driver. But, in case an I/O fails, DMP can bypass the SCSI layer to get more detailed error information directly from the HBA interface. The SCSI bypass capability is enabled and disabled using the tunable parameter `dmp_fast_recovery`. SCSI bypass gives DMP access to error information that the SCSI layer would normally not pass on and allows DMP to make better decisions on how errors should be handled. Although provided as a tunable parameter, in most cases you should not disable this capability.

Dynamic Multi-Pathing I/O failure handling

This section describes the following aspects of Dynamic Multi-Pathing's (DMP) I/O failure handling:

- When an I/O issued on a path fails, when does DMP retry the I/O on the same path before trying another path?
- If an I/O has failed on repeated retries, when does DMP decide to stop retrying and fail the I/O altogether?

When an I/O issued on a path fails, DMP first performs error analysis on the path on which the I/O failed. If error analysis indicates that the path is faulty, the I/O will be tried on another path, if one is available. However, if error analysis indicates that the path is not faulty, DMP will retry the I/O on the same path a certain number of times as specified by the tunable parameter `dmp_retry_count` before trying the I/O on another path. After each unsuccessful retry, DMP performs error analysis to determine whether the path is faulty. The default value of `dmp_retry_count` is 5; lowering this value causes DMP to try alternate paths sooner.

When an I/O fails on repeated retries, DMP at some point decides to fail the I/O instead of retrying it further. The decision on when to fail an I/O is based on how I/O failure handling has been configured for the enclosure. By default, when a

time limit of 300 seconds is reached, DMP stops retrying an I/O and fails it. This is the time-bound method of error retry with a time limit of 300 seconds. The time limit for the time-bound method can be changed for an individual enclosure, array or array-type using the `vxdmpadm setattr` command. Lowering the value of the time limit causes DMP to fail I/Os sooner; if the value is set too low DMP will fail I/Os that could have succeeded.

Instead of the time-bound method, administrators can configure an enclosure, array, or array-type to use the fixed-retry method for error retry. In this method, DMP fails an I/O that has not succeeded after a specified number of retries. The fixed-retry method can be configured for an enclosure, array, or array-type using the `vxdmpadm setattr` command, and the number of retries to use is specified as part of the command. The number of retries when the fixed-retry method is used for error retry should generally be set to a value greater than the value of `dmp_retry_count`, which specifies the number of retries per path.

For more information on configuring the response to I/O failures, see the *Dynamic Multi-Pathing Administrator's Guide*.

Avoiding suspect paths in Dynamic Multi-Pathing

Dynamic Multi-Pathing's (DMP) tries pro-actively to identify paths whose health is suspect and avoids these paths until their health is verified. To understand why this is important for good performance, consider the case where DMP issues an I/O on a path and the I/O fails because the path is faulty. The I/O might eventually succeed on another path, but the application will see a higher completion time for the I/O because of the time spent on the faulty path; this includes SCSI and HBA timeouts and DMP error processing time. If DMP could avoid the faulty path in the first place, performance would be better; of course, this is not always possible.

One way in which DMP tries pro-actively to identify and avoid I/O paths that might be faulty is by monitoring for Fibre Channel events that notify of errors in the Fibre Channel. This behavior is controlled by the tunable parameter `dmp_monitor_fabric`. DMP also uses the notion of Subpath Failover Group (SFG) along with the tunable parameter `dmp_sfg_threshold` to mark a whole group of related paths as suspect when it sees errors on some of the paths in the group.

See "Dynamic Multi-Pathing path analysis" on page 117.

DMP, by default, also tries to avoid scheduling I/Os on a path that is failing intermittently until it sees evidence that the state of the path is stable. When a path that was previously marked faulty is detected as healthy through one of DMP's path analysis features, DMP tracks the health of the path to make sure it is stable. If the state of the path changes to failed once again within a specified time, as specified by the tunable parameter `dmp_health_time`, the path is

considered to be intermittently failing. In this case, DMP does not schedule I/Os on the path until the path is seen to stay healthy for a specified period, as specified by the tunable parameter `dmp_path_age`. The default value of `dmp_health_time` is 60 seconds and `dmp_path_age` is 300 seconds. If one or both of these tunable parameters is set to 0, DMP does not detect intermittently failing paths. The overhead of tracking paths that are failing intermittently is low and it is recommended that this feature be left enabled.

