Tru64 UNIX

System Configuration and Tuning

Part Number: AA-Q0R3G-TE

April 1999

Product Version: Tru64 UNIX Version 4.0F

This manual describes high-performance and high-availability configurations and provides recommendations for improving operating system performance.

Compaq Computer Corporation
Houston, Texas

Contents

About This Manual

This manual contains information about configuring Compaq Tru64 UNIX (formerly DIGITAL UNIX) for high performance and high availability. This manual also describes how to tune systems to improve performance.

For Tru64 UNIX Version 4.0F, it is recommended that you use the graphical user interface (GUI) for system administration. This GUI is presented by SysMan, an application that is loaded by default when the Common Desktop Environment (CDE) software is loaded on your system. The SysMan applications are available in the Application Manager, which you can access from the CDE Front Panel.

Audience

This manual is intended for system administrators who are responsible for managing a Tru64 UNIX operating system. Administrators should have an in-depth knowledge of their applications and users, in addition to operating system concepts, commands, and utilities. Such an understanding is crucial to successfully tuning a system for better performance.

New and Changed Features

Additions and changes that have been made to this manual for Tru64 UNIX Version 4.0F include the following:

- Kernel subsystem attributes are now documented in the reference pages. See `sys_attrs`(5) for more information.
- Section 1.1 describes a methodology that you can use to configure and tune high-performance and high-availability systems.
- NetRAIN (redundant array of independent network adapters) provides high network availability by allowing you to configure multiple interfaces on the same LAN segment into a single interface. See Section 2.6.3.
- This manual includes a description of how to increase the maximum number of open file descriptors for a specific process. See Section 5.5.2.
- The `mount` command allows you to disable flushing file system read access times to disk. You can improve file system performance for proxy

servers by specifying at mount time that the file system update only the in-memory file access time when a read system call is made to a file. The file system will update the on-disk stat structure only if the file is modified. See Section 9.2.8 for information.

- Prestoserve supports the `presto-buffer-hash-size` attribute, which allows you to control the size of the Prestoserve buffer cache hash table. See Section 9.2.10.
- If your system opens and then reuses many files (for example, if you have a proxy server), you may be able to improve Advanced File System (AdvFS) performance by increasing the number of AdvFS access structures that the system places on the access structure free list at startup time. See Section 9.3.4.3.
- You can prevent partial AdvFS writes if a system crash occurs. Use the `chfile -L on` command to enable atomic write data logging for a specified file. See Section 9.3.4.11.
- Smooth sync functionality improves UNIX File System (UFS) asynchronous I/O performance by preventing I/O spikes caused by the update daemon and increasing the chance of a buffer cache hit. Smooth sync is controlled by the `vfs` subsystem attribute `smoothsync_age`. See Section 9.4.3.7 for more information.
- You may be able to improve network performance by using the following new `inet` subsystem attributes:
  - The `tcbhashnum` attribute specifies the number of TCP hash tables. See Section 10.2.2.
  - The `ipport_userreserved_min` attribute specifies the minimum port number of the range from which the kernel selects outgoing ports. See Section 10.2.5.
  - The `ipqs` attribute specifies the number of IP input queues. See Section 10.2.19.

Organization

This manual consists of eleven chapters and a glossary:

Chapter 1	Describes a configuration and tuning methodology, and introduces the terms and concepts related to performance and availability.
Chapter 2	Describes how to characterize your applications and users and choose a configuration that will meet your needs.
Chapter 3	Describes how to monitor subsystems and identify and solve performance problems.

Chapter 4	Describes how to improve system performance by using system-specific and advanced tuning recommendations, how to modify the kernel, and how to solve common performance problems.
Chapter 5	Describes how to tune operating system limits in order to provide applications and users with more system resources.
Chapter 6	Describes memory operation and how to monitor and tune the virtual memory subsystem.
Chapter 7	Describes how to monitor CPUs and improve CPU performance.
Chapter 8	Describes how to configure, monitor, and tune the disk I/O subsystem, including the Logical Storage Manager (LSM).
Chapter 9	Describes how to configure, monitor, and tune file systems.
Chapter 10	Describes how to monitor and tune the network subsystem.
Chapter 11	Describes how to improve application performance.
Glossary	Lists terms relating to system performance and availability.

Related Documents

The *System Administration* manual provides information on managing and monitoring your system. The *Programmer's Guide* provides information on the tools for programming on the Tru64 UNIX operating system. It also provides information on how to optimize the code used to create an application program, and how to optimize the results of the build process. The *Asynchronous Transfer Mode* manual contains information about tuning Asynchronous Transfer Mode (ATM).

The following Tru64 UNIX manuals also provide useful, relevant information:

- *Technical Overview*
- *Network Administration*
- *Logical Storage Manager*
- *AdvFS Administration*
- *Systems & Options Catalog*

The printed version of the Tru64 UNIX documentation set is color coded to help specific audiences quickly find the books that meet their needs. (You can order the printed documentation from Compaq.) This color coding is reinforced with the use of an icon on the spines of books. The following list describes this convention:

Audience	Icon	Color Code
General users	G	Blue
System and network administrators	S	Red
Programmers	P	Purple
Device driver writers	D	Orange
Reference page users	R	Green

Some books in the documentation set help meet the needs of several audiences. For example, the information in some system books is also used by programmers. Keep this in mind when searching for information on specific topics.

The *Documentation Overview* provides information on all of the books in the Tru64 UNIX documentation set.

Reader's Comments

Compaq welcomes any comments and suggestions you have on this and other Tru64 UNIX manuals.

You can send your comments in the following ways:

- Fax: 603-884-0120 Attn: UBPG Publications, ZKO3-3/Y32
- Internet electronic mail: `readers_comment@zk3.dec.com`

  A Reader's Comment form is located on your system in the following location:

  `/usr/doc/readers_comment.txt`
- Mail:

  Compaq Computer Corporation
  UBPG Publications Manager
  ZKO3-3/Y32
  110 Spit Brook Road
  Nashua, NH 03062-9987

  A Reader's Comment form is located in the back of each printed manual. The form is postage paid if you mail it in the United States.

Please include the following information along with your comments:

- The full title of the book and the order number. (The order number is printed on the title page of this book and on its back cover.)
- The section numbers and page numbers of the information on which you are commenting.
- The version of Tru64 UNIX that you are using.

- If known, the type of processor that is running the Tru64 UNIX software.

The Tru64 UNIX Publications group cannot respond to system problems or technical support inquiries. Please address technical questions to your local system vendor or to the appropriate Compaq technical support office. Information provided with the software media explains how to send problem reports to Compaq.

Conventions

The following conventions are used in this manual:

#	A number sign represents the superuser prompt.
`%` **`cat`**	Boldface type in interactive examples indicates typed user input.
`file`	Italic (slanted) type indicates variable values, placeholders, and function argument names.
[|] { | }	In syntax definitions, brackets indicate items that are optional and braces indicate items that are required. Vertical bars separating items inside brackets or braces indicate that you choose one item from among those listed.
⋮	A vertical ellipsis indicates that a portion of an example that would normally be present is not shown.
`cat`(1)	A cross-reference to a reference page includes the appropriate section number in parentheses. For example, `cat`(1) indicates that you can find information on the `cat` command in Section 1 of the reference pages.

1 Introduction to Performance and Availability

Businesses need a computing environment that is dependable and able to handle the workload placed on that environment. Users and applications place different demands on a system, and both require consistent performance with minimal down time. A system also must be able to absorb an increase in workload without a decline in performance.

By following the guidelines in this manual, you can configure and tune a dependable, high-performance Tru64 UNIX system that will meet your current and future computing needs.

This chapter provides the following information:

- A methodology that you can use to configure and tune high-performance and high-availability systems (see Section 1.1)
- Terminology that is used to define system performance (see Section 1.2)
- Introduction to high-performance configurations (see Section 1.3)
- Introduction to high-availability configurations (see Section 1.4)

Later chapters in this manual provide detailed information about choosing high-performance and high-availability configurations and solving performance problems.

1.1 Using the Configuration and Tuning Methodology

Before you configure and tune a system, you must become familiar with the terminology and concepts relating to performance and availability. See Section 1.2, Section 1.3, and Section 1.4 for more information on these topics.

In addition, you must understand how your applications utilize system resources, because not all configurations and tuning recommendations are appropriate for all types of workloads. For example, you must determine if your applications are memory- or CPU-intensive or if they perform many disk or network operations. See Section 2.1 for information about identifying a resource model for your configuration.

To help you configure and tune a system that will meet your performance and availability needs, use the following methodology:

1. Ensure that your hardware and software configuration is appropriate for your workload resource model and your performance and availability goals. See Chapter 2.
2. Set up the configuration, making sure that you adhere to guidelines for the following:
   - Memory and swap space (Section 2.3.2)
   - Disks, LSM, and hardware RAID (Chapter 8)
   - AdvFS, UFS, and NFS file systems (Chapter 9)
3. Perform the following initial tuning tasks:
   a. If you have a large-memory system, Internet server, or NFS server, follow the tuning recommendations that are described in Section 4.1.
   b. Run `sys_check` and consider following its configuration and tuning recommendations (see Section 4.2).
4. Monitor the system and evaluate its performance, identifying any areas in which performance can be improved (see Chapter 3).
5. If performance is deficient in an area, use the advanced tuning recommendations described in this manual to improve performance. Not all recommendations are appropriate for all configurations, and some provide only marginal benefits. Before implementing a tuning recommendation, you must determine if it is appropriate for your configuration and consider its benefits and tradeoffs.

   See Section 4.5 for information about solving common performance problems. You can also use the guidelines shown in the following table to help you tune your system.

If your workload consists of:	**You can improve performance by:**
Applications requiring extensive system resources	Increasing resource limits (Chapter 5)
Memory-intensive applications	Increasing the memory available to processes (Section 6.4)
	Modifying paging and swapping operations (Section 6.5)
	Reserving shared memory (Section 6.6)
CPU-intensive applications	Freeing CPU resources (Section 7.2)
Disk I/O-intensive applications	Distributing the disk I/O load (Section 8.1)

If your workload consists of:	You can improve performance by:
File system-intensive applications	Modifying AdvFS, UFS, and NFS operation (Chapter 9)
Network-intensive applications	Modifying network operation (Section 10.2)
Non-optimized or poorly-written applications applications	Optimizing or rewriting the applications (Chapter 11)

System tuning usually involves modifying kernel subsystem attributes. See Section 4.4 for information.

1.2 Defining System Performance

System performance depends on an efficient utilization of system **resources**, which are the hardware and software components that are available to users or applications. A system must perform well under the normal workload exerted on the system by the applications and the users.

The system workload changes over time. For example, you may add users or run additional applications. **Scalability** refers to a system's ability to utilize additional resources with a predictable increase in performance, or the ability to absorb an increase in workload without a significant performance degradation.

A performance problem in a specific area of the configuration is called a **bottleneck**. Potential bottlenecks include the virtual memory subsystem and SCSI buses. A bottleneck can occur if the workload demands more from a resource than its **capacity**, which is the maximum theoretical throughput of a system resource.

Performance is often described in terms of two rates. **Bandwidth** is the rate at which an I/O subsystem or component can transfer bytes of data. Bandwidth is often called the **transfer rate**. Bandwidth is especially important for applications that perform large sequential data transfers. **Throughput** is the rate at which an I/O subsystem or component can perform I/O operations. Throughput is especially important for applications that perform many small I/O operations.

Performance is also measured in terms of **latency**, which is the amount of time to complete a specific operation. Latency is often called delay. High-performance systems require a low latency time. I/O latency is measured in milliseconds; memory latency is measured in nanoseconds. Memory latency depends on the memory bank configuration and the amount of memory.

Disk performance is often described in terms of **disk access time**, which is a combination of the **seek time**, the amount of time for a disk head to move to a specific disk track, and the **rotational latency**, which is the amount of time for a disk to rotate to a specific disk sector.

Data transfers also have different access patterns. A **sequential access pattern** is an access pattern in which data is read from or written to contiguous (adjacent) blocks on a disk. A **random access pattern** is an access pattern in which data is read from or written to blocks in different (usually nonadjacent) locations on a disk.

In addition, data transfers can consist of file-system data or **raw I/O**, which is I/O to a disk or disk partition that does not contain a file system. Raw I/O bypasses buffers and caches, and it may provide better performance than file system I/O. Raw I/O is often used by the operating system and by database application software.

1.3 Understanding High-Performance Configurations

A system **configuration** consists of the hardware, operating system, and layered products that comprise a single system or a cluster of systems. CPUs, memory boards, the operating system kernel, and disk storage are all parts of a configuration. Different hardware and software configurations provide various degrees of CPU power, memory resources, I/O performance, and storage capacity.

After you configure your system, disk storage, networks, and applications, using the guidelines described in this manual, you may be able to **tune** the operating system in order to improve performance. Tuning usually involves modifying the kernel by changing **attribute** values, which affect the behavior and performance of the various kernel subsystems.

The following sections provide some background information about how the CPU, memory, and I/O configuration impact performance. See the Compaq *Systems & Options Catalog* and the Tru64 UNIX *Technical Overview* for information about hardware and operating system performance features.

1.3.1 CPU Resources

CPUs support different processor speeds and onboard cache sizes. In addition, you can choose single-CPU systems or **multiprocessor** systems, which allow two or more processors to share common physical memory. Environments that are CPU-intensive, such as large database environments, require multiprocessing systems to handle the workload.

An example of a multiprocessing system is a symmetrical multiprocessing (**SMP**) system, in which the CPUs execute the same version of the operating system, access common memory, and execute instructions simultaneously.

When programs are executed, the operating system moves data and instructions through CPU caches, physical memory, and disk swap space. Accessing the data and instructions occurs at different speeds, depending on the location. Table 1–1 describes the various hardware resources.

Table 1–1: Memory Management Hardware Resources

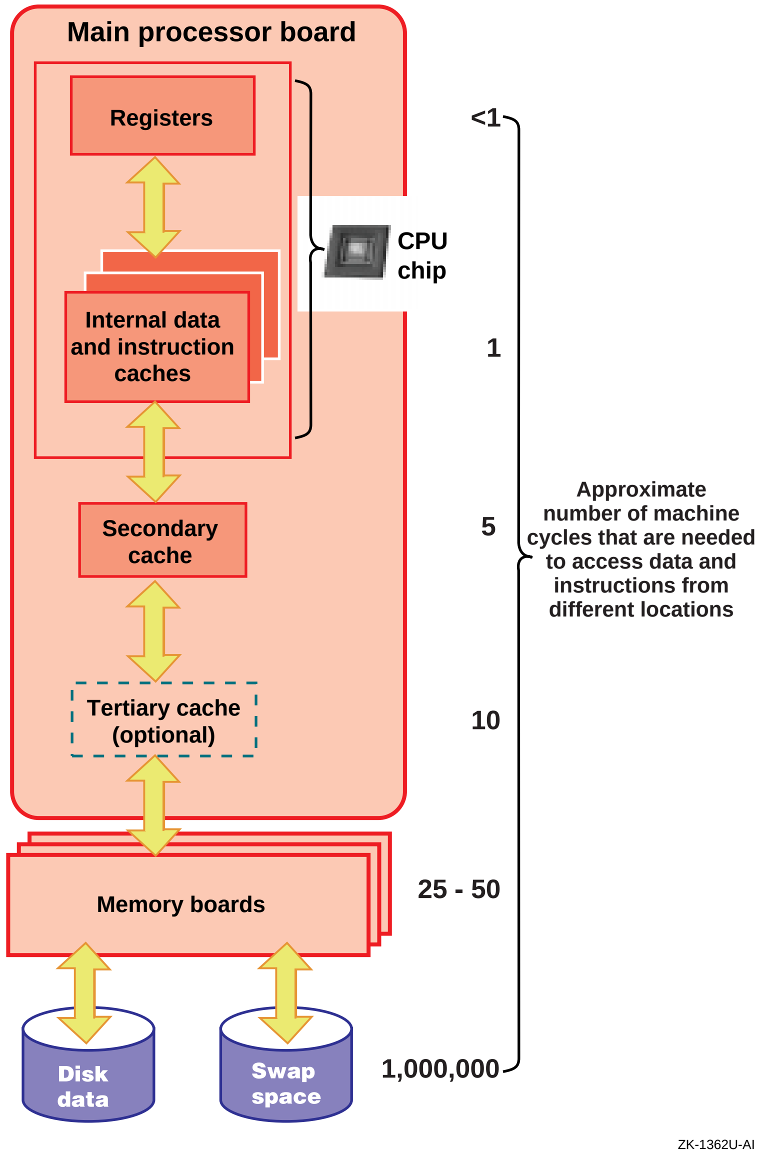
Resource	Description
CPU chip caches	Various internal caches reside in the CPU chip. They vary in size up to a maximum of 64 KB, depending on the processor. These caches include the translation lookaside buffer, the high-speed internal virtual-to-physical translation cache, the high-speed internal instruction cache, and the high-speed internal data cache.
Secondary cache	The secondary direct-mapped physical data cache is external to the CPU, but usually resides on the main processor board. Block sizes for the secondary cache vary from 32 bytes to 256 bytes (depending on the type of processor). The size of the secondary cache ranges from 128 KB to 8 MB.
Tertiary cache	The tertiary cache is not available on all Alpha CPUs; otherwise, it is identical to the secondary cache.
Physical memory	The actual amount of physical memory varies.
Swap space	Swap space consists of one or more disks or disk partitions (block special devices).

The hardware logic and the Privileged Architecture Library (PAL) code control much of the movement of addresses and data among the CPU cache, the secondary and tertiary caches, and physical memory. This movement is transparent to the operating system.

Movement between caches and physical memory is significantly faster than movement between disk and physical memory, because of the relatively slow speed of disk I/O. Applications should utilize caches and avoid disk I/O operations whenever possible.

Figure 1–1 shows how instructions and data are moved among various hardware components during program execution, and shows the machine cycles needed to access data and instructions from the hardware locations.

Figure 1–1: Moving Instructions and Data Through the Memory Hardware



Main processor board
Registers
CPU chip
Internal data and instruction caches
Secondary cache
Tertiary cache (optional)
Memory boards
Disk data
Swap space
<1
1
5
10
25 - 50
1,000,000
Approximate number of machine cycles that are needed to access data and instructions from different locations
ZK-1362U-AI

For more information on the CPU, secondary cache, and tertiary cache, see the *Alpha Architecture Reference Manual*.

There are several ways that you can optimize CPU performance. You can reschedule processes or use the Class Scheduler to allocate a percentage of CPU time to a task or application. This allows you to reserve a majority of

CPU time for important processes, while limiting CPU usage by less critical processes. See Chapter 7 for more information.