Dynamic Multi-Pathing tunable parameters for error handling

The error handling behavior of Dynamic Multi-Pathing (DMP) can be controlled using the following tunable parameters:

`dmp_fast_recovery` — This tunable parameter controls whether DMP tries to obtain error information directly from the HBA interface, bypassing the SCSI layer. For AIX, Linux, and Solaris, the default is on and is the recommended value.

`dmp_retry_count` — This tunable parameter specifies the limit on the number of retries on the same path for cases where the I/O fails but error analysis indicates that the path is not faulty. The default value of this parameter is 5. While this parameter controls the retry limit for a path, the retry limit for an I/O can be configured using the `vxdmpadm setattr` command to use either the time-bound error retry method or the fixed-retry method.

`dmp_delayq_interval` — This is the time interval for which DMP waits before retrying an I/O in the case where an array fails over to a standby path. Some arrays are not capable of accepting I/O requests immediately after failover. The default value for this parameter is 15 seconds. This parameter should be changed only with proper knowledge of the characteristics of all the arrays attached to a system.

`dmp_health_time` — This parameter specifies the time in seconds for which a path must stay healthy; a path whose state changes from enabled to disabled within this time is marked intermittently failing. DMP does not enable I/Os on such paths until `dmp_path_age` seconds elapse. A value of 0 prevents DMP from detecting intermittently failing paths.

`dmp_path_age` — This parameter works with `dmp_health_time` as described above. A value of 0 prevents DMP from detecting intermittently failing paths.

`dmp_lun_retry_timeout`	This parameter determines DMP behavior in the event that all paths to a LUN have failed. This can sometimes be a transient error and can happen with some disk arrays during controller firmware upgrade, for example. This tunable parameter provides a way to handle such situations without failing the I/O outright. `dmp_lun_retry_timeout` specifies the time for which an I/O should be retried before failing it. The default is 0 which means I/Os are not retried in this situation. Instead of changing the default value of this parameter, it is recommended that enclosure-specific settings be changed for arrays that need this kind of handling.
`dmp_scsi_timeout`	This parameter specifies the timeout value for any SCSI command issued by DMP. The default value of this parameter is 30 seconds for Solaris and AIX, and 20 seconds for Linux. This parameter should generally be left at the default value unless some other parameter like the Fibre Channel timeout has been changed.

Dynamic Multi-Pathing path analysis

Path analysis helps improve the status information Dynamic Multi-Pathing (DMP) has on I/O paths, such as whether an I/O path is healthy or faulty. Good status information on I/O paths helps DMP provide better performance. With good status information on I/O paths, DMP can proactively avoid faulty paths during path selection. If an I/O path is faulty, DMP discovers the fact when it schedules I/O on the path and the I/O fails. However, there is a performance penalty when I/O is issued on a path that is faulty: error detection itself involves timeouts and retries, following which the I/O must be re-issued on another path. In many cases, the status information on I/O paths gathered by the path analysis features help DMP avoid faulty paths.

When paths that were previously marked as failed are known to be healthy again, DMP can start using these paths in its load balancing logic. This gives DMP more options for scheduling I/Os and in general results in better performance. The path analysis features help DMP identify previously faulty paths that have recovered.

Path status updates happen in two ways in DMP:

- In response to storage network fabric events and I/O errors.
- Periodically, as part of a process called path restoration.

The mechanism used for path analysis is path probing, where a SCSI inquiry command is issued on a path to determine whether it is healthy or faulty. If the

inquiry command fails, the path can be assumed to be faulty. Issuing path probes and the associated processing has an overhead. This overhead is usually small, and can be managed using the tunable parameters for path analysis.

In some cases, DMP might mark the status of a path as suspect; this usually happens when DMP has encountered an error on a related path. DMP does not issue new I/Os on a suspect path until its status has been verified, unless the path is the last path available for the DMP node.

Subpath Failover Group

Dynamic Multi-Pathing (DMP) uses the concept of a Subpath Failover Group (SFG) to make path analysis more efficient. An SFG is a group of I/O paths from the same HBA port to the same array port. Essentially, paths in an SFG are paths to different LUNs that share the same sub-path from the HBA port to the array port, meaning that the paths have the same endpoints in the storage network. Even in storage environments with a large number of LUNs and a large number of I/O paths, there are likely to be relatively few SFGs. Since paths in the SFG all have the same route through the storage network, these paths are likely to fail and recover at the same time as faults and recovery happen in the network fabric. DMP is able to use this fact to optimize path analysis by sometimes taking collective action for all paths in an SFG rather than for each path individually.

Path analysis on path errors and fabric events

When Dynamic Multi-Pathing (DMP) encounters a path error on an I/O, it can initiate path state changes for other paths in the same Subpath Failover Group (SFG). This feature is governed by the following tunable parameter:

`dmp_sfg_threshold`
: When DMP detects that the number of failed I/O paths in an SFG has reached this threshold, it marks all the paths in the SFG as suspect. DMP does not issue new I/Os on a suspect path until its state has been verified, unless the path is the last path available. This behavior of DMP helps proactively avoid faulty paths, since an error in one or more paths of an SFG likely points to a problem that might affect all paths in the SFG. If the `dmp_sfg_threshold` parameter is set to 0, path analysis based on SFG is disabled; on a path error, DMP does not take any action for other paths in the same SFG. The default value of this parameter is 1, in which case the failure of any one path in an SFG causes other paths in the SFG to be marked as suspect.

DMP can also use network fabric monitoring to detect events that might affect the state of I/O paths. This feature is controlled by the following tunable parameter:

`dmp_monitor_fabric` When this parameter is set to **on**, DMP monitors for Fibre Channel events and updates the path status based on events received. Typically, fabric events affect all paths in one of more SFGs rather than just an individual path. Fabric event monitoring works with Fibre Channel event information that DMP builds as part of discovery to identify the paths that might be affected by an event. Fabric monitoring uses the Storage Networking Industry Association (SNIA) HBA API. This vendor and platform specific HBA-API library must be available for this feature to work. The default value of `dmp_monitor_fabric` on Solaris and Linux is **on**, and on these platforms Symantec recommends that you set this parameter to **on**. The default value of `dmp_monitor_fabric` on AIX is **off**, and Symantec recommends that you set this parameter to **off** on AIX to avoid performance issues.

Overview of path restoration

The main component of path analysis in Dynamic Multi-Pathing (DMP) is path restoration, in which a kernel thread called the restore daemon periodically issues probes on a collection of I/O paths to determine their state. Path restoration is a useful mechanism that augments other ways in which DMP collects status information about I/O paths, namely as part of regular I/Os, some of which might return an error, and by monitoring fabric events.

Tuning path restoration is mainly about achieving the desired balance between better status information on paths and lower overhead due to path restoration: more frequent and comprehensive path probing results in better status information, but adds more overhead.

Default path restoration in Dynamic Multi-Pathing

Path restoration is enabled by default in Dynamic Multi-Pathing (DMP); the tunable parameter `dmp_enable_restore`, which can be used to turn on or turn off path restoration, has the value of **on** by default. The default behavior for path restoration is aimed at probing only a subset of the I/O paths rather than probing all paths; the paths probed are those that are likely to benefit DMP operation the most. In most cases, the default behavior of path restoration has low overhead, but yields the following benefits:

- Path probing happens at a relatively large interval of 300 seconds, which is the default value of the tunable parameter `dmp_restore_interval`. The restore daemon wakes up once in this interval and initiates probing of I/O paths. Since the default value of the interval is large, it helps keep path restoration overhead low.
- Rather than probe all I/O paths, the default setting results in probing of the following paths in each interval:
  - Paths that have been marked as failed due to previous errors
  - Paths that have been marked suspect
  - Paths for LUNs that have been idle for a while, meaning the LUNs have had no I/Os in a while