1.3.2 Memory Resources

Sufficient memory resources are vital to system performance. Configurations running CPU and memory-intensive applications often require very-large memory (**VLM**) systems that utilize 64-bit architecture, multiprocessing, and at least 2 GB of memory. Very-large database (**VLDB**) systems are VLM systems that also utilize complex storage configurations.

The total amount of **physical memory** is determined by the capacity of the memory boards installed in your system. The **virtual memory subsystem** tracks and manages this memory in 8-KB portions called **pages**, distributing them among three areas:

- **Wired memory**

  At boot time, the operating system and the PAL code wire a contiguous portion of physical memory in order to perform basic system operations.

  Static wired memory is reserved for operating system data, text and system tables, and the metadata buffer cache. The **Unified Buffer Cache** (**UBC**) also temporarily wires memory for AdvFS buffer cache pages.

  User processes also need to wire memory for address space, so the kernel allocates **dynamically wired memory** for these dynamically allocated data structures. The amount of dynamically wired memory varies according to the demand, but has a tunable limit (the default value is 80 percent of physical memory).

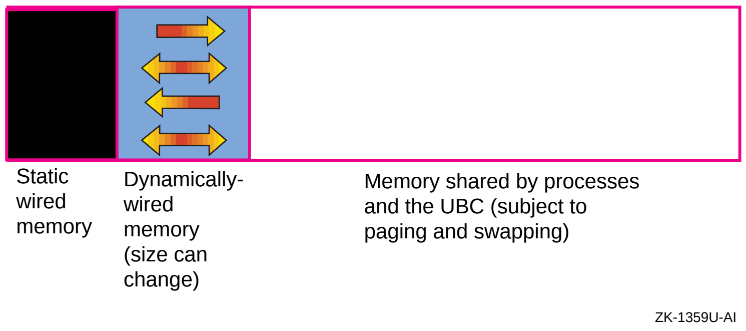
- **Virtual memory**

  Virtual memory refers to physical memory that is not wired by the kernel. This memory is managed by the virtual memory subsystem and used for processes' most-recently accessed **anonymous memory** (modifiable virtual address space) and **file-backed memory** (memory that is used for program text or shared libraries). Virtual memory is also used by the UBC to cache file system data.

The virtual memory subsystem uses physical memory, swap space, and various daemons and algorithms to efficiently allocate memory to processes and to the UBC, according to the demand. When the demand for memory increases, **paging** occurs. The oldest (least-recently used) pages are reclaimed and their contents moved to swap space, if necessary. The newly clean pages are then made available for reuse. Wired pages are not reclaimed. If paging cannot satisfy the memory demand, **swapping** is used to suspend processes and free large amounts of memory.

Figure 1–2 shows the division of physical memory.

Figure 1–2: Physical Memory Usage


Static wired memory
Dynamically-wired memory (size can change)
Memory shared by processes and the UBC (subject to paging and swapping)
ZK-1359U-AI

Because memory operations are significantly faster than disk I/O operations, **buffer caching** is used to cache recently-used disk data in physical memory. I/O performance is improved if the cached data is later reused, because a memory operation is used to retrieve data instead of a disk operation.

There are several buffer caches that are used to cache data:

- The UBC caches the most-recently accessed file data for reads and writes and for page faults from mapped file regions, in addition to Advanced File System (AdvFS) metadata and file data. The UBC and processes compete for the portion of physical memory that is not wired by the kernel.
- The **metadata buffer cache** caches only UFS and CDFS metadata and is part of static wired memory.
- The **AdvFS buffer cache** is a subset of the UBC.

Various kernel subsystem attributes control the amount of memory available to processes and to the file system buffer caches and the rate of page reclamation. You may be able to tune the attributes in order to optimize memory performance. See Chapter 6 for more information.

1.3.3 Disk Storage Configuration

Disk storage configurations vary greatly, so you must determine which configuration will meet the performance and availability needs of your applications and users.

Disk storage configurations can consist of single disks with traditional discrete **disk partitions**. Large storage configurations often use a **shared pool of storage** that is managed by the **Logical Storage Manager**

(**LSM**), which also provides high-performance and high-availability features, including RAID support.

Storage configurations can also include **hardware RAID** subsystems, which greatly expand the number of disks that can be connected to a single I/O bus and provide many high-performance and high-availability features, including RAID support and write-back caches. There are various types of hardware RAID subsystems, suitable for different environments.

Host bus adapters, RAID controllers, and disks have various performance features and support different Parallel Small Computer System Interface (**SCSI**) variants. SCSI is a device and interconnect technology that continues to evolve in terms of high performance, availability, and configuration flexibility. See Section 1.3.3.2 for more information about SCSI. See Section 1.3.3.1 for more information about RAID functionality.

See Chapter 2 and Chapter 8 for more information about storage configurations.

1.3.3.1 RAID Technology

You can use redundant array of independent disks (**RAID**) technology in a storage configuration for high performance and high data availability. You can obtain RAID functionality by using LSM (only RAID 0 and RAID 1) or a hardware-based RAID subsystem.

There are four primary RAID levels:

- **RAID 0**—Also known as disk **striping**, RAID 0 divides data into blocks and distributes the blocks across multiple disks in a array. Distributing the disk I/O load across disks and controllers improves throughput. Striping does not provide disk data availability.
- **RAID 1**—Also known as disk **mirroring**, RAID 1 maintains identical copies of data on different disks in an array. Duplicating data on different disks provides high data availability and improves disk read performance. You can combine RAID 1 with RAID 0 in order to mirror striped disks.
- **RAID 3**—A type of **parity RAID**, RAID 3 divides data blocks and distributes (stripes) the data across a disk array, providing parallel access to data and increasing bandwidth. RAID 3 also provides high data availability by placing redundant parity information on a separate disk, which is used to regenerate data if a disk in the array fails.
- **RAID 5**—A type of parity RAID, RAID 5 distributes data blocks across disks in an array. RAID 5 allows independent access to data and can handle simultaneous I/O operations, which improves throughput. RAID

5 provides data availability by distributing redundant parity information across the array of disks.

In addition, high-performance RAID controllers support **dynamic parity RAID** (also called **adaptive RAID 3/5**), which combines the benefits of RAID 3 and RAID 5 to improve disk I/O performance for a wide variety of applications. Dynamic parity RAID dynamically adjusts, according to workload needs, between data transfer-intensive algorithms and I/O operation-intensive algorithms.

Table 1–2 compares the performance features and degrees of availability for the different RAID levels.

Table 1–2: RAID Level Performance and Availability Comparison

RAID Level	Performance Feature	Degree of Availability
RAID 0 (striping)	Balances I/O load and improves throughput	Lower than single disk
RAID 1 (mirroring)	Improves read performance, but degrades write performance	Highest
RAID 0+1	Balances I/O load and improves throughput, but degrades write performance	Highest
RAID 3	Improves bandwidth, but performance may degrade if multiple disks fail	Higher than single disk
RAID 5	Improves throughput, but performance may degrade if multiple disks fail	Higher than single disk
Dynamic parity RAID	Improves bandwidth and throughput, but performance may degrade if multiple disks fail	Higher than single disk

It is important to understand that RAID performance depends on the state of the devices in the RAID subsystem. There are three possible states: steady state (no failures), failure (one or more disks have failed), and recovery (subsystem is recovering from failure).

There are many variables to consider when choosing a RAID configuration:

- Not all RAID products support all RAID levels.

  For example, only high-performance RAID controllers support dynamic parity RAID.

- RAID products provide different performance features.

  For example, only RAID controllers support write-back caches and relieve the CPU of the I/O overhead.

- Some RAID configurations are more cost-effective than others.

In general, LSM provides more cost-effective RAID functionality than hardware RAID subsystems. In addition, parity RAID provides data availability at a cost that is lower than RAID 1 (mirroring), because mirroring *n* disks requires 2*n* disks.

- Data recovery rates depend on the RAID configuration.

  For example, if a disk fails, it is faster to regenerate data by using a mirrored copy than by using parity information. In addition, if you are using parity RAID, I/O performance declines as additional disks fail.

See Chapter 2 and Chapter 8 for more information about RAID configurations.

1.3.3.2 SCSI Concepts

The most common type of SCSI is **parallel SCSI**, which supports SCSI variants that provide you with a variety of performance and configuration options. The SCSI variants are based on bus speed (Slow, Fast, or Ultra), data path (narrow or wide), and transmission method (single-ended or differential). These variants determine the bus bandwidth and the maximum allowable SCSI bus length.

Serial SCSI is the next generation of SCSI. Serial SCSI addresses parallel SCSI's limitations on speed, distance, and connectivity (number of devices on bus), and also provides availability features like hot swap and fault tolerance.

Fibre Channel is an example of serial SCSI. A high-performance I/O bus that support multiple protocols (SCSI, IPI, FIPS60, TCP/IP, HIPPI, and so forth.), Fibre Channel is based on a network of intelligent switches. Link speeds are available up to 100 MB/sec full duplex.

The following sections describe parallel SCSI concepts in detail.

1.3.3.2.1 Data Paths

Disks, host bus adapters, I/O controllers, and storage enclosures have a **data path**. The data path and the bus speed determine the actual bandwidth for a bus. There are two data paths available:

- **Narrow**

  Specifies an 8-bit data path. The performance of this mode is limited. SCSI bus specifications restrict the number of devices on a narrow SCSI bus to eight.

- **Wide**

  Specifies a 16-bit data path for Slow and Fast SCSI, and a 32-bit data path for UltraSCSI. This mode increases the amount of data that is transferred in parallel on the bus. SCSI bus specifications restrict the number of devices on a wide bus to 16.

Disks and host bus adapters that use a wide data path can provide nearly twice the bandwidth of disks and adapters that use a narrow data path. Wide devices can greatly improve I/O performance for large data transfers.

Most current disks have both wide and narrow versions. Devices with different data paths (or transmission methods) cannot be directly connected on a single bus. Use a SCSI signal converter (for example, a DWZZA or DWZZB signal converter) or a DWZZC UltraSCSI extender to connect devices with different data paths.

1.3.3.2.2 Transmission Methods

The **transmission method** for a bus refers to the electrical implementation of the SCSI specification. **Single-ended SCSI** is a low cost solution for devices that are usually located within the same cabinet. Single-ended SCSI usually requires short cable lengths. However, **differential SCSI** can be used to connect devices that are up to 25 meters apart.

A single-ended SCSI bus uses one data lead and one ground lead for the data transmission. A single-ended receiver looks at only the signal wire as the input. The transmitted signal arrives at the receiving end of the bus on the signal wire slightly distorted by signal reflections. The length and loading of the bus determine the magnitude of this distortion. Therefore, the single-ended transmission method is economical, but it is more susceptible to noise than the differential transmission method and requires short cables.

A differential SCSI bus uses two wires to transmit a signal. The two wires are driven by a differential driver that places a signal on one wire (+SIGNAL) and another signal that is 180 degrees out of phase (-SIGNAL) on the other wire. The differential receiver generates a signal output only when the two inputs are different. Because signal reflections are virtually the same on both wires, they are not seen by the receiver, which notices only differences on the two wires. The differential transmission method is less susceptible to noise than single-ended SCSI, enables the use of long cables, and uses 68-pin, high-density or 68-pin VHDCI (UltraSCSI) connectors.

You can directly connect devices only if they have the same transmission method (differential or single-ended) and data path (narrow or wide). Use a SCSI signal converter (for example, a DWZZA or DWZZB signal converter or a DWZZC UltraSCSI extender) to connect devices with different transmission methods or data paths.

1.3.3.2.3 SCSI Bus Speeds

The **SCSI bus speed**, also called the transfer rate or bandwidth, is the number of transfers per second. Fast bus speeds provide the best performance. Both bus speed and the data path (narrow or wide) determine the actual bus bandwidth (number of bytes transferred per second).

Table 1–3 shows the currently available bus speeds.

Table 1–3: SCSI Bus Speeds

Bus Speed	Maximum Transfer Rate (million transfers/sec)	Maximum Byte Transfer Rate - Narrow (MB/sec)	Maximum Byte Transfer Rate - Wide (MB/sec)
Slow	5	5	10
Fast	10	10	20
Ultra	20	20	40

Fast SCSI bus speed, also called **Fast10**, is an extension to the SCSI-2 specification. It uses the fast synchronous transfer option, enabling I/O devices to attain high peak-rate transfers in synchronous mode.

UltraSCSI, also called **Fast20**, is a high-performance, extended version of SCSI-2 that addresses many performance and configuration deficiencies. Compared to Fast SCSI bus speed, UltraSCSI doubles the bandwidth and configuration distances, but with no increase in cost. UltraSCSI also provides faster transaction times and faster, more accurate data analysis.

UltraSCSI devices are wide and can be either single-ended or differential. All UltraSCSI components are backward compatible with regular SCSI-2 components.

Because of UltraSCSI's high bus speed, single-ended UltraSCSI signals cannot maintain their strength and integrity over the same distance as single-ended Fast SCSI signals. Therefore, UltraSCSI technology uses **bus segments** and **bus extenders**, so that systems and storage can be configured over long distances.

An UltraSCSI bus extender couples two bus segments together without any impact on SCSI protocol. A bus segment is defined as an unbroken electrical path consisting of conductors (in cables or backplanes) and connectors. Every UltraSCSI bus segment must have two terminators, one at each end of the bus segment. Therefore, an UltraSCSI bus segment corresponds to an entire bus in Fast SCSI. The SCSI domain is the collection of SCSI devices on all the bus segments. As with a Fast SCSI bus, an UltraSCSI bus segment can only support devices of the same type (single-ended or differential).

In addition to extending the effective length of a bus, UltraSCSI bus extenders can be used as **SCSI signal converters**, so you can connect differential bus segments to single-ended bus segments. This allows you to mix differential and single-ended devices on the same bus. A bus extender also enables bus segments to be isolated from each other for maintenance reasons.

Although UltraSCSI components allow an UltraSCSI domain to extend for longer distances than a Fast SCSI bus, there are still limits. Also, because the use of bus expanders allows UltraSCSI domains to look like a tree, instead of a straight line, the concept of bus length must be replaced with the concept of the UltraSCSI domain diameter.

To set the bus speed on a host bus adapter, use either console commands or the Loadable Firmware Update (LFU) utility, depending on the type of adapter. Not all devices support all bus speeds. See the Compaq *Systems & Options Catalog* for information about SCSI device support.

1.3.3.2.4 SCSI Bus Length and Termination

There is a limit to the length of the cables in a SCSI bus. The maximum cable length depends on the bus speed and the transmission method (single-ended or differential). The total cable length for a physical bus or UltraSCSI bus segment is calculated from one terminated end to the other.

In addition, each SCSI bus or bus segment must be terminated only at each end. Improper bus termination and lengths are a common cause of bus malfunction.

If you are using devices that have the same transmission method and data path (for example, wide and differential), a bus will consist of only one physical bus (or multiple bus fragments in the case of UltraSCSI). If you have devices with different transmission methods, you will have both single-ended and differential physical buses or bus segments, each of which must be terminated only at both ends and adhere to the rules on bus length.

Table 1–4 shows the maximum bus lengths for different bus speeds and transmission methods.

Table 1–4: SCSI Bus and Segment Lengths

Bus Speed	Transmission Method	Maximum Bus or Segment Length
Slow	Single-ended	6 meters
Fast	Single-ended	3 meters
Fast	Differential	25 meters

Table 1–4: SCSI Bus and Segment Lengths (cont.)

Bus Speed	Transmission Method	Maximum Bus or Segment Length
Ultra	Differential	25 meters
Ultra	Single-ended	1.5 meters (daisy-chain configuration, in which devices are spaced less than 1 meter apart)
Ultra	Single-ended	4 meters (daisy-chain configuration, in which devices are spaced more than 1 meter apart)
Ultra	Single-ended	20 meters (point to point configuration, in which devices are only at the ends of the bus segment)

Note that the total length of a physical bus must include the amount of cable that is located inside each system and disk storage shelf. This length varies, depending on the device. For example, the length of cable inside a BA350, BA353, or BA356 storage shelf is approximately 1.0 meter.

1.3.4 Network Subsystem

The operating system supports various networks and network adapters that provide different performance features. For example, an Asynchronous Transfer Mode (ATM) high-performance network is ideal for applications that need the high speed and the low latency (switched, full duplex network infrastructure) that ATM networks provide.

In addition, you can configure multiple network adapters or use NetRAIN to increase network access and provide high network availability.

Kernel attributes control network subsystem operation. You may be able to modify attributes and tune the network subsystem in order to optimize network performance for your applications and workload.

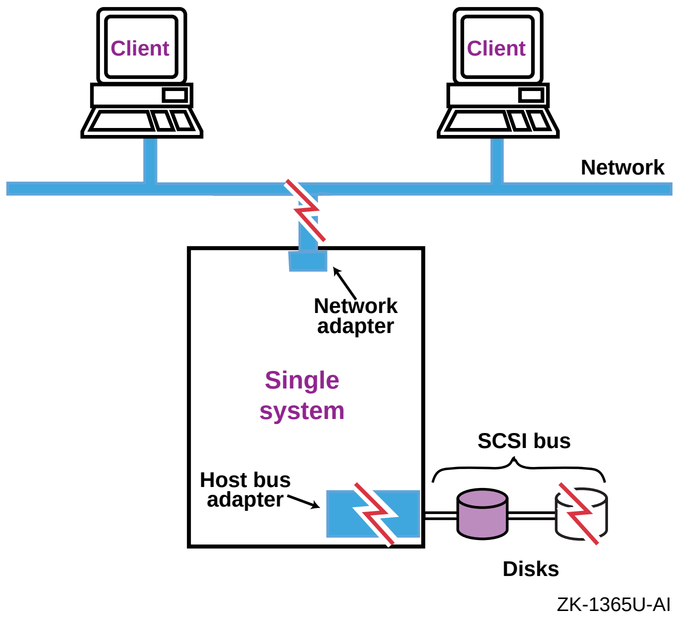
1.4 Understanding High-Availability Configurations

In addition to high performance, many environments require some degree of **high availability**, which is the ability to withstand a hardware or software failure. Resources (for example, disk data, systems, and network connections) can be made highly available by using some form of resource duplication or **redundancy**. In some cases, an automatic **failover** mechanism may also be used in order to make the resource failure virtually imperceptible to users.

There are various degrees of availability, and you must determine how much you need for your environment. Critical operations may require a configuration that does not have a **single point of failure**; that is, one in which you have duplicated each vital resource. However, some environments may be able to accommodate down time and may require only data to be highly available.

Figure 1–3 shows a configuration that is vulnerable to various single failures, including network, disk, and bus failures.