  As a result, the number of paths probed in each interval by the restore daemon is usually much less than the total number of paths; this keeps the overhead of path probing low. This default behavior of probing failed, suspect, and idle paths results from the default values of two tunable parameters: `dmp_restore_policy`, which specifies the path restoration policy and has default value of **check_disabled**, and `dmp_probe_idle_lun`, which has a default value of **on**. In addition, I/O statistics gathering has to be enabled for `dmp_probe_idle_lun` to have effect. With I/O statistics gathering enabled and `dmp_probe_idle_lun=on`, the **check_disabled** policy probes failed and suspect paths, as well as paths for idle LUNs.
- The rationale for focusing on failed paths, suspect paths, and idle LUN paths is that these are the paths for which DMP's status information is most likely to be outdated and hence probing these paths is likely to give the most benefit. The status information that DMP has on active paths on which I/Os are being issued is generally likely to be good; there is not much to be gained by additional periodic probing of these paths as part of path restoration.
- By default, DMP uses an optimization called low impact path probing that relies on the concept of the SFG to further reduce overhead of path probing. This optimization allows DMP to probe only a few paths per SFG and infer the state of other paths in the SFG based on these few probes. The low impact path probing optimization is controlled by two tunable parameters: `dmp_low_impact_probe`, which enables or disables this feature and has the default value of **true**, meaning that the feature is enabled, and `dmp_probe_threshold`, which decides the number of paths probed in each SFG and has the default value of **5**.

Enabling or disabling path restoration

Path restoration can be enabled or disabled using the tunable parameter `dmp_enable_restore`. Path restoration is enabled by default, and is an important mechanism by which Dynamic Multi-Pathing (DMP) updates the status information on I/O paths. If path restoration is disabled, the status information that DMP has on the paths is not as accurate as when path restoration is enabled. In some cases, this may result in DMP sending I/Os down faulty paths. The bigger problem with disabling path restoration is that most paths that had failed but have now recovered are not recognized as healthy by DMP; over time this can create a scenario in which DMP has very few scheduling options and therefore operates sub-optimally.

Symantec recommends that you do not disable path restoration. If the performance overhead of path restoration is a concern, you should keep path restoration enabled and tune other parameters to reduce overhead. The default settings for path restoration result in low overhead and are appropriate for most environments, but you can further reduce the overhead of path restoration through tuning.

Path restoration policy

When path restoration is enabled, the restore policy determines which paths are probed. The value of the tunable parameter `dmp_restore_policy` determines which path restoration policy is in effect. You can specify the following parameter values:

`check_disabled`

This is the default path restoration policy. It is a low overhead policy that mainly probes paths that have been marked as failed or as suspect due to previous errors to see if these paths are now healthy. The overhead of the policy can be further reduced by enabling the low impact path probing optimization via the `dmp_low_impact_probe` parameter, which uses the notion of an SFG to probe a subset of suspect paths rather than all of them. If idle LUN probing, which is set by the `dmp_probe_idle_lun` parameter, and statistics gathering are enabled, paths for idle LUNs are probed in addition to failed and suspect paths.

See "Tunable parameters for path restoration" on page 123.

The status information on I/O paths gathered by this policy complements the status information that DMP gets as a result of regular application I/O. Regular I/O on active paths gives DMP knowledge of the health of those paths, and the `check_disabled` policy in path restoration gives DMP current status information on previously failed and suspect paths, and optionally, on paths to idle LUNs. If the DMP I/O policy in effect does not distribute I/O load on available paths, such as due to the single-active I/O policy, the `check_disabled` path restoration policy results in limited status information on important I/O paths. However, the default I/O policy (`minimumq`) does a good job of distributing I/Os on all available paths, so the `check_disabled` policy's focus on probing only failed and suspect paths and idle LUNs works well with the default I/O policy. With active/passive arrays, status information on secondary paths is not very good when the `check_disabled` policy is in effect; this is usually acceptable since secondary paths come into play only after a controller failover.

`check_all`

In this policy, all I/O paths are probed in each interval. This policy gives the best status information of all the path restoration policies, but its overhead can be concern especially in environments with a large number of I/O paths.

`check_periodic`	This policy tries to get the benefits of both `check_disabled` and `check_all`. It uses `check_all` only once every *N* intervals, where *N* is the value of the tunable parameter `dmp_restore_cycles`; other intervals use the `check_disabled` policy. The default value of `dmp_restore_cycles` is 10. Thus, by default, `check_periodic` incurs the higher overhead of the `check_all` policy only in 1 of 10 intervals; in the other 9 intervals, it uses the low overhead `check_disabled` policy. For the intervals in which the `check_disabled` policy is in effect, low impact probing and idle LUN probing will be used depending on whether these are enabled as based on the values of the relevant tunable parameters. The `check_periodic` policy offers a way to benefit from the low overhead of `check_disabled` most of the time, while infrequently collecting the more complete status information that `check_all` provides. For environments with a large number of paths, the overhead in the interval with the `check_all` policy can still be a concern.
`check_alternate`	This policy checks that there are at least two healthy paths for each DMP node. Typically, a DMP node corresponds to a LUN. In effect, the policy tries to ensure that in the event of a single path failure, there will be another healthy alternate path to provide availability. The policy probes I/O paths for each DMP node until it finds two paths that are healthy. If the policy is not able to find two healthy paths, it generates a notification. If the policy finds two healthy paths, it does not probe more paths for the DMP node. In the ideal case where there are no faulty paths, this policy probes exactly two paths per DMP node; in real life scenarios where some of the paths may be faulty, the number of paths probed by this policy will be somewhat more. In environments with a large number of LUNs and very few I/O path failures, the `check_disabled` policy can have a lower overhead than the `check_alternate` policy.

Tunable parameters for path restoration

The following table contains the complete list of tunable parameters related to path restoration:

`dmp_enable_restore` This tunable parameter enables or disables path restoration. When path restoration is enabled, a restore daemon is started that periodically probes I/O paths and updates their status based on the results of the probes. Path restoration can be disabled by setting `dmp_enable_restore` to **off**. Other path mechanisms in DMP for updating path status, such as updating path status based on fabric events, continue to function independently even when path restoration is disabled. To disable path restoration when the restore daemon is already running, use the `vxdmpadm stop restore` command. Since path restoration is a useful mechanism by which DMP keeps its path status information updated, Symantec does not recommend disabling path restoration. If the overhead of path restoration is a concern, other parameters for path restoration can be tuned to reduce this overhead. The default value of `dmp_enable_restore` is **on**, meaning that path restoration is enabled by default.

`dmp_restore_interval` This tunable parameter specifies the interval at which the restore daemon wakes up and initiates path probing. Increasing the value of this parameter decreases path restoration overhead since path probing is initiated less frequently. Decreasing the value of this parameter below the default value is generally not recommended as it can increase path restoration overhead and affect performance adversely. The default value is 300 seconds.

`dmp_restore_policy` This tunable parameter specifies the path restoration policy, which largely determines which paths are probed as part of path restoration. Path restoration policies are described in more detail in a separate section above. The default policy is `check_disabled`, which probes paths that have been marked as failed or as suspect, and based on other settings can probe the I/O paths of idle LUNs.

`dmp_restore_cycles`

This tunable parameter takes effect with the `check_periodic` policy.

See "Path restoration policy" on page 121.

Since the default policy is `check_disabled`, this parameter does not affect path restoration unless the policy is changed via the `dmp_restore_policy` tunable parameter . If `dmp_restore_cycles=`*N* and the policy in effect is the `check_periodic` policy, then DMP uses the `check_disabled` policy for *N*-1 intervals, and then uses the `check_all` policy in the *N*th interval; this pattern of switching between `check_disabled` and `check_all` is repeated. In other words, the `check_all` policy is used once every `dmp_restore_cycles` number of intervals and `check_disabled` policy is used in other intervals. The default value of `dmp_restore_cycles` is 10.