Figure 1–3: Configuration with Potential Points of Failure



By duplicating important resources, a configuration can be resistant to resource failures, as follows:

- Disk data

  RAID technology provides you with various degrees of data availability. For example, you can use RAID 1 (mirroring) to replicate data on different disks. If one disk fails, a copy is still available to users and applications. You can also use parity RAID for high data availability. The parity information is used to reconstruct data if a failure occurs.

  To protect data against a host bus adapter or bus failure, mirror the data across disks located on different buses. In addition, some host bus adapters support multiple paths to protect against bus and adapter failures and for concurrent access.

- Network access

  You can also make the network connection highly available by using redundant network connections. If one connection becomes unavailable, you can still use the other connection for network access. Whether you can use multiple networks depends on the application, network configuration, and network protocol.

  In addition, you can use **NetRAIN** (redundant array of independent network adapters) to configure multiple interfaces on the same LAN segment into a single interface and provide failover support for network adapter and network connections.

- System

  To make systems and applications highly available, you must use a TruCluster product to set up a **cluster**, which is a loosely coupled group of servers configured as **member systems** and usually connected to shared disk storage. Software applications are installed on every member system, but only one system runs an application at one time and makes it available to users.

  A cluster utilizes a **failover** mechanism to protect against failures. If a member system fails, all cluster-configured applications running on that system will fail over to a viable member system; that is, the new system starts the applications and makes them available to users.

  Some cluster products support a high-performance **cluster interconnect** that enables fast and reliable communications between members. You can configure redundant cluster interconnects for high availability. If one cluster interconnect fails, the cluster members can still communicate over the remaining interconnect.

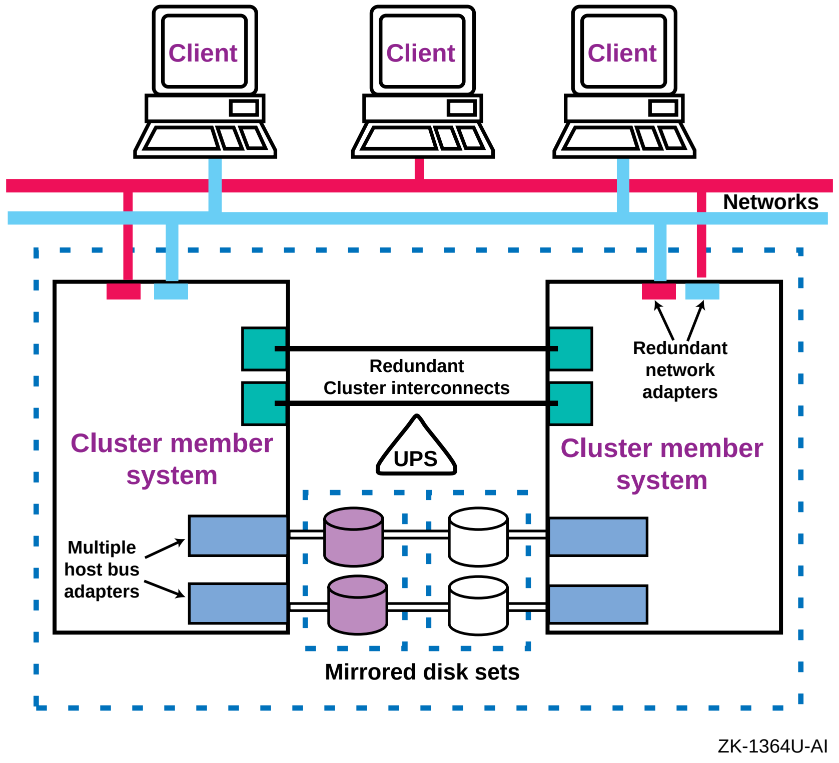
- Power

  Systems and storage units are vulnerable to power failures. To protect against a power supply failure, use redundant power supplies from different power sources. You can also protect disks against a power supply failure in a storage cabinet by mirroring the disks across independently powered cabinets.

  In addition, use an uninterruptible power system (**UPS**) to protect against a total power failure (for example, the power in a building fails). A UPS depends on a viable battery source and monitoring software.

Figure 1–4 shows a fully redundant cluster configuration with no single point of failure.

Figure 1–4: Fully Redundant Cluster Configuration



You must repair or replace a failed component as soon as possible to maintain some form of redundancy. This will help to ensure that you do not experience down time.

Production environments often require that resources be resistant to multiple failures. The more levels of resource redundancy, the greater the resource availability. For example, if you have only two cluster member systems and one fails, the remaining system is now a potential point of failure. Therefore, a cluster with three or more member systems has higher availability than a two-system cluster, because it has more levels of redundancy and can survive multiple system failures.

Availability is also measured by a resource's **reliability**, which is the average amount of time that a component will perform before a failure that causes a loss of data. It is often expressed as the mean time to data loss (MTDL), the mean time to first failure (MTTF), and the mean time between failures (MTBF).

See Section 2.6 for detailed information about setting up a high-availability configuration.

2 Planning a High-Performance and High-Availability Configuration

A high-performance configuration is one that will rapidly respond to the demands of a normal workload, and also maintain an adequate level of performance if the workload increases. A high-availability configuration provides protection against single points of failure.

This chapter describes how to perform the following tasks:

- Identify a resource model for your workload (Section 2.1)
- Identify performance and availability goals (Section 2.2)
- Choose hard-performance system hardware (Section 2.3)
- Choose hard-performance disk storage hardware (Section 2.4)
- Choose how to manage your disk storage (Section 2.5)
- Choose a high-availability configuration (Section 2.6)

2.1 Identifying a Resource Model for Your Workload

Before you can configure or tune a system, you must identify a resource model for your workload. That is, you must determine if your applications are memory-intensive or CPU-intensive, and how they perform disk and network I/O. This information will help you to choose the configuration and tuning recommendations that are appropriate for your workload.

For example, if the resource model for a database server indicates that the workload consists of large sequential data transfers, an appropriate configuration is one that provides high bandwidth. If a system performs many disk write operations, a mirrored disk configuration may not be an appropriate configuration.

Use Table 2–1 to help you determine the resource model for your workload and identify a possible configuration solution for each model.

Table 2–1: Resource Models and Possible Configuration Solutions

Resource Model	Configuration Solution
CPU-intensive	Multiprocessing system, fast CPUs, or RAID array
Memory-intensive	VLM system or large onboard CPU cache
Requires large amount of disk storage	System with a large I/O capacity, LSM, or RAID array
Requires low disk latency	Solid-state disks, fast disks, RAID array, or Fibre Channel
Requires high throughput	High-performance adapters, striping, RAID 5, or dynamic parity RAID
Requires high bandwidth	High-performance adapters, wide devices, RAID 3, or dynamic parity RAID
Performs many large sequential data transfers	High-performance disks, wide devices, striping, RAID 3, RAID 5, dynamic parity RAID
Performs many small data transfers	RAID 5
Issues predominantly read transfers	Disk mirroring, RAID 5, or striping
Issues predominantly write transfers	Prestoserve or write-back cache
Performs many network operations	Multiple network adapters, NetRAIN, or high-performance adapters
Application must be highly available	Cluster
Data must be highly available	Mirroring (especially across different buses), RAID 3, RAID 5, or dynamic parity RAID
Network I/O-intensive	Multiple network adapters or NetRAIN

2.2 Identifying Performance and Availability Goals

Before you choose a configuration, you must determine the level of performance and availability that you need. In addition, you must account for cost factors and plan for future workload expansion.

When choosing a system and disk storage configuration, you must be sure to evaluate the configuration choices in terms of the following criteria:

- Performance

  You must determine an acceptable level of performance for the applications and users. For example, may want a real-time environment that responds immediately to user input, or you may want an environment that has high throughput.

- Availability

  You must determine how much availability is needed. Some environments require only highly available data. Other environments require you to eliminate all single points of failure.

- Cost

  You must determine the cost limitations for the environment. For example, solid-state disks provide high throughput and high bandwidth, but at a high cost.

- Scalability

  A system that is scalable is able to utilize additional resources (for example, additional CPUs) with a predictable increase in performance. Scalability can also refer to the ability of a system to absorb an increase in workload without a significant performance degradation. Be sure to include in your plans any potential workload increases and, if necessary, choose a configuration that is scalable.

After you determine the goals for your environment, you can choose the system and disk storage configuration that will address these goals.

2.3 Choosing High-Performance System Hardware

Different systems provide different configuration and performance features. A primary consideration for choosing a system is its CPU and memory capabilities. Some systems support multiple CPUs, fast CPU speeds, and very-large memory (VLM) configurations.

Because very-large database (VLDB) systems and cluster systems usually require many external I/O buses, another consideration is the number of I/O bus slots in the system.

Be sure that a system is adequately scalable, which will determine whether you can increase system performance by adding resources, such as CPU and memory boards. If applicable, choose a system that supports RAID controllers, high-performance network adapters, and cluster products.

Table 2–2 describes some hardware options that can be used in a high-performance system and the performance benefit for each option.

Table 2–2: High-Performance System Hardware Options

Hardware Option	Performance Benefit
Multiple CPUs (Section 2.3.1)	Improves processing time
Fast CPU speed (Section 2.3.1)	Improves processing time
Onboard CPU cache (Section 2.3.1)	Improves processing time

Table 2–2: High-Performance System Hardware Options (cont.)

Hardware Option	Performance Benefit
Very-large memory (Section 2.3.2)	Improves processing time and decreases disk I/O latency
Large I/O capacity (Section 2.3.3)	Allows you to connect many I/O adapters and controllers for disk storage and network connections
High-performance disk storage support (Section 2.3.4)	Improves overall system performance and availability
High-performance network support (Section 2.3.5)	Increases network access and network performance

For detailed information about hardware performance features see the Compaq *Systems & Options Catalog.* For information about operating system hardware support, see the Tru64 UNIX Version 4.0F *Software Product Description*.

The following sections describe these hardware options in detail.

2.3.1 CPU Configuration

To choose a CPU configuration that will meet your needs, you must determine your requirements for the following:

- Number of CPUs

  Only certain types of systems support multiprocessing. If your environment is CPU-intensive or if your applications can benefit from multiprocessing, you may want a system that supports multiple CPUs.

  Depending on the type of multiprocessing system, you can install two or more CPUs. You must determine the number of CPUs that you need, and then choose a system that supports that number of CPUs and has enough backplane slots available for the CPU boards.

- CPU processing speed

  CPUs have different processing speeds and other performance features. For example, some processors support EV56 technology.

  If your environment is CPU-intensive, you may want to choose a system that supports high-performance CPUs.

- CPU cache size

  CPUs also have different sizes for on-chip caches, which can improve performance. Some systems have secondary caches that reside on the main processor board and some have tertiary caches.

Caches that reside on the CPU chip can vary in size up to a maximum of 64 KB (depending on the type of processor). These caches include the translation lookaside buffer, the high-speed internal virtual-to-physical translation cache, the high-speed internal instruction cache, and the high-speed internal data cache.

The secondary direct-mapped physical data cache is external to the CPU, but usually resides on the main processor board. Block sizes for the secondary cache vary from 32 bytes to 256 bytes (depending on the type of processor). The size of the secondary cache ranges from 128 KB to 8 MB. The tertiary cache is not available on all Alpha CPUs; otherwise, it is identical to the secondary cache.

2.3.2 Memory and Swap Space Configuration

You must determine the total amount of memory and swap space that you need to handle your workload. Insufficient memory resources and swap space will cause performance problems. In addition, your memory bank configuration will affect performance.

To configure memory and swap space, perform the following tasks:

1. Determine how much physical memory your configuration requires and choose a system that provides the necessary memory and has enough backplane slots for memory boards (Section 2.3.2.1).
2. Choose a swap space allocation mode (Section 2.3.2.2).
3. Determine how much swap space you need (Section 2.3.2.3).
4. Configure swap disks in order to efficiently distribute the disk I/O (Section 6.2).

The following sections describe these tasks.

2.3.2.1 Determining Your Physical Memory Requirements

You must have enough system memory to provide an acceptable level of user and application performance. The amount of memory installed in your system must be at least as much as the sum of the following:

- The total amount of memory that will be wired, including operating system data and text, system tables, the metadata and AdvFS buffer caches, and dynamically allocated data structures
- The total amount of memory your processes need for anonymous memory, which holds data elements and structures that are modified during process execution (heap space, stack space, and space allocated by the `malloc` function)

- The total amount of memory that the UBC requires to cache file system data

In addition, each network connection to your server requires the following memory resources:

- Kernel socket structure
- Internet protocol control block (`inpcb`) structure
- TCP control block structure
- Any additional socket buffer space that is needed as packets arrive and are consumed

These memory resources total 1 KB for each connection endpoint (not including the socket buffer space), so you need 10 MB of memory in order to accommodate 10,000 connections. There is no limit on a system's ability to handle millions of TCP connections, if you have enough memory resources to service the connections. However, when memory is low, the server will reject new connection requests until enough existing connections are freed. Use the `netstat -m` command to display the memory that is currently being used by the network subsystem.

To ensure that your server has enough memory to handle high peak loads, you should have available 10 times the memory that is needed on a busy day. For optimal performance and for scalability, configure more than the minimum amount of memory needed.

2.3.2.2 Choosing a Swap Space Allocation Mode

There are two modes that you can use to allocate swap space. The modes differ in how the virtual memory subsystem reserves swap space for anonymous memory (modifiable virtual address space). Anonymous memory is memory that is not backed by a file, but is backed by swap space (for example, stack space, heap space, and memory allocated by the `malloc` function).

There is no performance benefit attached to either mode; however, deferred mode is recommended for large-memory systems. The swap space allocation modes are as follows:

- **Immediate mode**—This mode reserves swap space when a process first allocates anonymous memory. Immediate mode is the default swap space allocation mode and is also called **eager mode**.

  Immediate mode may cause the system to reserve an unnecessarily large amount of swap space for processes. However, it ensures that swap space will be available to processes if it is needed.

- **Deferred mode**—This mode reserves swap space only if the virtual memory subsystem needs to write a modified virtual page to swap space. It postpones the reservation of swap space for anonymous memory until it is actually needed. Deferred mode is also called **lazy mode**.

  Deferred mode requires less swap space than immediate mode and may cause the system to run faster because it requires less swap space bookkeeping. However, because deferred mode does not reserve swap space in advance, the swap space may not be available when a process needs it, and the process may be killed asynchronously.

  You can enable the deferred swap space allocation mode by removing or moving the `/sbin/swapdefault` file.

See the *System Administration* manual for more information on swap space allocation methods.

2.3.2.3 Determining Swap Space Requirements

Swap space is used to hold the recently accessed modified pages from processes and from the UBC. In addition, if a crash dump occurs, the operating system writes all or part of physical memory to swap space.

It is important to configure a sufficient amount of swap space and to distribute swap space across multiple disks. An insufficient amount of swap space can severely degrade performance and prevent processes from running or completing. A minimum of 128 MB of swap space is recommended.

The optimal amount of swap space for your configuration depends on the following factors:

- Total amount of memory configured in the system

  In general, systems with large amounts of memory require less swap space than low-memory systems.

- Your applications' anonymous memory requirements

  Anonymous memory holds data elements and structures that are modified during process execution, such as heap space, stack space, and memory allocated with the `malloc` function.

- Crash dump configuration

  Swap space should be large enough to hold a complete crash dump file. Tru64 UNIX supports compressed crash dumps, which require less swap space to hold the dump file than uncompressed crash dumps. See the *System Administration* manual for more information on compressed crash dumps.

- Swap space allocation mode

  Less swap space is required if you are using deferred mode, instead of immediate mode.

To calculate the amount of swap space required by your configuration, first identify the total amount of anonymous memory (modifiable virtual address space) required by all of your processes. If you are using immediate mode, the optimal amount of swap space will be this value plus 10 percent. If you are using deferred mode, the optimal amount of swap space will be half the total anonymous memory requirements.

You can configure swap space when you first install the operating system, or you can add swap space at a later date. See Section 6.2 for information about adding swap space after installation and configuring swap space for high-performance.

2.3.3 I/O Capacity

Systems provide support for different numbers of storage shelves and I/O buses to which you can connect external storage devices and network adapters. Some enterprise systems provide up to 132 PCI slots for external storage. These systems are often used in VLDB systems and cluster configurations.

You must ensure that the system you choose has sufficient I/O buses and slots available for your disk storage and network configuration.

2.3.4 High-Performance Disk Storage Support

Systems support different local disk storage configurations, including multiple storage shelves, large disk capacity, and UltraSCSI devices. In addition, some systems support high-performance PCI buses, which are required for hardware RAID subsystems and clusters.

You must ensure that the system you choose supports the disk storage configuration that you need. See Section 2.4 and Section 2.5 for more information on disk storage configurations.

2.3.5 High-Performance Network Support

Systems support various networks and network adapters that provide different performance features. For example, an Asynchronous Transfer Mode (ATM) high-performance network is ideal for applications that need the high speed and the low latency (switched, full duplex network infrastructure) that ATM networks provide.

In addition, you can configure multiple network adapters or use NetRAIN to increase network access and provide high network availability.

2.4 Choosing High-Performance Disk Storage Hardware

The disk storage subsystem is used for both data storage and for swap space. Therefore, an incorrectly configured or tuned disk subsystem can degrade both disk I/O and virtual memory performance. Using your resource model, as described in Section 2.1, choose the disk storage hardware that will meet your performance needs.

Table 2–3 describes some hardware options that can be used in a high-performance disk storage configuration and the performance benefit for each option.

Table 2–3: High-Performance Disk Storage Hardware Options

Hardware Option	Performance Benefit
Fast disks (Section 2.4.1)	Improves disk access time and sequential data transfer performance
Solid-state disks (Section 2.4.2)	Provides very low disk access time
Wide devices (Section 2.4.3)	Provides high bandwidth and improves performance for large data transfers
High-performance host bus adapters (Section 2.4.4)	Increases bandwidth and throughput, and supports wide data paths and fast bus speeds
DMA host bus adapters (Section 2.4.5)	Relieves CPU of data transfer overhead
RAID controllers (Section 2.4.6 and Section 8.4)	Decreases CPU overhead; increases the number of disks that can be connected to an I/O bus; provides RAID functionality; and optionally provides write-back caches
Fibre Channel (Section 2.4.7)	Provides high access speeds and other high-performance features
Prestoserve (Section 2.4.8)	Improves synchronous write performance

For detailed information about hardware performance features, see the Compaq *Systems & Options Catalog*. For information about operating system hardware support, see the Tru64 UNIX Version 4.0F *Software Product Description*.

The following sections describe some of these high-performance disk storage hardware options in detail.

2.4.1 Fast Disks

Disks that spin with a high rate of revolutions per minute (RPM) have a low disk access time (latency). High-RPM disks are especially beneficial to the performance of sequential data transfers.