`dmp_low_impact_probe`

This parameter is used to enable or disable the low impact path probing feature of DMP which uses the concept of an Subpath Failover Group (SFG) to optimize path probing. Low impact probing is relevant only with the check_disabled policy, and in the `check_disabled` phases of the `check_periodic` policy. This feature reduces path probing overhead in the case of suspect paths. When low impact path probing is enabled, instead of probing all suspect paths, DMP probes only a certain number of suspect paths as specified by the tunable parameter `dmp_probe_threshold` per SFG. If all probed paths turn out to be in failed state, the status of all suspect paths in the SFG is set to failed; if at least one suspect path turns out to be healthy, the remaining suspect paths are probed individually. The default value of `dmp_low_impact_probe` is **on**, meaning that the feature is enabled by default.

`dmp_probe_threshold`

This parameter determines the number of paths to probe per SFG when low impact probing is enabled. The default value of this parameter is 5.

`dmp_probe_idle_lun`

This parameter is used to turn on or turn off probing of paths to idle LUNs by the restore daemon. Without idle LUN probing, the status information that DMP has for paths to an idle LUN may not be recent because of the fact that there have been no recent I/Os to the LUN. Hence, if a LUN that was previously idle becomes active, meaning that the LUN receives I/Os, DMP's scheduling of I/Os may be sub-optimal initially until it builds a clearer picture of path statuses for the LUN. An administrator can choose to maintain improved status information for idle LUNs by keeping idle LUN probing on. Alternately, an administrator can choose to reduce path restoration overhead by turning off idle LUN probing. In environments where you know that many LUNs will continue to be idle, turning off idle LUN might improve performance; however, for the LUNs that become active later, the initial I/Os after the LUNs have become active might incur penalties because the status information on the paths to the LUNs was outdated. For this parameter to take effect, DMP statistics gathering must be enabled. If statistics gathering is not enabled, DMP is not able to identify idle LUNs and this parameter is considered to be off. This parameter is relevant mainly when the path restoration policy is `check_disabled`. When the policy is `check_all`, all paths--including paths for idle LUNs--are probed anyway. The default value is **on**.

Summary of Dynamic Multi-Pathing tuning

Dynamic Multi-Pathing (DMP) has a large number of tunable parameters and configuration options that have an impact on performance. However, in most cases, the default settings are appropriate. Many of the tunable parameters rarely need to be changed. The following list briefly summarizes some of the more common performance tuning changes that an administrator might perform:

- The I/O policy in DMP determines how load is balanced across multiple paths for better performance. The default I/O policy, `minimumq`, is the recommended I/O policy. Among the strengths of this policy is its ability to adapt well when there are fluctuations in the performance delivered by paths by automatically redirecting more I/Os to better performing paths. The `balanced` and `round-robin` I/O policies have strengths that make them reasonable alternatives to `minimumq` under certain conditions. However, these policies should be considered only when the different paths for a LUN are delivering roughly the same performance and path performance is seen to be steady rather than fluctuating.

- When one or more paths to a LUN are performing poorly compared to the others, a load-balancing policy such as `minimumq` automatically redirects much of the I/O load away from the poorly performing paths. The negative performance impact of slow paths can be further reduced by configuring timeout-based I/O throttling for the relevant enclosures.
- In environments with a large number of I/O paths per LUN, `dmp_retry_count` can be reduced from its default of 5. This causes DMP to try alternate paths sooner when there are I/O errors and can improve performance.
- Path restoration in DMP can be tuned for better status information on I/O paths or for lower operating overhead; the default setting provides a good balance that is suited for most complex environments. In some environments, the overhead of path restoration can be reduced by turning off idle LUN probing, especially if the LUNs are expected to continue to be idle.
- The tunable parameter `dmp_daemon_count` determines the number of kernel threads used for DMP administrative activities. On high end servers, increasing the value of this parameter can improve performance.

Appendix A

Tuning Virtual Memory for Veritas File System on AIX

This appendix includes the following topics:

About tuning Virtual Memory for Veritas File System on AIX

The following list describes key AIX-specific characteristics of Veritas File System (VxFS) that are relevant in tuning for improved buffered I/O performance:

- The first mount of a VxFS file system on the AIX platform allocates Virtual Memory Management (VMM) structures, specifically Page Device Table (PDT) structures and VMM buffers.
- The last unmount of a VxFS file system de-allocates these VMM structures.
- Between 1 and 64 VxFS PDT structures and many thousands of VxFS VMM buffers are created and the default values are autotuned based on the system configuration.
- When an inode is brought in-core, a PDT is associated with the inode. The PDT is selected on a round-robin basis.
- All buffered I/Os to and from a given file are performed using the VMM buffers attached to the PDT that is associated to the file's in-core inode.
- The VMM buffers are only used to perform buffered I/O: either the paging-in or paging-out of client pages.