High-performance disks (7200 RPM) can improve performance for many transaction processing applications (TPAs). UltraSCSI disks (10,000 RPM) are ideal for demanding applications, including network file servers and Internet servers, that require high bandwidth and high throughput.

2.4.2 Solid-State Disks

Solid-state disks provide outstanding performance in comparison to magnetic disks, but at a higher cost. By eliminating the seek and rotational latencies that are inherent in magnetic disks, solid-state disks can provide very high disk I/O performance. Disk access time is under 100 microseconds, which allows you to access critical data more than 100 times faster than with magnetic disks.

Available in both wide (16-bit) and narrow (8-bit) versions, solid-state disks are ideal for response-time critical applications with high data transfer rates, such as online transaction processing (OLTP), and applications that require high bandwidth, such as video applications.

Solid-state disks complement hardware RAID configurations by eliminating bottlenecks caused by random workloads and small data sets. Solid-state disks also provide data reliability through a nonvolatile data-retention system.

For the best performance, use solid-state disks for your most frequently accessed data to reduce the I/O wait time and CPU idle time. In addition, connect the disks to a dedicated bus and use a high-performance host bus adapter.

2.4.3 Devices with Wide Data Paths

Disks, host bus adapters, SCSI controllers, and storage expansion units support wide data paths, which provide nearly twice the bandwidth of narrow data paths. Wide devices can greatly improve I/O performance for large data transfers.

Disks with wide (16-bit) data paths provide twice the bandwidth of disks with narrow (8-bit) data paths. To obtain the performance benefit of wide disks, all the disks on a SCSI bus must be wide. If you use both wide and narrow disks on the same SCSI bus, the bus performance will be constrained by the narrow disks.

2.4.4 High-Performance Host Bus Adapters

Host bus adapters and interconnects provide different performance features at various costs. For example, FWD (fast, wide, and differential) SCSI bus adapters provide high bandwidth and high throughput connections to disk devices. Other adapters support UltraSCSI.

In addition, some host bus adapters provide dual-port (dual-channel) support, which allows you to connect two buses to one I/O bus slot.

Bus speed (the rate of data transfers) depends on the host bus adapter. Different adapters support bus speeds ranging from 5 million bytes per second (5 MHz) for slow speed to 40 million bytes per second (20 MHz) for UltraSCSI.

You must use high-performance host bus adapters, such as the KZPSA adapter, to connect systems to high-performance RAID array controllers.

2.4.5 DMA Host Bus Adapters

Some host bus adapters support direct memory access (DMA), which enables an adapter to bypass the CPU and go directly to memory to access and transfer data. For example, the KZPAA is a DMA adapter that provides a low-cost connection to SCSI disk devices.

2.4.6 RAID Controllers

RAID controllers are used in hardware RAID subsystems, which greatly expand the number of disks connected to a single I/O bus, relieve the CPU of the disk I/O overhead, and provide RAID functionality and other high-performance and high-availability features.

There are various types of RAID controllers, which provide different features. High-performance RAID array controllers support dynamic parity RAID and battery-backed write-back caches. Backplane RAID storage controllers provide a low-cost RAID solution.

See Section 8.4 for more information about hardware RAID subsystems.

2.4.7 Fibre Channel

Fibre Channel is a high-performance I/O bus that is an example of serial SCSI and provides network storage capabilities. Fibre Channel supports multiple protocols, including SCSI, Intellitan Protocol Interface (IPI), TCP/IP, and High-Performance Peripheral Interface (HIPPI).

Fibre Channel is based on a network of intelligent switches. Link speeds are available up to 100 MB/sec full duplex. Although Fibre Channel is more expensive than parallel SCSI, Fibre Channel Arbitrated Loop (FC-AL) decreases costs by eliminating the Fibre Channel fabric and using connected nodes in a loop topology with simplex links. In addition, an FC-AL loop can connect to a Fibre Channel fabric.

2.4.8 Prestoserve

Prestoserve uses a nonvolatile, battery-backed memory cache to improve synchronous write performance. Prestoserve temporarily caches file system writes that otherwise would have to be written to disk. This capability improves performance for systems that perform large numbers of synchronous writes.

To optimize Prestoserve cache use, you may want to enable Prestoserve only on the most frequently used file systems. Prestoserver can greatly improve performance for NFS servers.

You cannot use Prestoserve in a cluster or for nonfile system I/O.

2.5 Choosing How to Manage Disks

Disk configurations vary in capacity, performance features, and degree of availability. Use your workload resource model, as described in Section 2.1, to identify the disk configuration that will meet your performance needs.

Table 2–4 describes some disk storage management options and the performance benefit and availability impact for each option.

Table 2–4: High-Performance Disk Storage Configuration Solutions

Configuration Option	Performance Benefit
Shared pool of storage (LSM) (Section 2.5.1)	Facilitates management of large amounts of storage
Disk striping (RAID 0) (Section 2.5.2)	Distributes disk I/O and improves throughput, but decreases availability
RAID 3 (Section 2.5.3)	Improves bandwidth and provides availability
RAID 5 (Section 2.5.3)	Improves throughput and provides availability

Table 2–4: High-Performance Disk Storage Configuration Solutions (cont.)

Configuration Option	Performance Benefit
Dynamic parity RAID (Section 2.5.3)	Improves overall disk I/O performance and provides availability
Disk mirroring (RAID 1) (Section 2.6.2)	Improves read performance and provides high availability, but decreases write performance

Not only is it important to choose the correct disk storage configuration, you also must follow the configuration guidelines, as described in Chapter 8.

The following sections describe some of these high-performance storage configurations in detail. See Section 2.6 for information about high-availability configurations.

2.5.1 Using a Shared Pool of Storage for Flexible Management

There are two methods that you can use to manage the physical disks in your environment. The traditional method of managing disks and files is to divide each disk into logical areas called disk partitions, and to then create a file system on a partition or use a partition for raw I/O.

Each disk type has a default partition scheme. The `disktab` database file lists the default disk partition sizes. The size of a partition determines the amount of data it can hold. It can be time-consuming to modify the size of a partition. You must back up any data in the partition, change the size by using the `disklabel` command, and then restore the data to the resized partition.

An alternative to managing disks with static disk partitions is to use the Logical Storage Manager (LSM) to set up a shared pool of storage that consists of multiple disks. You can create virtual disks (LSM volumes) from this pool of storage, according to your performance and capacity needs, and then place file systems on the volumes or use them for raw I/O.

LSM provides you with flexible and easy management for large storage configurations. Because there is no direct correlation between a virtual disk and a physical disk, file system or raw I/O can span disks as needed. In addition, you can easily add disks to and remove disks from the pool, balance the load, and perform other storage management tasks. LSM also provides you with high-performance and high-availability RAID functionality, hot spare support, and load balancing.

See Section 8.3 for information about LSM configurations.

2.5.2 Striping Disks to Distribute I/O

Disk striping (RAID 0) distributes disk I/O and can improve throughput. The striped data is divided into blocks (sometimes called chunks or stripes) and distributed across multiple disks in a array. Striping enables parallel I/O streams to operate concurrently on different devices, so that I/O operations can be handled simultaneously by multiple devices.

Disk striping requires LSM or a hardware RAID subsystem.

The performance benefit of striping depends on the size of the stripe and how your users and applications perform disk I/O. For example, if an application performs multiple simultaneous I/O operations, you can specify a stripe size that will enable each disk in the array to handle a separate I/O operation. If an application performs large sequential data transfers, you can specify a stripe size that will distribute a large I/O evenly across the disks.

For volumes that receive only one I/O at a time, you may not want to use striping if access time is the most important factor. In addition, striping may degrade the performance of small data transfers, because of the latencies of the disks and the overhead associated with dividing a small amount of data.

Striping decreases data availability because one disk failure makes the entire disk array unavailable. To make striped disks highly available, you can combine RAID 0 with RAID 1 to mirror the striped disks.

See Chapter 8 for more information about LSM and hardware RAID subsystems.

2.5.3 Using Parity RAID to Improve Disk Performance

Hardware RAID subsystems support parity RAID for high performance and high availability. Tru64 UNIX supports three types of parity RAID, each with different performance and availability benefits:

- RAID 3—Divides data blocks and distributes the data across a disk array, providing parallel access. RAID 3 provides a high data transfer rate and increases bandwidth, but it provides no improvement in throughput (the I/O transaction rate).

  Use RAID 3 to improve the I/O performance of applications that transfer large amounts of sequential data. RAID 3 provides no improvement for applications that perform multiple I/O operations involving small amounts of data.

  RAID 3 also provides high data availability by storing redundant parity information on a separate disk. The parity information is used to regenerate data if a disk in the array fails. However, performance degrades as multiple disks fail, and data reconstruction is slower than if you had used mirroring.

- RAID 5—Distributes data blocks across disks in an array. RAID 5 allows independent access to data and can handle simultaneous I/O operations.

  Use RAID 5 to improve throughput, especially for large file I/O operations, multiple small data transfers, and I/O read operations. RAID 5 is not suitable for write-intensive applications.

  RAID 5 also provides high data availability by distributing redundant parity information across disks. Each array member contains enough parity information to regenerate data if a disk fails. However, performance may degrade and data may be lost if multiple disks fail. In addition, data reconstruction is slower than if you had used mirroring.

- Dynamic parity RAID—Dynamically adjusts, according to the workload, between data transfer-intensive algorithms and I/O operation-intensive algorithms, combining the performance benefits of RAID 3 and RAID 5. Also known as adaptive RAID 3/5, dynamic parity RAID improves disk I/O performance for a wide variety of applications.

  Only high-performance RAID array controllers support dynamic parity RAID.

See Section 8.4 for more information about hardware RAID subsystems.

2.6 Choosing a High-Availability Configuration

You can set up a configuration that provides the level of availability that you need. For example, you can make only disk data highly available or you can set up a cluster configuration with no single point of failure, as shown in Figure 1–4.

Table 2–5 lists each possible point of failure, the configuration solution that will provide high availability, and any performance benefits and tradeoffs.

Table 2–5: High-Availability Configurations

Point of Failure	Configuration Solution	Benefits and Tradeoffs
Single system	Latest hardware, firmware, and operating system releases	Provides the latest hardware and software enhancements, but may require down time during upgrade
	Cluster with at least two systems (Section 2.6.1)	Improves overall performance by spreading workload across member systems, but increases costs and management complexity
Multiple systems	Cluster with more than two members (Section 2.6.1)	Improves overall performance by spreading workload across member systems, but increases costs and management complexity
Cluster interconnect	Second cluster interconnect (Section 2.6.1)	Increases costs
Disk	Mirrored disks (Section 2.6.2)	Improves read performance, but increases costs and decreases write performance
	Parity RAID (Section 2.6.2)	Improves disk I/O performance, but increases management complexity and decreases performance when under heavy write loads and in failure mode
Host bus adapter or bus	Mirrored data across disks on different buses (Section 2.6.2)	Improves read performance, but increases costs and decreases write performance
Network connection	Multiple network connections or use NetRAIN (Section 2.6.3)	Improves network access and possibly performance, but increases costs
System cabinet power supply	Redundant power supplies (Section 2.6.4)	Increases costs

Table 2–5: High-Availability Configurations (cont.)

Point of Failure	Configuration Solution	Benefits and Tradeoffs
Storage unit power supply	Redundant power supplies or mirrored disks across cabinets with independent power supplies (Section 2.6.4 and Section 2.6.2)	Increases costs
Total power supply	Battery-backed uninterruptible power system (UPS) (Section 2.6.4)	Increases costs

The following sections describe some of the previous high-availability configurations in detail.

2.6.1 Using a Cluster for System Availability

If users and applications depend on the availability of a single system for CPU, memory, data, and network resources, they will experience down time if a system crashes or an application fails. To make systems and applications highly available, you must use the TruCluster products to set up a cluster.

A cluster is a loosely coupled group of servers configured as member systems and connected to highly available shared disk storage and common networks. Software applications are installed on every member system, but only one system runs an application at one time.

A cluster utilizes a failover mechanism to protect against failures. If a member system fails, all cluster-configured applications running on that system will fail over to a viable member system; that is, the new system will start the applications and make them available to users. Use more than two member systems in a cluster to protect against multiple system failures.

Cluster products include TruCluster Available Server Software and TruCluster Production Server Software, which supports a high-performance cluster interconnect that enables fast and reliable communications between members. To protect against interconnect failure, use redundant cluster interconnects.

You can use only specific systems, host bus adapters, RAID controllers, and disks with the cluster products. In addition, member systems must have enough I/O bus slots for adapters, controllers, and interconnects.

See a specific cluster product's *Software Product Description* for detailed information about the product.

2.6.2 Using RAID for Disk Data Availability

RAID technology provides you with high data availability, in addition to high performance. RAID 1 (disk mirroring) provides high data availability by maintaining identical copies of data on different disks in an array. If the original disk fails, the copy is still available to users and applications. To protect data against a host bus adapter or bus failure, mirror the data across disks located on different buses.

Mirroring disks can improve read performance because data can be read from two different locations. However, it decreases disk write performance, because data must be written to two different locations.

Disk mirroring requires LSM or a hardware RAID subsystem.

Hardware RAID subsystems also provide high data availability and high performance by using parity RAID, in which data is spread across disks and parity information is used to reconstruct data if a failure occurs. Tru64 UNIX supports three types of parity RAID (RAID 3, RAID 5, and dynamic parity RAID), and each provides different performance and availability benefits. See Section 2.5.3 for more information about parity RAID.

See Chapter 8 for more information about disk storage configurations.

2.6.3 Using Redundant Networks

Network connections may fail because of a failed network interface or a problem in the network itself. You can make the network connection highly available by using redundant network connections. If one connection becomes unavailable, you can still use the other connection for network access. Whether you can use multiple networks depends on the application, network configuration, and network protocol.

You can also use NetRAIN (redundant array of independent network adapters) to configure multiple interfaces on the same LAN segment into a single interface, and to provide failover support for network adapter and network connections. One interface is always active while the other interfaces remain idle. If the active interface fails, an idle interface is brought online within less than 10 seconds.

NetRAIN supports only Ethernet and FDDI.

See `nr`(7) for more information about NetRAIN. See the *Network Administration* manual for information about network configuration. See Chapter 10 for information about improving network performance.

2.6.4 Using Redundant Power Sources

To protect against a cabinet power supply failure, use redundant power supplies from different power sources. For disk storage units, you can mirror disks across cabinets with independent power supplies.

In addition, use an uninterruptible power system (UPS) to protect against a total power failure (for example, the power in a building fails). A UPS depends on a viable battery source and monitoring software.

3 Monitoring Systems and Diagnosing Performance Problems

You must gather a wide variety of performance information in order to identify performance problems or areas where performance is deficient.

Some symptoms or indications of performance problems are obvious. For example, applications complete slowly or messages appear on the console indicating that the system is out of resources. Other problems or performance deficiencies are not obvious and can be detected only by monitoring system performance.

This chapter describes how to perform the following tasks:

- Understand how the system logs event messages (Section 3.1)
- Set up system accounting and disk quotas to track and control resource utilization (Section 3.2)
- Use tools to gather a variety of performance information (Section 3.3)
- Establish a method to continuously monitor system performance (Section 3.4)
- Profile and debug kernels (Section 3.5)

After you identify a performance problem or an area in which performance is deficient, you can identify an appropriate solution. See Chapter 4 for information about improving system performance.

3.1 Obtaining Information About System Events

It is recommended that you set up a routine to continuously monitor system events and to alert you when serious problems occur. In addition to helping you diagnose performance problems, periodically examining event and log files allows you to correct a problem before it impacts performance or availability.

The operating system uses the system event logging facility and the binary event logging facility to log system events. The system event logging facility uses the `syslog` function to log events in ASCII format. The `syslogd` daemon collects the messages logged from the various kernel, command, utility, and application programs. This daemon then writes the messages to a local file or forwards the messages to a remote system, as specified in the `/etc/syslog.conf` event logging configuration file. You should periodically monitor these ASCII log files for performance information.

The binary event logging facility detects hardware and software events in the kernel and logs detailed information in binary format records. The binary event logging facility uses the `binlogd` daemon to collect various event log records. The daemon then writes these records to a local file or forwards the records to a remote system, as specified in the `/etc/binlog.conf` default configuration file.

You can examine the binary event log files by using the following methods:

- The DECevent utility continuously monitors system events through the binary event logging facility, decodes events, and tracks the number and the severity of events logged by system devices. DECevent can analyze system events and provides a notification mechanism (for example, mail) that can warn of potential problems.

  You must register a license to use DECevent's analysis and notification features. These features may be available as part of your service agreement. A license is not needed to use DECevent to translate the binary log file to ASCII format.

- You can use the `uerf` command to translate binary log files to ASCII format. See `uerf`(8) for information.

In addition, it is recommended that you configure crash dump support into the system. Significant performance problems may cause the system to crash, and crash dump analysis tools can help you diagnose performance problems.

See the *System Administration* manual for more information about event logging and crash dumps.

3.2 Using System Accounting and Disk Quotas

It is recommended that you set up system accounting, which allows you to obtain information about the amount of CPU usage and connect time, the number of processes spawned, memory and disk usage, the number of I/O operations, and the number of printing operations by each user.

In addition, you should establish Advanced File System (AdvFS) and UNIX File System (UFS) **disk quotas** in order to track and control disk usage. Disk quotas allow you to limit the disk space available to users and to monitor disk space usage.

See the *System Administration* manual for information about setting up system accounting and UFS disk quotas. See the *AdvFS Administration* manual for information about AdvFS quotas.

3.3 Gathering Performance Information

There are various commands and utilities that you can use to obtain a comprehensive understanding of your system performance. It is important that you gather statistics under a variety of conditions. Comparing sets of data will help you to diagnose performance problems.

For example, to determine how an application impacts system performance, you can gather performance statistics without the application running, start the application, and then gather the same statistics. Comparing different sets of data will enable you to identify whether the application is consuming memory, CPU, or disk I/O resources.

Be sure to gather information at different stages during the application processing to obtain comprehensive performance information. For example, an application may be I/O-intensive during one stage and CPU-intensive during another.

There are three commands that you can use to obtain some fundamental performance information:

- `vmstat`

  The primary source of performance problems is a lack of memory, which can affect response time and application completion time. In addition, a lack of CPU resources can also result in long application completion times.

  Use the `vmstat` command to determine if the system is using a lot of memory resources or CPU cycles. The command output shows the number of free pages and the percentages of user, system, and idle CPU times. See Section 6.3.2 for information.

- `iostat`

  If your disk I/O load is not spread evenly among the available disks, bottlenecks may occur at the disks that are being excessively used. Use the `iostat` command to determine if disk I/O is being evenly distributed. The command output shows which disks are being excessively used. See Section 8.2.1 for information.