- VxFS 5.1 and later can function in D_REFUND mode (`drefund_enable`=1). However, D_REFUND is supported (`drefund_supported`=1) only on AIX 6.1 TL2 and later.
- The `vxtunefs` command can be used to determine the current values for `drefund_supported` and `drefund_enable`.
- In the absence of D_REFUND mode, the number of PDTs are automatically tuned to 4 times the number of CPUs, up to 64. The total number of VMM buffers are autotuned similarly based on the amount of physical memory of the system. The VMM buffers are then evenly distributed across the PDTs. Hence, if there are either fewer PDTs or a greater number of VMM buffers, then the number of VMM buffers per PDT is larger. The following `vxtunefs` commands can be used to view the number PDTs and VMM buffers on the system:

```
# /opt/VRTS/bin/vxtunefs -D print | grep num_pdt
# /opt/VRTS/bin/vxtunefs -b
```

- In D_REFUND mode, the number of VMM buffers on a PDT can be dynamically grown and shrunk based on the demand, unlike the static distribution in absence of D_REFUND mode.
- D_REFUND mode faces the limitation of having only 50,000 VMM buffer to allocate dynamically across the PDTs. This can be a limitation in some cases where D_REFUND should be turned off deliberately followed by the appropriate tuning for number of PDTs and VMM buffers.
- Number of PDTs and VMM buffers should only be tuned in absence of D_REFUND.
- In the absence of D_REFUND, VxFS performs accounting by allowing the reservation and un-reservation of VMM buffers that are associated with a PDT, before performing an I/O. No such accounting is performed in D_REFUND mode.
- Such accounting can sometimes lead to frequent memory allocation, which causes high CPU usage by `vx_sched`. In such scenarios, you can disable the accounting by setting the `vmmbufs_resv_disable` tunable using the `vxtunefs` command.

See "Advice for tuning Veritas File System on AIX " on page 130.

Advice for tuning Veritas File System on AIX

The following list provides advice for tuning Veritas File System (VxFS) on AIX:

- Tuning the number of Page Device Tables (PDTs)
  Symantec recommends that you reduce the number of PDTs by either 50% or 75%. For example, if the autotuned value is 64 PDTs, then reduce the value to 16 PDTs. To tune the number of PDT, all VxFS file systems must be unmounted. The number of PDTs should never be tuned to a drastically lower number as this can lead to a contention over the PDT lock.
  Set the `num_pdt` tunable to tune the number of PDTs:

  ```
  # vxtunefs -D num_pdt=PDTs
  ```

  See the `vxtunefs`(1M) manual page.

  Warning: Do not tune the number of PDTs to 1.

- Tuning the number of VMM buffers
  Symantec recommends that you increase the number of VxFS VMM buffers by a large amount. You can assume that each VMM buffer structure and its related objects use a little less than 768 bytes. Symantec recommends that you tune the value to be lower of the two upper limits of either 2-GB worth of VMM buffers at 768 bytes each, or up to ~1% of the system memory size.
  The maximum number of VMM buffers that can be tuned is now up to 1000% of the autotuned value, whereas previously the value could only be increased by 100% of the maximum. 2-GB worth of VMM buffers at 768 bytes each is approximately 2.75 million VMM buffers. However, if 1% of the physical memory is less than 2 GB, then you should tune the value to 1% of the physical memory. You might want to tune the value to higher than 1% of the physical memory, but do not tune the value to higher than 2.75 million VMM buffers.
  For example, a system with 64 GB of physical memory will have the number of VMM buffers auto-tuned to around 162,000. The auto-tuned number is far less than the 1% of physical memory recommended. If we want to increase the number of VMM buffers to utilize 1% of the 64GB (which is 6.4GB), we can use the following calculation:
  1% of 64GB / 768 bytes per VMM buffer = 64 * 1024 * 1024 * 1024 bytes * 1% / 768 bytes per VMM buffer
  = 894784 VMM buffers.
  To increase the number of VMM buffers to the above value, we need to increase it to 894784 / 162000 = 5.52 times of the current value.
  Using the `vxtunefs -b` commaned, the percentage increase should be (5.52 -1) * 100% = 452%:

  ```
  # vxtunefs -b 452
  ```

If we specify `vxtunefs -b 100`, that will double the current value. So if we specify 452, it will increase the value to 5.52 times of the current value.