- `swapon -s`

  Insufficient swap space for your workload can result in poor application performance and response time. To check swap space, use the `swapon -s` command. The command output shows the total amount of swap space and the percentage of swap space that is being used. See Section 6.3.3 for information.

To obtain application performance information, use profiling tools to collect statistics on CPU usage, call counts, call cost, memory usage, and I/O operations at various levels (for example, at a procedure level or at an instruction level). Profiling identifies sections of application code that consume large portions of execution time, so that you can focus on improving code efficiency in these sections.

There are many tools that you can use to query subsystems, profile the system kernel and applications, and collect CPU statistics. See the following tables for information:

Kernel profiling and debugging	Table 3–1
Memory resource monitoring	Table 6–1
CPU monitoring	Table 7–1
Disk operation monitoring	Table 8–1
LSM monitoring	Table 8–7
AdvFS monitoring	Table 9–4
UFS monitoring	Table 9–7
NFS monitoring	Table 9–9
Network subsystem monitoring	Table 10–1
Application profiling and debugging	Table 11–1

3.4 Continuously Monitoring Performance

You may want to set up a routine to continuously monitor system performance. Some monitoring tools will alert you when serious problems occur (for example, sending mail). It is important that you choose a continuous monitoring tool that does not impact system performance.

The following tools allow you to continuously monitor performance:

- Performance Manager

  Simultaneously monitors multiple Tru64 UNIX systems, detects performance problems, and performs event notification. See Section 3.4.1 for more information.

- Performance Visualizer

  Graphically displays the performance of all significant components of a parallel system. Using Performance Visualizer, you can monitor the performance of all the member systems in a cluster. See Section 3.4.2 for more information.

- Digital Continuous Profiling Infrastructure

  Provides continuous, low-overhead system profiling and allows you to track cycles during program execution. See Section 3.4.3 for information.

- `monitor`

  Collects a variety of performance data on a running system and either displays the information or saves it to a binary file. The `monitor` utility is available on the Tru64 UNIX Freeware CD-ROM. See `ftp://gatekeeper.dec.com/pub/DEC` for information.

- `top`

  Provides continuous reports on the state of the system, including a list of the processes using the most CPU resources. The `top` command is available on the Tru64 UNIX Freeware CD-ROM. See `ftp://eecs.nwu.edu/pub/top` for information.

- `tcpdump`

  Continuously monitors the network traffic associated with a particular network service and allows you to identify the source of a packet. See `tcpdump`(8) for information.

- `nfswatch`

  Continuously monitors all incoming network traffic to a Network File System (NFS) server, and displays the number and percentage of packets received. See `nfswatch`(8) for information.

- `xload`

  Displays the system load average in a histogram that is periodically updated. See `xload`(1X) for information.

- `volstat`

  Provides information about activity on volumes, plexes, subdisks, and disks under LSM control. The `volstat` utility reports statistics that reflect the activity levels of LSM objects since boot time or since you reset the statistics. See Section 8.3.4.2 for information.

- `volwatch`

  Monitors the Logical Storage Manager (LSM) for failures in disks, volumes, and plexes, and sends mail if a failure occurs. See Section 8.3.4.4 for information.

The following sections describe the Performance Manager and Performance Visualizer products and the `dcpi` tool.

3.4.1 Using Performance Manager

Performance Manager (PM) for Tru64 UNIX allows you to simultaneously monitor multiple Tru64 UNIX systems, so you can detect and correct performance problems. PM can operate in the background, alerting you to performance problems. Monitoring only a local node does not require a PM license. However, a license is required to monitor multiple nodes and clusters.

PM gathers and displays Simple Network Protocol (SNMP and eSNMP) data for the systems you choose, and allows you to detect and correct performance problems from a central location. PM has a graphical user interface (GUI) that runs locally and displays data from the monitored systems.

Use the GUI to choose the systems and data that you want to monitor. You can customize and extend PM, so you can create and save performance monitoring sessions. Graphs and charts can show hundreds of different system values, including CPU performance, memory usage, disk transfers, file-system capacity, network efficiency, database performance, and AdvFS and cluster-specific metrics. Data archives can be used for high-speed playback or long-term trend analysis.

PM provides comprehensive thresholding, rearming, and tolerance facilities for all displayed metrics. You can set a threshold on every key metric, and specify the PM reaction when a threshold is crossed. For example, you can configure PM to send mail, to execute a command, or to display a notification message.

PM also has performance analysis and system management scripts, as well as cluster-specific and AdvFS-specific scripts. Run these scripts separately to target specific problems, or run them simultaneously to check the general system performance. The PM analyses include suggestions for eliminating problems. PM can monitor both individual cluster members and an entire cluster concurrently.

See the Performance Manager online documentation for more information.

3.4.2 Using Performance Visualizer

Performance Visualizer is a valuable tool for developers of parallel applications. Because it monitors the performance of several systems simultaneously, it allows you to see the impact of a parallel application on all the systems, and to ensure that the application is balanced across all systems. When problems are identified, you can change the application code and use Performance Visualizer to evaluate the effects of these changes. Performance Visualizer is a Tru64 UNIX layered product and requires a license.

Performance Visualizer also helps you identify overloaded systems, underutilized resources, active users, and busy processes.

Using Performance Visualizer, you can monitor the following:

- CPU utilization by each CPU in a multiprocessing system
- Load average
- Use of paged memory
- Paging events, which indicate how much a system is paging
- Use of swap space
- Behavior of individual processes

You can choose to look at all of the hosts in a parallel system or at individual hosts. See the Performance Visualizer documentation for more information.

3.4.3 Using Digital Continuous Profiling Infrastructure

Use the Digital Continuous Profiling Infrastructure (`dcpi`) tool to provide low-overhead system profiling and to track cycles during program execution. Using `dcpi`, you can continuously profile entire systems, including the kernel, user programs, drivers, and shared libraries, which may enable you to make coding improvements.

The `dcpi` tool maintains a database of profile information that is updated incrementally for every executable image that runs. A suite of profile analysis tools analyzes the profile information at various levels. For example, you can determine the percentage of CPU cycles that were used to execute the kernel and each user program. You can also determine how long a specific instruction stalled, on average, because of a D-cache miss.

The `dcpi` tool may not support all configurations.

The `dcpi` tool is available from the Systems Research Center at the following World Wide Web location:

`http://www.research.digital.com/SRC/dcpi`

3.5 Profiling and Debugging Kernels

Table 3–1 describes the tools that you can use to profile and debug the kernel. Detailed information about these profiling and debugging tools is located in the *Kernel Debugging* manual and in the tools' reference pages.

Table 3–1: Kernel Profiling and Debugging Tools

Name	Use	Description
`prof`	Analyzes profiling data	Analyzes profiling data and produces statistics showing which portions of code consume the most time and where the time is spent (for example, at the routine level, the basic block level, or the instruction level). The `prof` command uses as input one or more data files generated by the `kprofile`, `uprofile`, or `pixie` profiling tools. The `prof` command also accepts profiling data files generated by programs linked with the `-p` switch of compilers such as `cc`. See `prof`(1) for more information.
`kprofile`	Produces a program counter profile of a running kernel	Profiles a running kernel using the performance counters on the Alpha chip. You analyze the performance data collected by the tool with the `prof` command. See `kprofile`(1) for more information.
`dbx`	Debugs running kernels, programs, and crash dumps, and examines and temporarily modifies kernel variables	Provides source-level debugging for C, Fortran, Pascal, assembly language, and machine code. The `dbx` debugger allows you to analyze crash dumps, trace problems in a program object at the source-code level or at the machine code level, control program execution, trace program logic and flow of control, and monitor memory locations. Use `dbx` to debug kernels, debug stripped images, examine memory contents, debug multiple threads, analyze user code and applications, display the value and format of kernel data structures, and temporarily modify the values of some kernel variables. See `dbx`(8) for more information.

Table 3–1: Kernel Profiling and Debugging Tools (cont.)

Name	Use	Description
`kdbx`	Debugs running kernels and crash dumps	Allows you to examine a running kernel or a crash dump. The `kdbx` debugger, a frontend to the `dbx` debugger, is tailored specifically to debugging kernel code and displays kernel data in a readable format. The debugger is extensible and customizable, allowing you to create commands that are tailored to your kernel debugging needs. You can also use extensions to check resource usage (for example, CPU usage). See `kdbx`(8) for more information.
`ladebug`	Debugs kernels and applications	Debugs programs and the kernel and helps locate run-time programming errors. The `ladebug` symbolic debugger is an alternative to the `dbx` debugger and provides both command-line and graphical user interfaces and support for debugging multithreaded programs. See the *Ladebug Debugger Manual* and `ladebug`(1) for more information.

4 Improving System Performance

You may be able to improve Tru64 UNIX performance by tuning the operating system or performing other tasks. You may need to tune the system under the following circumstances:

- You are running a large or specialized configuration that requires a specific type of tuning.
- You want to optimize performance in a generally well-functioning system.
- You want to solve a specific performance problem.

To help you improve system performance, this chapter describes how to perform the following tasks:

- Apply any configuration-specific tuning recommendations (Section 4.1)
- Run `sys_check` and consider applying its configuration and tuning recommendations (Section 4.2)
- Use the advanced tuning recommendations described in this manual (Section 4.3)
- Modify the kernel in order to tune subsystems (Section 4.4)
- Identify and solve some common performance problems (Section 4.5)

In order to effectively tune a system, you must identify the area in which performance is deficient. See Chapter 3 for information about monitoring the system.

4.1 Tuning Special Configurations

Large configurations or configurations that run memory-intensive or network-intensive applications may require special tuning. The following sections provide information about tuning these special configurations:

- Internet servers (Section 4.1.1)
- Large-memory servers (Section 4.1.2)
- NFS servers (Section 4.1.3)

In addition, your application product documentation may include specific configuration and tuning recommendations.

4.1.1 Tuning Internet Servers

Internet servers (including Web, proxy, firewall, and gateway servers) run network-intensive applications that usually require significant system resources. If you have an Internet server, it is recommended that you modify the default values of some kernel attributes.

Follow the recommendations in Table 4–1 to help you tune an Internet server.

Table 4–1: Internet Server Tuning Recommendations

Action	Reference
Increase the system resources available to processes.	Section 5.1
Increase the available address space.	Section 5.3.1
Increase the number of memory-mapped files in a user address.	Section 5.3.2
Increase the number of pages with individual protections.	Section 5.3.3
Ensure that the Unified Buffer Cache (UBC) has sufficient memory.	Section 9.2.3
Increase the size of the hash table that the kernel uses to look up TCP control blocks.	Section 10.2.1
Increase the number of TCP hash tables.	Section 10.2.2
Increase the limits for partial TCP connections on the socket listen queue.	Section 10.2.3
For only proxy servers, increase the maximum number of concurrent nonreserved, dynamically allocated ports.	Section 10.2.4
Disable use of a path maximum transmission unit (PMTU).	Section 10.2.16
Increase the number of IP input queues.	Section 10.2.19
For only proxy servers, enable `mbuf` cluster compression.	Section 10.2.21

4.1.2 Tuning Large-Memory Systems

Large memory systems often run memory-intensive applications, such as database programs, that usually require significant system resources. If you have a large memory system, it is recommended that you modify the default values of some kernel attributes.

Follow the recommendations in Table 4–2 to help you tune a large-memory system.

Table 4–2: Large-Memory System Tuning Recommendations

Action	Reference
Increase the system resources available to processes.	Section 5.1
Increase the size of a System V message and queue.	Section 5.4.1
Increase the maximum size of a single System V shared memory region.	Section 5.4.4
Increase the minimum size of a System V shared memory segment.	Section 5.4.5
Increase the available address space.	Section 5.3.1
Increase the maximum number of memory-mapped files that are available to a process.	Section 5.3.2
Increase the maximum number of virtual pages within a process' address space that can have individual protection attributes.	Section 5.3.3
Reduce the size of the AdvFS buffer cache.	Section 6.4.5
Increase the number of AdvFS buffer hash chains, if you are using AdvFS.	Section 9.3.4.2
Increase the memory reserved for AdvFS access structures, if you are using AdvFS.	Section 9.3.4.3
Increase the size of the metadata buffer cache to more than 3 percent of main memory, if you are using UFS.	Section 9.4.3.1
Increase the size of the metadata hash chain table, if you are using UFS.	Section 9.4.3.2

4.1.3 Tuning NFS Servers

NFS servers run only a few small user-level programs, which consume few system resources. File system tuning is important because processing NFS requests consumes the majority of CPU and wall clock time. See Chapter 9 for information on file system tuning.

In addition, if you are running NFS over TCP, tuning TCP may improve performance if there are many active clients. If you are running NFS over UDP, network tuning is not needed. See Section 10.2 for information on network subsystem tuning.

Follow the recommendations in Table 4–3 to help you tune a system that is only serving NFS.

Table 4–3: NFS Server Tuning Recommendations

Action	Reference
Set the value of the `maxusers` attribute to the number of server NFS operations that are expected to occur each second.	Section 5.1
Increase the size of the namei cache.	Section 9.2.1
Increase the number of AdvFS access structures, if you are using AdvFS.	Section 9.3.4.3
Increase the size of the metadata buffer cache, if you are using UFS.	Section 9.4.3.1

4.2 Checking the Configuration by Using the sys_check Utility

After you apply any configuration-specific tuning recommendations, as described in Section 4.1, run the `sys_check` utility to check your system configuration.

The `sys_check` utility creates an HTML file that describes the system configuration, and can be used to diagnose problems. The utility checks kernel attribute settings and memory and CPU resources, provides performance data and lock statistics for SMP systems and for kernel profiles, and outputs any warnings and tuning recommendations.

Consider applying the `sys_check` utility's configuration and tuning recommendations before applying any advanced tuning recommendations.

Note

You may experience impaired system performance while running the `sys_check` utility. Invoke the utility during offpeak hours to minimize the performance impact.

You can invoke the `sys_check` utility from the SysMan graphical user interface or from the command line. If you specify `sys_check` without any command-line options, it performs a basic system analysis and creates an HTML file with configuration and tuning recommendations. Options that you can specify at the command line include the following:

- The `-all` option provides information about all subsystems, including security information and `setld` inventory verification.
- The `-perf` option provides only performance data and excludes configuration data.

- The `-escalate` option creates escalation files required for reporting problems to Compaq.

See `sys_check`(8) for more information.

4.3 Using the Advanced Tuning Recommendations

If system performance is still deficient after applying the initial tuning recommendations for your configuration (see Section 4.1) and considering the `sys_check` recommendations (see Section 4.2), you may be able to improve performance by using the advanced tuning recommendations.

Before using the advanced tuning recommendations, you must:

- Understand your workload resource model, because not all tuning recommendations are appropriate for all configurations (see Section 2.1)
- Gather performance information to identify an area in which to focus your efforts (see Section 3.3 for information on using the `vmstat`, `iostat`, and `swapon` commands to obtain a basic understanding of system performance)

Use the advanced tuning guidelines shown in Table 4–4 to help you tune your system. In addition, Section 4.5 describes some solutions to common performance problems. Before implementing any tuning recommendation, you must ensure that it is appropriate for your configuration and workload and also consider its benefits and tradeoffs.

Table 4–4: Advanced Tuning Guidelines

If your workload consists of:	You can improve performance by:
Applications requiring extensive system resources	Increasing resource limits (Chapter 5)
Memory-intensive applications	Increasing the memory available to processes (Section 6.4)
	Modifying paging and swapping operations (Section 6.5)
	Reserving shared memory (Section 6.6)
CPU-intensive applications	Freeing CPU resources (Section 7.2)
Disk I/O-intensive applications	Distributing the disk I/O load (Section 8.1)
File system-intensive applications	Modifying AdvFS, UFS, or NFS operation (Chapter 9)

Table 4–4: Advanced Tuning Guidelines (cont.)

If your workload consists of:	You can improve performance by:
Network-intensive applications	Modifying network operation (Section 10.2)
Non-optimized or poorly-written applications applications	Optimizing or rewriting the applications (Chapter 11)

4.4 Modifying the Kernel

The operating system includes various subsystems that are used to define or extend the kernel. Kernel variables control subsystem behavior or track subsystem statistics since boot time.

Kernel variables are assigned default values at boot time. For certain configurations and workloads, especially memory or network-intensive systems, the default values of some attributes may not be appropriate, so you must modify these values to provide optimal performance.

There are several methods that you can use to modify kernel variable values:

- You can use the `dbx` debugger to directly change variable values on a running kernel. However, these changes are lost if you rebuild the kernel.
- In some cases, you can modify the system configuration file and rebuild the kernel, but this is not recommended.
- The preferred method of modifying kernel variable values is to access the variables by using subsystem attributes.

Each system includes different subsystems, depending on the configuration and the installed kernel options. For example, all systems include the mandatory subsystems, such as the `generic`, `vm`, and `vfs` subsystems. Other subsystems are optional, such as `presto`.

Most subsystems include one or more attributes. These attributes control or monitor some part of the subsystem. For example, the `vm` subsystem includes the `vm-page-free-swap` attribute, which controls when swapping starts. The `socket` subsystem includes the `sobacklog_hiwat` attribute, which monitors the maximum number of pending socket requests.

Kernel subsystem attributes are documented in reference pages. For example, `sys_attrs_advfs`(5) includes definitions for all the `advfs` subsystem attributes. See `sys_attrs`(5) for more information.

Subsystem attributes are managed by the configuration manager server, `cfgmgr`. You access subsystem attributes by using the `sysconfig` and sysconfigdb commands and by using the Kernel Tuner, dxkerneltuner, which is provided by the Common Desktop Environment (CDE).

You permanently modify an attribute value by including the modification in the /etc/sysconfigtab database file, using a special format. In some cases, you can modify attributes while the system is running. However, these run-time modifications are lost when the system reboots.

The following sections describe how to perform these tasks:

- Display the subsystems configured in a system (Section 4.4.1)
- Display the current values of subsystem attributes (Section 4.4.2)
- Display the maximum and minimum values of subsystem attributes (Section 4.4.3)
- Modify run-time (current) attribute values (Section 4.4.4)
- Permanently modify attribute values (Section 4.4.5)
- Use dbx to display and modify kernel variables (Section 4.4.6)

4.4.1 Displaying Kernel Subsystems

Use one of the following methods to display the kernel subsystems currently configured in your operating system:

- sysconfig -s

  This command displays the subsystems currently configured in the operating system. See sysconfig(8) for more information.

- Kernel Tuner

  This GUI displays the subsystems currently configured in the operating system. To access the Kernel Tuner, click on the Application Manager icon in the CDE menu bar, select System_Admin, and then select MonitoringTuning. You can then click on Kernel Tuner. A pop-up menu containing a list of subsystems appears.

The following example shows how to use the sysconfig -s command to display all the subsystems configured in the operating system:

```
# sysconfig -s
cm: loaded and configured
hs: loaded and configured
ksm: loaded and configured
generic: loaded and configured
io: loaded and configured
ipc: loaded and configured
proc: loaded and configured
sec: loaded and configured
socket: loaded and configured
rt: loaded and configured
bsd_tty: loaded and configured
```

```
xpr: loaded and configured
kdebug: loaded and configured
dli: loaded and configured
ffm_fs: loaded and configured
atm: loaded and configured
atmip: loaded and configured
lane: loaded and configured
atmifmp: loaded and configured
atmuni: loaded and configured
atmilmi3x: loaded and configured
uni3x: loaded and configured
bparm: loaded and configured
advfs: loaded and configured
net: loaded and configured
 .
 .
 .
```

4.4.2 Displaying Current Attribute Values

Use one of the following methods to display the current (run-time) value of an attribute:

- `sysconfig -q` *subsystem* [*attribute*]

  This command displays the current values for the attributes of the specified subsystem or the current value for only the specified attribute. See `sysconfig`(8) for more information.

- Kernel Tuner

  This GUI displays the current values of all the subsystem attributes. To access the Kernel Tuner, click on the Application Manager icon in the CDE menu bar, select System_Admin, and then select MonitoringTuning. You can then click on Kernel Tuner. A pop-up menu containing a list of subsystems appears. Select a subsystem to display the subsystem attributes and their current (run-time) values.

The following example shows how to use the `sysconfig -q` command to display the current values of the `vfs` subsystem:

```
# sysconfig -q vfs
vfs:
name-cache-size = 1029
name-cache-hash-size = 256
buffer-hash-size = 512
special-vnode-alias-tbl-size = 64
bufcache = 3
bufpages = 238
path-num-max = 64
sys-v-mode = 0
ucred-max = 256
nvnode = 468
max-vnodes = 6558
min-free-vnodes = 468
vnode-age = 120
namei-cache-valid-time = 1200
max-free-file-structures = 0
```

```
max-ufs-mounts = 1000
vnode-deallocation-enable = 1
pipe-maxbuf-size = 65536
pipe-single-write-max = -1
pipe-databuf-size = 8192
pipe-max-bytes-all-pipes = 81920000
noadd-exec-access = 0
fifo-do-adaptive = 1
```

Note

The current value of an attribute may not reflect a legal value, if you are not actually using a subsystem. For example, if you do not have an AdvFS fileset mounted, the current value of the `advfs` subsystem attribute `AdvfsPreallocAccess` will be 0 (zero), even though the minimum value is 128. After you mount an AdvFS fileset, the current value will be changed to 128.

4.4.3 Displaying Minimum and Maximum Attribute Values

Each subsystem attribute has a minimum and maximum value. If you modify an attribute, the value must be between these values. However, the minimum and maximum values should be used with caution. Instead, use the tuning recommendations described in this manual to determine an appropriate attribute value for your configuration.

Use one of the following methods to display the minimum and maximum allowable values for an attribute:

- `sysconfig -Q` *subsystem* [*attribute*]

  Displays the minimum and maximum values for all the attributes of the specified subsystem or for only the specified attribute. The command output includes information in the `type` field about the value expected (for example, integer, unsigned integer, long integer, or string). The output also specifies, in the `op` field, the operations that you can perform on the attribute:

  – `C`—The attribute can be modified only when the subsystem is initially loaded; that is, the attribute supports only boot time, permanent modifications.

  – `R`—The attribute can be tuned at run time; that is, you can modify the value that the system is currently using.

  – `Q`—The attribute's current value can be displayed (queried).

For example:

```
# sysconfig -Q vfs bufcache
vfs:
bufcache -      type=INT op=CQ min_val=0 max_val=50
```

The output of the previous command shows that the minimum value is 0 and the maximum value is 50. The output also shows that you cannot modify the current (run-time) value.

See `sysconfig`(8) for more information.

- Kernel Tuner

  This GUI displays the minimum and maximum values for the subsystem attributes. To access the Kernel Tuner, click on the Application Manager icon in the CDE menu bar, select System_Admin, and then select MonitoringTuning. You can then click on Kernel Tuner. A pop-up menu containing a list of subsystems appears. Select a subsystem to display a list of the subsystem's attributes and their minimum and maximum values.

4.4.4 Modifying Run-Time Attribute Values

Modifying an attribute's current (run-time) value allows the change to occur immediately, without rebooting the system. Not all attributes support run-time modifications.

Modifications to run-time values are lost when you reboot the system and the attribute values return to their permanent values. To make a permanent change to an attribute value, see Section 4.4.5.

To determine if an attribute can be tuned at run time, use one of the following methods:

- `sysconfig -Q` *subsystem* [*attribute*]

  The command output indicates that an attribute can be tuned at run time if the `op` field includes an `R`. The following command shows that the `max-vnodes` attribute can be tuned at run time:

```
# sysconfig -Q vfs max-vnodes
vfs:
max-vnodes -    type=INT op=CRQ min_val=0 max_val=2147483647
```

  See `sysconfig`(8) for more information.

- Kernel Tuner

  To access the Kernel Tuner, click on the Application Manager icon in the CDE menu bar, select System_Admin, and then select MonitoringTuning. You can then click on Kernel Tuner. A pop-up menu containing a list of subsystems appears. Select a subsystem to display the subsystem's attributes. The attribute can be tuned at run time if the attribute's `Current Value` field allows you to enter a value.

To modify an attribute's run-time value, use one of the following methods:

- `sysconfig -r` *subsystem attribute*=*value*

  This command modifies the run-time value of the specified subsystem attribute. The *value* argument specifies the new current attribute value that you want the system to use. The following example changes the current value of the `socket` subsystem attribute `somaxconn` to 65536.

  ```
  # sysconfig -r socket somaxconn=65535
  somaxconn: reconfigured
  ```

- Kernel Tuner

  This GUI allows you to enter a new current value for an attribute. To access the Kernel Tuner, click on the Application Manager icon in the CDE menu bar, select System_Admin, and then select MonitoringTuning. You can then click on Kernel Tuner. A pop-up menu containing a list of subsystems appears. Select a subsystem and enter the new attribute value that you want the kernel to use in the attribute's `Current Value` field.

Note

Do not specify erroneous values for subsystem attributes, because system behavior may be unpredictable. If you want to modify an attribute, use only the recommended values described in this manual.

To return to the original attribute value, either modify the run-time value or reboot the system.

4.4.5 Permanently Modifying Attribute Values

To permanently change the value of an attribute, you must include the new value in the `/etc/sysconfigtab` file, using the required format. Do not edit the file manually.

Note

Before you permanently modify a subsystem attribute, it is recommended that you maintain a record of the original value, in case you need to return to this value.

Use one of the following methods to permanently modify the value of an attribute:

- `sysconfigdb -a -f` *stanza_file subsystem*

  Use this command to specify the stanza-formatted file that contains the subsystem, the attribute, and the new permanent attribute value. The *subsystem* argument specifies the subsystem whose attribute you want to modify. See `sysconfigdb`(8) for more information.

  The following is an example of a stanza-formatted file that changes the permanent values of the `socket` subsystem attributes `somaxconn` and `somaxconn` to 65535:

  ```
  socket:
   somaxconn = 65535
   sominconn = 65535
  ```

  See `stanza`(4) for information about stanza-formatted files.

  To use the new permanent value, reboot the system or, if the attribute can be tuned at run time, use the `sysconfig -r` command to change the current value (see Section 4.4.4).

- Kernel Tuner

  This GUI allows you to change the permanent value of an attribute. To access the Kernel Tuner, click on the Application Manager icon in the CDE menu bar, select System_Admin, and then select MonitoringTuning. You can then click on Kernel Tuner. A pop-up menu containing a list of subsystems appears. Select the subsystem for the attribute that you want to modify. Enter the new permanent value in the attribute's `Boot Time Value` field.

  To use the new attribute value, reboot the system or, if the attribute can be tuned at run time, enter the new value in the `Current Value` field (see Section 4.4.4).

Note

Do not specify erroneous values for subsystem attributes, because system behavior may be unpredictable. If you want to modify an attribute, use only the recommended values described in this manual.

4.4.6 Displaying and Modifying Kernel Variables by Using the dbx Debugger

Use the `dbx` debugger to examine the values of kernel variables and data structures and to modify the current (run-time) values of kernel variables.

The following example of the `dbx print` command displays the current (run-time) value of the `vm_page_free_count` kernel variable:

```
# /usr/ucb/dbx -k /vmunix /dev/mem
(dbx) print vm_page_free_count
248
(dbx)
```

The following example of the `dbx print` command displays the current (run-time) values of the kernel variables in the `vm_perfsum` data structure:

```
# /usr/ucb/dbx -k /vmunix /dev/mem
(dbx) print vm_perfsum
struct {
    vpf_pagefaults = 1689166
    vpf_kpagefaults = 13690
    vpf_cowfaults = 478504
    vpf_cowsteals = 638970
    vpf_zfod = 255372
    vpf_kzfod = 13654
    vpf_pgiowrites = 3902
    .
    .
    .

    vpf_vmwiredpages = 440
    vpf_ubcwiredpages = 0
    vpf_mallocpages = 897
    vpf_totalptepages = 226
    vpf_contigpages = 3
    vpf_rmwiredpages = 0
    vpf_ubcpages = 2995
    vpf_freepages = 265
    vpf_vmcleanpages = 237
    vpf_swapspace = 7806
}
(dbx)
```

Use the `dbx patch` command to modify the current (run-time) values of kernel variables. The values you assign by using the `dbx patch` command are lost when you rebuild the kernel.

Notes

If possible, use the `sysconfig` command or the Kernel Tuner to modify subsystem attributes instead of using `dbx` to modify kernel variables. Do not specify erroneous values for kernel variables, because system behavior may be unpredictable. If you want to modify a variable, use only the recommended values described in this manual.

The following example of the `dbx patch` command changes the current value of the `cluster_consec_init` variable to 8:

```
# /usr/ucb/dbx -k /vmunix /dev/mem
(dbx) patch cluster_consec_init = 8
32767
(dbx)
```

To ensure that the system is utilizing a new kernel variable value, reboot the system. See the *Programmer's Guide* for detailed information about the `dbx` debugger.

You can also use the `dbx assign` command to modify run-time kernel variable values. However, the modifications are lost when you reboot the system.

4.5 Solving Common Performance Problems

The following sections provide examples of some common performance problems and solutions:

- Slow application performance (Section 4.5.1)
- Excessive paging (Section 4.5.2)
- Insufficient swap space (Section 4.5.3)
- Insufficient CPU cycles (Section 4.5.4)
- Swapped out processes (Section 4.5.5)
- Disk bottleneck (Section 4.5.6)
- Poor disk I/O performance (Section 4.5.7)
- Poor AdvFS performance (Section 4.5.8)
- Poor UFS performance (Section 4.5.9)
- Poor NFS performance (Section 4.5.10)
- Poor network performance (Section 4.5.11)

Each section describes how to detect the problem, the possible causes of the problem, and how to eliminate or diminish the problem.

4.5.1 Application Completes Slowly

Use the following table to detect a slow application completion time and to diagnose the performance problem:

How to detect	Check application log files. Use the `ps` command to display information about application processing times and whether an application is swapped out. See Section 6.3.1. Use process accounting commands to obtain information about process completion times. See `accton`(8).
Cause	Application is poorly written.
Solution	Rewrite the application so that it runs more efficiently. See Chapter 7. Use profiling and debugging commands to analyze applications and identify inefficient areas of code. See Section 11.1.
Cause	Application is not optimized.
Solution	Optimize the application. See Chapter 7.
Cause	Application is being swapped out.
Solution	Delay swapping processes. See Section 6.5.1. Increase the memory available to processes. See Section 6.4. Reduce an application's use of memory. See Section 11.2.6.
Cause	Application requires more memory resources.
Solution	Increase the memory available to processes. See Section 6.4. Reduce an application's use of memory. See Section 11.2.6.
Cause	Insufficient swap space.
Solution	Increase the swap space and distribute it across multiple disks. See Section 4.5.3.
Cause	Application requires more CPU resources.
Solution	Provide more CPU resources to processes. See Section 4.5.4.
Cause	Disk I/O bottleneck.
Solution	Distribute disk I/O efficiently. See Section 4.5.6.

4.5.2 Excessive Memory Paging

A high rate of paging or a low free page count may indicate that you have inadequate memory for the workload. Avoid paging if you have a large memory system. Use the following table to detect insufficient memory and to diagnose the performance problem:

How to detect	Use the `vmstat` command to display information about paging and memory consumption. See Section 6.3.2 for more information. Use the `dbx vm_perfsum` data structure to display the number of page faults and other memory information. See Section 6.3.4. Check the UBC paging information by using the `dbx vm_perfsum` data structure. See Section 6.3.5.
Cause	Insufficient memory resources available to processes.
Solution	Reduce an application's use of memory. See Section 11.2.6. Increase the memory resources that are available to processes. See Section 6.4.

4.5.3 Insufficient Swap Space

If you consume all the available swap space, the system will display messages on the console indicating the problem. Use the following table to detect if you have insufficient swap space and to diagnose the performance problem:

How to detect	Invoke the `swapon -s` while you are running a normal workload. See Section 6.3.3.
Cause	Insufficient swap space for your configuration.
Solution	Configure enough swap space for your configuration and workload. See Section 2.3.2.3.
Cause	Swap space not distributed.
Solution	Distribute the swap load across multiple swap devices to improve performance. See Section 6.2.
Cause	Applications are utilizing excessive memory resources.
Solution	Increase the memory available to processes. See Section 6.4. Reduce an application's use of memory. See Section 11.2.6.

4.5.4 Insufficient CPU Cycles

Although a low CPU idle time can indicate that the CPU is being fully utilized, performance can suffer if the system provides an insufficient number of CPU cycles to processes. Use the following table to detect insufficient CPU cycles and to diagnose the performance problem:

How to detect	Use the `vmstat` command to display information about CPU system, user, and idle times. See Section 6.3.2 for more information. Use the `kdbx cpustat` extension to check CPU usage. See Section 7.1.4).
Cause	Excessive CPU demand from applications.
Solution	Optimize applications. See Section 11.2.4. Use hardware RAID to relieve the CPU of disk I/O overhead. See Section 8.4.

4.5.5 Processes Swapped Out

Swapped out (suspended) processes will decrease system response time and application completion time. Avoid swapping if you have a large memory system or large applications. Use the following table to detect if processes are being swapped out and to diagnose the performance problem:

How to detect	Use the `ps` command to determine if your system is swapping processes. See Section 6.3.1.
Cause	Insufficient memory resources.
Solution	Increase the memory available to processes. See Section 6.4. Reduce an application's use of memory. See Section 11.2.6.
Cause	Swapping occurs too early during page reclamation.
Solution	Decrease the rate of swapping. See Section 6.5.1.

4.5.6 Disk Bottleneck

Excessive I/O to only one or a few disks may cause a bottleneck at the over-utilized disks. Use the following table to detect an uneven distribution of disk I/O and to diagnose the performance problem:

How to detect	Use the `iostat` command to display which disks are being used the most. See Section 8.2.1. Use the `swapon -s` command to display the utilization of swap disks. See Section 6.3.3. Use the `volstat` command to display information about the LSM I/O workload. See Section 8.3.4.2 for more information. Use the `advfsstat` to display AdvFS disk usage information. See Section 9.3.3.1.
Cause	Disk I/O not evenly distributed.
Solution	Use disk striping. See Section 2.5.2 Distribute disk, swap, and file system I/O across different disks and, optimally, multiple buses. See Section 8.1.

4.5.7 Poor Disk I/O Performance

Because disk I/O operations are much slower than memory operations, the disk I/O subsystem is often the source of performance problems. Use the following table to detect poor disk I/O performance and to diagnose the performance problem:

How to detect	Use the `iostat` command to determine if a you have a bottleneck at a disk. See Section 8.2.1 for more information. Check for disk fragmentation by using the AdvFS `defragment` utility with the `-v` and `-n` options. Check the hit rate of the namei cache with the `dbx nchstats` data structure. See Section 9.1.2. Monitor the memory allocated to the UBC by using the `dbx vm_perfsum`, `ufs_getapage_stats`, and `vm_tune` data structures. See Section 6.3.5. Check UFS clustering with the `dbx ufs_clusterstats` data structure. See Section 6.3.5. Check the hit rate of the metadata buffer cache by using the `dbx bio_stats` data structure. See Section 9.4.2.3. Use the `advfsstat` command to monitor the performance of AdvFS domains and filesets. See Section 9.3.3.1.
Cause	Disk I/O is not efficiently distributed.
Solution	Use disk striping. See Section 2.5.2 Distribute disk, swap, and file system I/O across different disks and, optimally, multiple buses. See Section 8.1.
Cause	File systems are fragmented.
Solution	Defragment file systems. See Section 9.3.4.4 and Section 9.4.3.3.
Cause	Maximum open file limit is too small.
Solution	Increase the maximum number of open files. See Section 5.5.1.
Cause	The namei cache is too small.
Solution	Increase the size of the namei cache. See Section 9.2.1.

4.5.8 Poor AdvFS Performance

Use the following table to detect poor AdvFS performance and to diagnose the performance problem:

How to detect	Use the `advfsstat` command to monitor the performance of AdvFS domains and filesets. See Section 9.3.3.1 Check for disk fragmentation by using the AdvFS `defragment` command with the `-v` and `-n` options. See `defragment`(8) for more information.
Cause	Single-volume domains are being used.
Solution	Use multiple-volume file domains. See Section 9.3.2.1.
Cause	File system is fragmented.
Solution	Defragment the file system. See Section 9.3.4.4.
Cause	There are too few AdvFS buffer cache hits.
Solution	Allocate sufficient memory to the AdvFS buffer cache. See Section 9.3.4.1. Increase the number of AdvFS buffer hash chains (Section 9.3.4.2. Increase the dirty data caching threshold. See Section 9.3.4.5. Modify the AdvFS device queue limit. See Section 9.3.4.6.
Cause	The `advfsd` daemon is running unnecessarily.
Solution	Stop the daemon. See Section 7.2.5.