- Counters that show a lack of VMM buffers per PDT
  Rapidly increasing or high values of the following counters obtained through the `vxfsstat` command show a lack of VMM buffers per PDT:
  - `vxi_putpage_dirty_partial`
    This counter is incremented when a write I/O VMM buffer reservation request gets satisfied only in part.
  - `vxi_pgalloc2_async_partial`
    This counter is incremented when the number of VMM buffers that we want to reserve is less than the number that are currently available for reservation. In this case, we trim the size of the asynchronous read I/O to match the number of VMM buffers that were reserved.
  - `vxi_putpagedirty_eagain`
    This counter is incremented when a VMM buffer reservation request cannot be satisfied in full or in part
  - `vxi_pagealloc2_eagain`
    This counter is incremented when the are no VMM buffers available for reservation, except the last 32. The current asynchronous read I/O is not performed when this counter is incremented.
  - `vxi_pdt_sleep_cnt`
    This counter indicates the number of threads that have had to sleep waiting for VMM buffers to be unreserved on the PDT. If the number of VMM buffers we wanted to reserve is not currently available, we increment this counter according to the PDT we are utilizing, and then sleep in a queue waiting for the VMM buffers to be unreserved.
  - `vxi_pdt_wake_cnt`
    This counter indicates the number of threads that have been woken up after sleeping. Usually, the `vxi_pdt_sleep_cnt` and `vxi_pdt_wake_cnt` have the same value. But in a situation when the threads are currently sleeping, the `vxi_pdt_sleep_cnt` might be a higher value than the `vxi_pdt_wake_cnt`.
- On AIX TLS and VxFS releases that support D_REFUND, you do not need to tune the number of PDTs and VMM buffers.
  If `drefund_enable` is set to 1, then D_REFUND mode is enabled and you do not need to tune the number of PDTs nor the number of VMM buffers. The VMM buffers are dynamically grown and shrunk as required. The D_REFUND mode is supported only on 6.1 TL2 and later releases. The default

drefund_enable value on 5.3 TL2 is 0 and the default drefund_enable value on 6.1 TL2 is 1.
The correct D_REFUND mode operation requires certain APARs to be installed. For more information on the necessary APARs, see the *Veritas Storage Foundation Release Notes*.
VxFS keeps track of this D_REFUND support internally. VxFS operates in D_REFUND mode only if the drefund_enable value is 1 and the operating system supports the D_REFUND mode. The operating system supports the D_REFUND mode if the D_REFUND supported value is 1. You can check the value of D_REFUND supported by running the vxtunefs -D print command.
Set the drefund_enable tunable to enable or disable D_REFUND mode:

```
# vxtunefs -D drefund_enable={0|1}
```

See the vxtunefs(1M) manual page.

Note: If the D_REFUND mode is disabled, then you must tune the number of PDTs and the number of VMM buffers.

- During poor performance, collect the following output so that Symantec can recommend tuning:
  AIX VMM statistics:

```
# vmstat -Ilt 1
# topas
# vmo -a
# vmstat -v
```

  Capture the vmstat -v output every second.
  VxFS statistics:

```
# /opt/VRTS/bin/vxtunefs -p /mnt1
# /opt/VRTS/bin/vxtunefs -D print
# /opt/VRTS/bin/vxfsstat -w /tmp/vxfsstat.out -t 1 /mnt1
# /opt/VRTS/bin/vxtrace -d /tmp/vxtrace.out -g dg1 vol1
```

  VxVM statistics:

```
# vxstat -g dgname -r
# vxstat -g dgname -vps -i 1
```

Index