4.5.9 Poor UFS Performance

Use the following table to detect poor UFS performance and to diagnose the performance problem:

How to detect	Use the `dumpfs` command to display UFS information. See Section 9.4.2.1. Check the hit rate of the namei cache with the `dbx nchstats` data structure. See Section 9.1.2. Monitor the memory allocated to the UBC by using the `dbx vm_perfsum`, `ufs_getapage_stats`, and `vm_tune` data structures. Check how effectively the system is clustering and check fragmentation by using the `dbx print` command to examine the `ufs_clusterstats`, `ufs_clusterstats_read`, and `ufs_clusterstats_write` data structures. See Section 9.4.2.2. Check the hit rate of the metadata buffer cache by using the `dbx bio_stats` data structure. See Section 9.4.2.3.
Cause	The UBC is too small.
Solution	Increase the amount of memory allocated to the UBC. See Section 9.2.3.
Cause	The metadata buffer cache is too small.
Solution	Increase the size of metadata buffer cache. See Section 9.4.3.1.
Cause	The file system fragment size is incorrect.
Solution	Make the file system fragment size equal to the block size. See Section 9.4.1.1.
Cause	File system is fragmented.
Solution	Defragment the file system. Section 9.4.3.3.

4.5.10 Poor NFS Performance

Use the following table to detect poor NFS performance and to diagnose the performance problem:

How to detect	Use the `nfsstat` command to display the number of NFS requests and other information. See Section 9.5.1.1. Use the `dbx print nchstats` command to determine the namei cache hit rate. See Section 9.1.2. Use the `dbx print bio_stats` command to determine the metadata buffer cache hit rate. See Section 9.4.2.3. Use the `dbx print vm_perfsum` command to check the UBC hit rate. See Section 6.3.5. Use the `dbx print nfs_sv_active_hist` command to display a histogram of the active NFS server threads. See Section 4.4.6. Use the `ps axlmp` command to display the number of idle threads. See Section 9.5.2.2 and Section 9.5.2.3.
Cause	NFS server threads busy.
Solution	Reconfigure the server to run more threads. See Section 9.5.2.2.
Cause	Memory resources are not focused on file system caching.
Solution	Increase the number of vnodes on the free list. See Section 9.2.11. If you are using AdvFS, increase the memory allocated for AdvFS buffer caching. See Section 9.3.4.1. If you are using AdvFS, increase the memory reserved for AdvFS access structures. See Section 9.3.4.3.
Cause	UFS metadata buffer cache hit rate is low.
Solution	Increase the size of the metadata buffer cache. See Section 9.4.3.1. Increase the size of the namei cache. See Section 9.2.1.
Cause	CPU idle time is low.
Solution	Use UFS, instead of AdvFS. See Section 9.4.

4.5.11 Poor Network Performance

Use the following table to detect poor network performance and to diagnose the performance problem:

How to detect	Use the `netstat` command to display information about network collisions and dropped network connections. See Section 10.1.1. Check the socket listen queue statistics to check the number of pending requests and the number of times the system dropped a received SYN packet. See Section 10.1.2.
Cause	The TCP hash table is too small.
Solution	Increase the size of the hash table that the kernel uses to look up TCP control blocks. See Section 10.2.1.
Cause	The limit for the socket listen queue is too low.
Solution	Increase the limit for partial TCP connections on the socket listen queue. See Section 10.2.3.
Cause	There are too few outgoing network ports.
Solution	Increase the maximum number of concurrent nonreserved, dynamically allocated ports. See Section 10.2.4.
Cause	Network connections are becoming inactive too quickly.
Solution	Enable TCP keepalive functionality. See Section 10.2.6.

5 Tuning System Resource Limits

The Tru64 UNIX operating system sets resource limits at boot time. These limits control the size of system tables, virtual address space, and other system resources.

The default system resource limits are appropriate for most configurations. However, if your configuration has a large amount of memory or is running a program that requires extensive resources, you may need to increase the system limits by modifying subsystem attributes.

This chapter describes how to increase the following system-wide limits:

- Process limits (Section 5.1)
- Program size limits (Section 5.2)
- Virtual memory limits (Section 5.3)
- Interprocess communication (IPC) limits (Section 5.4)
- Open file limits (Section 5.5)

Instead of modifying system-wide limits, you can use the `setrlimit` function to control the consumption of system resources by a specific process and its child processes. See `setrlimit`(2) for information.

5.1 Tuning Process Limits

If your applications are memory-intensive or you have a large-memory configuration, you may want to increase the process limits. Increasing process limits will increase the amount of wired memory in the system.

The following `proc` (process) subsystem attributes control process limits:

- `maxusers`

  System algorithms use the `proc` subsystem attribute `maxusers` to size various system data structures, and to determine the amount of space allocated to system tables, such as the system process table, which is used to determine how many active processes can be running at one time.

  The default value assigned to the `maxusers` attribute depends on the amount of memory in your system. Most systems have a default value of 32.

If you have a large-memory system or your system experiences a lack of resources, increase the value of the `maxusers` attribute. A lack of resources may be indicated by a `No more processes` or `Out of processes` message. However, increasing the value of the `maxusers` attribute will increase the amount of wired memory consumed by the kernel.

To determine an appropriate value for the `maxusers` attribute, double the value until you notice a performance improvement. You can also use the following guidelines:

Configuration	Value of maxusers Attribute
1 GB of memory	512
2 GB of memory	1024
Internet, Web, proxy, firewall, or gateway server	2048

It is recommended that you not increase the value of the `maxusers` attribute to more than 2048.

If you increase the value of `maxusers,` you may want to increase the value of the `max-vnodes` attribute proportionally (see Section 5.5.1).

- `task-max`

  The `proc` subsystem attribute `task-max` specifies the maximum number of tasks that can run simultaneously. The default value is 20 + 8 * `maxusers`.

- `thread-max`

  The `proc` subsystem attribute `thread-max` specifies the maximum number of threads that can run simultaneously. The default value is 2 * `task-max`.

- `max-proc-per-user`

  The `proc` subsystem attribute `max-proc-per-user` specifies the maximum number of processes that can be allocated at any one time to each user, except superuser. The default value of the `max-proc-per-user` attribute is 64.

  If your system experiences a lack of processes, you can increase the value of the `max-proc-per-user` attribute. The value must be more than the maximum number of processes that will be started by your system. If you have a Web server, these processes include CGI processes.

  If you have an Internet, Web, proxy, firewall, or gateway server, increase the value of the `max-proc-per-user` attribute to 512.

- `max-threads-per-user`

The `proc` subsystem attribute `max-threads-per-user` specifies the maximum number of threads that can be allocated at any one time to each user, except superuser. The default value is 256.

If your system experiences a lack of threads, you can increase the `max-threads-per-user` attribute to a value that is greater than the maximum number of threads started by your system. For example, you could increase the value of the `max-threads-per-user` attribute to 512.

On a very busy server with sufficient memory or an Internet, Web, proxy, firewall, or gateway server, increase the value of the `max-threads-per-user` attribute to 4096.

Setting the value of the `max-threads-per-user` attribute to 0 (zero) will remove the limit on threads.

See Section 4.4 for information about modifying kernel subsystem attributes.

5.2 Tuning Program Size Limits

If you are running a very large application, you may need to increase the values of the `proc` subsystem attributes that control program size limits. Some extremely large programs may not run unless you modify the default values of the following attributes:

- `per-proc-stack-size` and `max-per-proc-stack-size`

  These `proc` subsystem attributes control the default and the maximum sizes of a user process stack. The default value of the `per-proc-stack-size` attribute is 2097152 bytes. The default value of the `max-per-proc-stack-size` attribute is 33554432 bytes.

  You may need to increase these values if you receive `Cannot grow stack` messages.

- `per-proc-data-size` and `max-per-proc-data-size`

  These `proc` subsystem attributes control the default and the maximum sizes of a user process data segment. The default value of the `per-proc-data-size` attribute is 134217728 bytes. The default value of the `max-per-proc-data-size` is 1 GB.

  You may need to increase the value of the `max-per-proc-data-size` attribute if you receive an `Out of process memory` message.

  If you have an Internet, Web, proxy, firewall, or gateway server, increase the value of the `max-per-proc-data-size` attribute to 10 GB (10737418240 bytes).

See Section 4.4 for information about modifying kernel subsystem attributes.

5.3 Tuning Virtual Memory Limits

If your configuration has a large amount of memory or you are running a large program, you may need to increase the limits on virtual address space and other resources.

Table 5–1 describes the recommendations for increasing the virtual memory limits for processes and lists the performance benefits as well as tradeoffs.

Table 5–1: Virtual Memory Limits Tuning Guidelines

Recommendation	Performance Benefit	Tradeoff
Increase the available address space (Section 5.3.1)	Improves performance for memory-intensive processes	Consumes a small amount of memory
Increase the maximum number of memory-mapped files that are available to a process (Section 5.3.2)	Increases file mapping and improves performance for memory-intensive processes, such as Internet servers	Consumes memory
Increase the maximum number of virtual pages within a process's address space that can have individual protection attributes (Section 5.3.3)	Improves performance for memory-intensive processes and for Internet servers that maintain large tables or resident images	None

The following sections describe the recommendations for increasing the virtual memory limits for processes.

5.3.1 Increasing Address Space

If your applications are memory-intensive, you may need to increase the address space limits. Increasing the address space limits will cause only a small increase in the demand for memory. You may not want to increase the address space if your applications use many forked processes.

The following attributes determine the available address space for processes:

- `vm-maxvas`

  This `vm` subsystem attribute specifies the maximum amount of valid virtual address space for a process (that is, the sum of all the valid pages). The default is 128000 pages (1 GB).

  If you have an Internet, Web, proxy, firewall, or gateway server, increase the value of the `vm-maxvas` attribute to 10737418240 (10 GB).

- `per-proc-address-space` and `max-per-proc-address-space`

  These `proc` subsystem attributes control the maximum amount of user process address space, which is the maximum number of valid virtual regions. The default value for both attributes is 1 GB.

  If you have an Internet, Web, proxy, firewall, or gateway server, increase the value of the `max-per-proc-address-space` attribute to 10 GB (10737418240 bytes).

See Section 4.4 for information about modifying kernel attributes.

5.3.2 Increasing the Number of Memory-Mapped Files

You may want to increase the maximum number of memory-mapped files in a user address. Each map entry describes one unique disjoint portion of a virtual address space. The `vm` subsystem attribute `vm-mapentries` specifies the maximum number of memory-mapped files in a user address. The default value is 200.

Increasing the value of the `vm-mapentries` attribute will increase the demand for memory.

You may want to increase the value of the `vm-mapentries` attribute for large-memory systems and Internet servers to increase the limit on file mapping. Because Web servers map files into memory, increase the value of the `vm-mapentries` attribute to 20000 on busy systems running multithreaded Web server software and on Internet, proxy, firewall, and gateway servers.

See Section 4.4 for information about modifying kernel subsystem attributes.

5.3.3 Increasing the Number of Pages with Individual Protections

The `vm` subsystem attribute `vm-vpagemax` specifies the maximum number of virtual pages that can be given individual protection attributes within the address space of a process. These protection attributes differ from the protection attributes associated with the other pages in the address space.

Changing the protection attributes of a single page within a virtual memory region causes all pages within that region to be treated as though they had individual protection attributes. For example, each thread of a multithreaded task has a user stack in the stack region for the process in which it runs. Because multithreaded tasks have guard pages (that is, pages that do not have read/write access) inserted between the user stacks for the threads, all pages in the stack region for the process are treated as though they have individual protection attributes.

If you have a large-memory system or if a stack region for a multithreaded task exceeds 16 KB pages, you may need to increase the value of the `vm-vpagemax` attribute. The default value is 16384.

You can determine an appropriate value for the `vm-vpagemax` attribute by dividing the value of the `vm` subsystem attribute `vm-maxvas` (the address space size in bytes) by 8192. For example, if the value of the `vm-maxvas` attribute is 1 GB (the default), set the value of `vm-vpagemax` to 131072 pages (1073741824/8192=131072). A value of 131072 pages may improve the efficiency of Internet servers that maintain large tables or resident images.

See Section 4.4 for information about modifying kernel subsystem attributes.

5.4 Tuning Interprocess Communication Limits

Interprocess communication (IPC) is the exchange of information between two or more processes. Some examples of IPC include messages, shared memory, semaphores, pipes, signals, process tracing, and processes communicating with other processes over a network. IPC is a functional interrelationship of several operating system subsystems. Elements are found in scheduling and networking.

The Tru64 UNIX operating system provides the following facilities for interprocess communication:

- Pipes—See the *Guide to Realtime Programming* for information about pipes.
- Signals—See the *Guide to Realtime Programming* for information about signals.
- Sockets—See the *Network Programmer's Guide* for information about sockets.
- Streams—See the *Programmer's Guide: STREAMS* for information about streams.
- X/Open Transport Interface (XTI)—See the *Network Programmer's Guide* for information about XTI.

You can track the use of IPC facilities with the `ipcs -a` command (see `ipcs`(1)). By looking at the current number of bytes and message headers in the queues, you can then determine whether you need to tune the System V message queue to diminish waiting.

Table 5–2 describes the recommendations for increasing IPC limits and lists the performance benefits as well as tradeoffs.

Table 5–2: IPC Limits Tuning Guidelines

Recommendation	Performance Benefit	Tradeoff
Increase the maximum size of a System V message (Section 5.4.1)	Enables memory-intensive or VLM systems to run efficiently	Consumes a small amount of memory
Increase the maximum number of bytes on a System V message queue (Section 5.4.2)	Prevents a process from sleeping if the queue is too full to accommodate a message	None
Increase the maximum number of messages on a System V queue (Section 5.4.3)	Enables memory-intensive or VLM systems to run efficiently	Consumes a small amount of memory
Increase the maximum size of a System V shared memory region (Section 5.4.4)	Enables memory-intensive or VLM systems to run efficiently	Consumes memory
Increase the minimum size of a System V shared memory segment (Section 5.4.5)	Enables memory-intensive or VLM systems to run efficiently	Consumes memory

The following sections describe how to tune some System V attributes. See `sys_attrs_ipc`(5) for information about additional IPC subsystem attributes.

5.4.1 Increasing the Maximum Size of a System V Message

If your applications are memory-intensive or you have a large-memory system, you may want to increase the maximum size of a single System V message.

The `ipc` subsystem attribute `msg-max` specifies the maximum size of a single System V message. The default value is 8192 bytes (1 page). Increasing the value of this attribute will slightly increase the demand for memory.

See Section 4.4 for information about modifying kernel subsystem attributes.

5.4.2 Increasing the Maximum Size of a System V Message Queue

A process will be unable to send a message to a queue if it will make the number of bytes in the queue greater than the limit specified by the `ipc` subsystem attribute `msg-mnb`. When the limit is reached, the process sleeps and waits for this condition to be resolved.

To prevent a process from sleeping if the queue is too full to accommodate a message, increase the value of the `msg-mnb` attribute. The default value is 16384 bytes.

See Section 4.4 for information about modifying kernel subsystem attributes.

5.4.3 Increasing the Maximum Number of Messages on a System V Queue

If your applications are memory-intensive or you have a large-memory system, you may want to increase the the maximum number of messages that can be queued to a single System V message queue at one time.

The `ipc` subsystem attribute `msg-tql` specifies the maximum number of messages that can be on a System V message queue. The default value is 40. Increasing the value of this attribute will increase the demand for memory.

See Section 4.4 for information about modifying kernel subsystem attributes.

5.4.4 Increasing the Maximum Size of a System V Shared Memory Region

If your applications are memory-intensive or you have a large-memory system, you may want to increase the maximum size of a single System V shared memory region.

The `ipc` subsystem attribute `shm-max` specifies the maximum size of a single System V shared memory region. The default value is 4194304 bytes (512 pages). Increasing the value of this attribute will increase the demand for memory.

In addition, the third-level page table is shared among processes when the shared memory region size is equal to or greater than the shared memory segment threshold, as specified by the value of the `ipc` subsystem attribute `ssm-threshold` (the default value is 8 MB). When this occurs, shared memory becomes segmented shared memory. You can set the value of the `ssm-threshold` attribute to 0 (zero) to disable the use of segmented shared memory. See Section 5.4.5 for more information.

In addition, you may want to increase the value of the `ipc` subsystem attribute `shm-seg`. This attribute specifies the maximum number of System V shared memory regions that can be attached to a single process at any point in time. The default value is 32. Increasing the value of this attribute will increase the demand for memory.

As a design consideration, consider whether you will get better performance by using threads instead of shared memory.

See Section 4.4 for information about modifying kernel subsystem attributes.

5.4.5 Increasing the Minimum Size of a System V Shared Memory Segment

Page table sharing occurs when the size of a System V shared memory segment reaches the value specified by the `ipc` subsystem attribute `ssm-threshold`. The default value is 8 MB (8388608 bytes). If your applications are memory-intensive, you may want to increase the value of this attribute.

Increasing the value of the `ssm-threshold` attribute will increase the demand for memory.

Setting the `ssm-threshold` attribute to 0 (zero) will disable the use of segmented shared memory.

See Section 4.4 for information about modifying kernel subsystem attributes.

5.5 Tuning the Open File Limits

The following sections describe how to increase the maximum number of open files (Section 5.5.1) and the maximum number of open file descriptors (Section 5.5.2).

5.5.1 Increasing the Maximum Number of Open Files

The kernel data structure for an open file is called a vnode. These are used by all file systems. The allocation and deallocation of vnodes is handled dynamically by the operating system. The number of vnodes determines the number of open files.

If your applications require many open files and you receive a message indicating you are out of vnodes, increase the value of the `vfs` subsystem attribute `max-vnodes` or the `proc` subsystem attribute `maxusers` to increase the maximum number of vnodes and open files. However, increasing the maximum number of vnodes will increase the memory demand.

The default value of the `max-vnodes` attribute is 5 percent of memory. See Section 5.1 for information about the `maxusers` attribute.

See Section 4.4 for information about modifying kernel subsystem attributes.

5.5.2 Increasing the Maximum Number of Open File Descriptors

You can increase the maximum number of open file descriptors for all processes or for a specific application.

The `proc` subsystem attributes `open-max-soft` and `open-max-hard` control the maximum system-wide number of open file descriptors (open files) for each process. These attributes prevent runaway allocations, such as allocations within a loop that cannot be exited because of an error condition, from consuming all of the available file descriptors. If a process reaches the `open-max-soft` limit, a warning message is issued. If a process reaches the `open-max-hard` limit, the process is stopped.

The default value of the `open-max-soft` and `open-max-hard` attributes is 4096, which is the maximum system-wide value that you can set in the `/etc/sysconfigtab` file.

If you have an application that requires many open files, you can increase the open file descriptor limit only for that application, instead of increasing the system-wide limit. To enable extended (64 KB) file descriptors for a specific application, follow these steps:

1. Set the `setsysinfo` system call's `SSI_FD_NEWMAX` operation parameter to 1, which sets the `utask` bit, enables up to 65,536 (64 KB) open file descriptors, and raises the process's hard file limit to 64 KB. This setting is inherited by any child process. See `setsysinfo`(2) for more information.
2. Set the process's file descriptor soft limit to a value that is more than 4096 (the default value) by using the `setrlimit` function as shown in the following code fragment:

```
#include <sys/resource.h>
struct rlimit *rlp;

rlp->rlim_cur = 6000;
rlp->rlim_max = 6000;
setrlimit(RLIMIT_NOFILE, rlp);
```

This setting is inherited by any child process. See `setrlimit`(2) for more information.

3. This step is required only for applications that use the `select` function's `fd_set` parameter, which points to an I/O descriptor set, and a `FD_CLR`, `FD_ISSET`, `FD_SET`, or `FD_ZERO` macro and can modify an I/O descriptor set. If you meet these qualifications, you can use one of two procedures, one that enables a static definition of the maximum number of file descriptors or one that enables a dynamic definition:

   - Static definition:

     Override the default value of 4096 for `FD_SETSIZE` in the <sys/select.h> header file by specifying the maximum value of 65536. You must specify this value before you include the <sys/time.h> header file (which also includes the <sys/select.h> header file) in the code, as follows:

     ```
     #define FD_SETSIZE 65536
     #include <sys/time.h>
     ```

     This setting is not inherited by child processes; therefore, `FD_SETSIZE` must be set explicitly in the code for each child process that requires 64 KB file descriptors.

   - Dynamic definition:

     Instead of using statically-defined `fd_set` structures, you can use `fd_set` pointers in conjunction with a `malloc` function, which provides forward compatibility with any future changes to the maximum file descriptor limit. For example:

     ```
     fd_set *fdp;

     fdp = (fd_set *) malloc(
     (fds_howmany(max_fds,FD_NFDBITS))*sizeof(fd_mask));
     ```

     The value for `max_fds` is the number of file descriptors to be manipulated. It is recommended that you use the file descriptor soft limit for this value. All other keywords are defined in the <sys/select.h> header file. The following code segment shows this choice:

```
#include <sys/time.h>
#include <sys/resource.h>

my_program()
{
fd_set *fdp;
struct rlimit rlim;
int max_fds;

getrlimit(RLIMIT_NOFILE, &rlim;);
max_fds = rlim.rlim_cur;

fdp = (fd_set *) malloc(
(fds_howmany(max_fds,FD_NFDBITS))*sizeof(fd_mask));

FD_SET(2, fdp);

for (;;) {
switch(select(max_fds, (fd_set *)0, fdp, (fd_set
*)0,
struct timeval *)0)) {
...
}
```

In addition, the `vfs` subsystem attribute `max-vnodes` must be set high enough for the needs of any application that requires a high number of descriptors. The `max-vnodes` attribute specifies the number of vnodes (open files), and is set to 5 percent of system memory by default. The recommended setting is 1 vnode for each file descriptor. See Section 5.5.1 for more information.

See Section 4.4 for information about modifying kernel subsystem attributes.

To disable support for up to 64 KB file descriptors for an application, set the `setsysinfo` system call's `SSI_FD_NEWMAX` operation parameter to 0, which disables the `utask` bit and returns the hard file limit to the default maximum of 4096 open file descriptors. However, if the process is using more than 4096 file descriptors, the `setsysinfo` system call will return an `EINVAL` error. In addition, if a calling process's hard or soft limit exceeds 4096, the limit is set to 4 KB after the call is successful. This setting is inherited by any child process.

6 Managing Memory Performance

You may be able to improve Tru64 UNIX performance by optimizing your memory resources. This chapter describes how to perform the following tasks:

- Understand how the operating system allocates memory to processes and to file system caches and how memory is reclaimed (Section 6.1)
- Configure swap space for high performance (Section 6.2)
- Obtain information about memory performance and consumption (Section 6.3)
- Provide more memory resources to processes (Section 6.4)
- Modify paging and swapping operation (Section 6.5)
- Reserve physical memory for shared memory (Section 6.6)

6.1 Understanding Memory Operation

The operating system allocates physical memory in 8-KB units called pages. The virtual memory subsystem tracks and manages all the physical pages in the system and efficiently distributes the pages among three areas:

- Static wired memory

  Allocated at boot time and used for operating system data and text and for system tables, static wired memory is also used by the metadata buffer cache, which holds recently accessed UNIX File System (UFS) and CD-ROM File System (CDFS) metadata.

  You can reduce the amount of static wired memory only by removing subsystems or by decreasing the size of the metadata buffer cache (see Section 6.1.2.1).

- Dynamically wired memory

  Allocated at boot time and used for dynamically allocated data structures, such as address space wired by user processes, the amount of dynamically wired memory varies according to the demand, but is limited, by default, to 80 percent of physical memory.

  You can reduce the amount of dynamically wired memory by reducing the value of the `vm` subsystem attribute `vm-syswiredpercent`, or by allocating more kernel resources to processes (for example, by increasing the value of the `maxusers` attribute). See Section 5.1 and Section 6.4.3.

- Virtual memory

  Used for processes' most-recently accessed anonymous memory (modifiable virtual address space) and file-backed memory (memory that is used for program text or shared libraries). Virtual memory is also allocated to the Unified Buffer Cache (UBC), which caches most-recently accessed UFS file system data for reads and writes and for page faults from mapped file regions, in addition to AdvFS metadata and file data.

  Processes and the UBC compete for a limited about of physical memory, and the virtual memory subsystem allocates physical pages according to the process and file system demand. To be able to meet the demands of competing claims on memory resources, the virtual memory subsystem periodically reclaims the oldest pages by writing their contents to swap space. Under heavy loads, entire processes may be suspended (swapped out) to free memory.

  You can control virtual memory allocation and operation by tuning various `vm` subsystem attributes, as described in Section 6.1.2 and Section 6.5.

You must understand memory operation to determine which tuning recommendations will improve performance for your workload. The following sections describe how the virtual memory subsystem:

- Tracks physical pages (Section 6.1.1)
- Allocates memory to file system buffer caches (Section 6.1.2)
- Allocates memory to processes (Section 6.1.3)
- Reclaims pages (Section 6.1.4)

6.1.1 Tracking Physical Pages

The virtual memory subsystem tracks all the physical pages of memory in the system. Page lists are used to identify the location and age of all the physical memory pages. The oldest pages are the first to be reclaimed. At any one time, each physical page can be found on one of the following lists:

- **Wired list**—Pages that are wired and cannot be reclaimed
- **Free list**—Pages that are clean and are not being used

  Page reclamation begins when the size of the free list decreases to a tunable limit.

- **Active list**—Pages that are currently being used by processes or the UBC

To determine which active pages should be reclaimed first, the page-stealer daemon identifies the oldest pages on the active list. **Inactive** pages are the oldest pages that are being used by processes. **UBC LRU** (least-recently used) pages are the oldest pages that are being used by the UBC.

Use the `vmstat` command to determine the number of pages that are on the page lists. Remember that pages on the active list (the `act` field in the `vmstat` output) include both inactive and UBC LRU pages.

6.1.2 Allocating Memory to the File System Buffer Caches

The operating system uses three caches to store file system user data and metadata. If the cached data is later reused, a disk I/O operation is avoided, which improves performance. This is because memory access is faster than disk access.

The following sections describe these file system caches:

- Metadata buffer cache (Section 6.1.2.1)
- Unified Buffer Cache (Section 6.1.2.2)
- AdvFS buffer cache (Section 6.1.2.3)

6.1.2.1 Allocating Wired Memory to the Metadata Buffer Cache

At boot time, the kernel allocates wired memory for the metadata buffer cache. The cache acts as a layer between the operating system and disk by storing recently accessed UFS and CDFS **metadata**, which includes file header information, superblocks, inodes, indirect blocks, directory blocks, and cylinder group summaries. Performance is improved if the data is later reused and a disk operation is avoided.

The metadata buffer cache uses `bcopy` routines to move data in and out of memory. Memory in the metadata buffer cache is not subject to page reclamation.

The size of the metadata buffer cache is specified by the value of the `vfs` subsystem attribute `bufcache`. See Section 9.4.3.1 for tuning information.

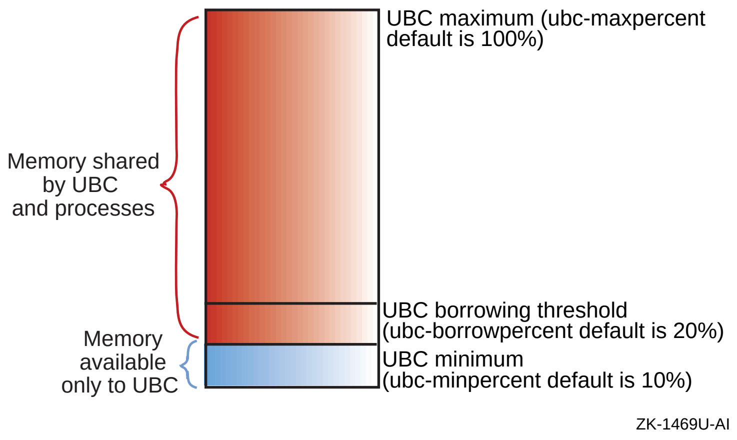
6.1.2.2 Allocating Virtual Memory to the Unified Buffer Cache

After the kernel wires memory at boot time, the remaining memory is available to processes and to the Unified Buffer Cache (UBC), which compete for this memory.

The UBC functions as a layer between the operating system and disk by temporarily holding recently accessed UFS file system data for reads and writes from conventional file activity and holding page faults from mapped file sections. The UBC also holds AdvFS metadata, which it wires so that the transaction log is written to disk before metadata, and AdvFS file data. Performance is improved if the data in the UBC is later reused and a disk I/O operation is avoided.

Figure 6–1 shows how the virtual memory subsystem allocates physical memory to the UBC and for processes.

Figure 6–1: UBC Memory Allocation


UBC maximum (ubc-maxpercent default is 100%)
Memory shared by UBC and processes
UBC borrowing threshold (ubc-borrowpercent default is 20%)
Memory available only to UBC
UBC minimum (ubc-minpercent default is 10%)
ZK-1469U-AI

The amount of memory that the UBC can utilize is determined by three `vm` subsystem attributes:

- `ubc-minpercent` attribute

  Specifies the minimum percentage of virtual memory that only the UBC can utilize. The remaining memory is shared with processes. The default is 10 percent.

- `ubc-maxpercent` attribute

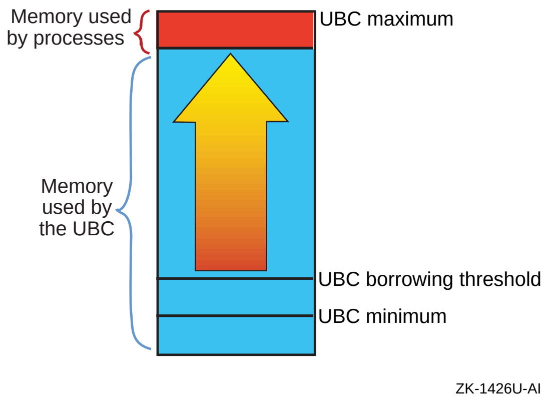
  Specifies the maximum percentage of virtual memory that the UBC can utilize. The default is 100 percent.

- `ubc-borrowpercent` attribute

  Specifies the UBC borrowing threshold. The default is 20 percent. From the value of the `ubc-borrowpercent` attribute to the value of the `ubc-maxpercent` attribute, the UBC is only borrowing virtual memory from processes. When paging starts, pages are first reclaimed from the UBC until the amount of memory allocated to the UBC decreases to the value of the `ubc-borrowpercent` attribute.

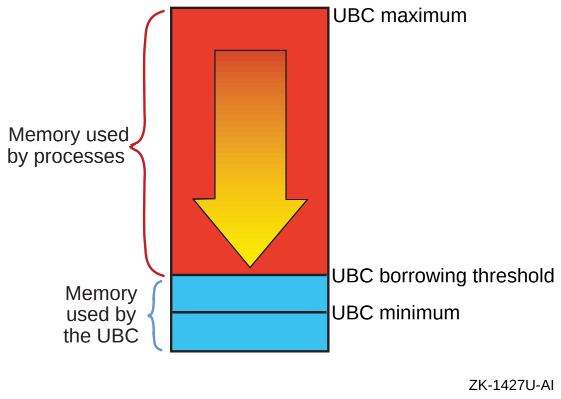
At any one time, the amount of virtual memory allocated to the UBC and to processes depends on the file system and process demands. For example, if file system activity is heavy and process demand is low, most of the pages will be allocated to the UBC, as shown in Figure 6–2.

Figure 6–2: Memory Allocation During High File System Activity and No Paging Activity



In contrast, heavy process activity, such as large increases in the working sets for large executables, will cause the virtual memory subsystem to reclaim UBC borrowed pages, down to the value of the `ubc-borrowpercent` attribute, as shown in Figure 6–3.

Figure 6–3: Memory Allocation During Low File System Activity and High Paging Activity



The UBC uses a hashed list to quickly locate the physical pages that it is holding. A hash table contains file and offset information that is used to speed lookup operations.

The UBC also uses a buffer to facilitate the movement of data between memory and disk. The `vm` subsystem attribute `vm-ubcbuffers` specifies maximum file system device I/O queue depth for writes (that is, the number of UBC I/O requests that can be outstanding). See Section 9.2.6 for tuning information.

6.1.2.3 Allocating Memory to the AdvFS Buffer Cache

The AdvFS buffer cache is part of the UBC and acts as a layer between the operating system and disk by storing recently accessed AdvFS file system data, including file system reads and writes. Performance is improved if the cached data is later reused and a disk operation is avoided.

At boot time, the kernel determines the amount of physical memory that is available for AdvFS buffer cache headers, and allocates a buffer cache header for each possible page. Buffer headers are maintained in a global array and temporarily assigned a buffer handle that refers to an actual buffer page.

The number of AdvFS buffer cache headers depends on the number of 8-KB pages that can be obtained from the amount of memory specified by the `advfs` subsystem attribute `AdvfsCacheMaxPercent`. The default value is 7 percent of physical memory. In addition, the AdvFS buffer cache cannot be more than 50 percent of the UBC.

The AdvFS buffer cache is organized as a fixed-size hash chain table, which uses a file page offset, fileset handle, and domain handle to calculate the hash key that is used to look up a page.

When a page of data is requested, AdvFS searches the hash chain table for a match. If the entry is already in memory, AdvFS returns the buffer handle and a pointer to the page of data to the requester.

If no entry is found, AdvFS obtains a free buffer header and initializes it to represent the requested page. AdvFS performs a read operation to obtain the page from disk and attaches the buffer header to a UBC page. The UBC page is then wired into memory. AdvFS buffer cache pages remain wired until the buffer needs to be recycled, the file is deleted, or the fileset is unmounted.

See Section 6.4.5, and Section 9.3.4.1, and Section 9.3.4.2 for information about tuning the AdvFS buffer cache.

6.1.3 Allocating Virtual Memory to Processes

After the kernel wires memory at boot time, the remaining memory is available to processes and the UBC, which compete for this memory. The virtual memory subsystem allocates memory resources to processes and to the UBC according to the demand, and reclaims the oldest pages if the demand depletes the number of available free pages.

The following sections describe how the virtual memory subsystem allocates memory to processes.

6.1.3.1 Allocating Virtual Address Space to Processes

The `fork` system call creates new processes. When you invoke a process, the `fork` system call:

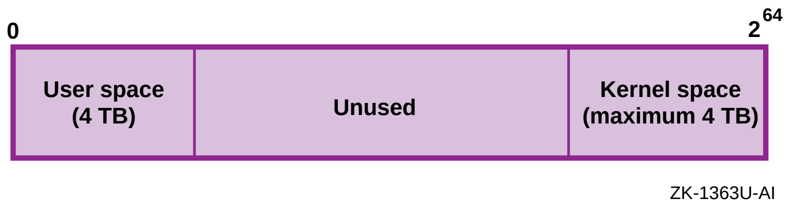
1. Creates a UNIX process body, which includes a set of data structures that the kernel uses to track the process and a set of resource limitations. See `fork`(2) for more information.
2. Allocates a contiguous block of **virtual address space** to the process. Virtual address space is the array of virtual pages that the process can use to map into actual physical memory. Virtual address space is used for **anonymous memory** (memory that holds data elements and structures that are modified during process execution) and for **file-backed memory** (memory used for program text or shared libraries).

   Because physical memory is limited, a process' entire virtual address space cannot be in physical memory at one time. However, a process can execute when only a portion of its virtual address space (its working set) is mapped to physical memory. Pages of anonymous memory and file-backed memory are paged in only when needed. If the memory demand increases and pages must be reclaimed, the pages of anonymous memory are paged out and their contents moved to swap space, while the pages of file-backed memory are simply released.
3. Creates one or more threads of execution. The default is one thread for each process. Multiprocessing systems support multiple process threads.

Although the virtual memory subsystem allocates a large amount of virtual address space for each process, it uses only part of this space. Only 4 TB is allocated for user space. User space is generally private and maps to a nonshared physical page. An additional 4 TB of virtual address space is used for kernel space. Kernel space usually maps to shared physical pages. The remaining space is not used for any purpose.

Figure 6–4 shows the use of process virtual address space.

Figure 6–4: Virtual Address Space Usage



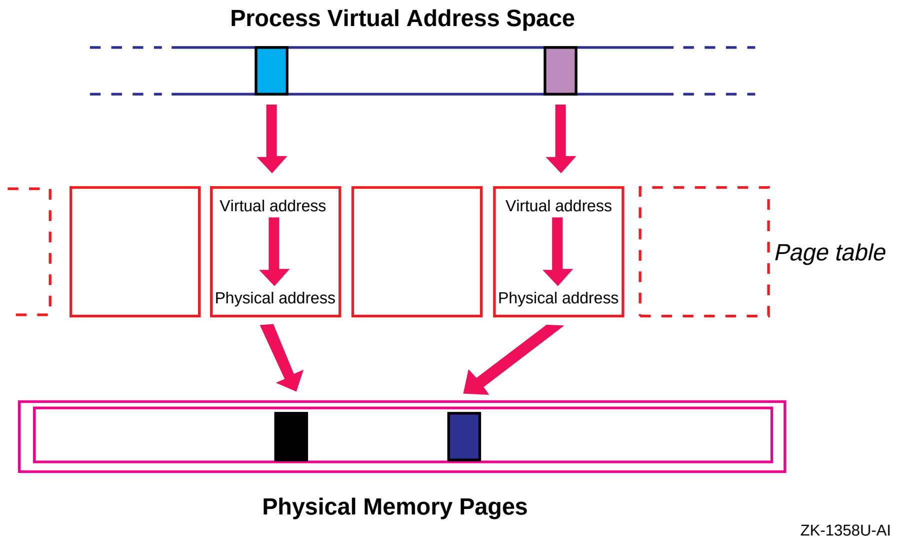
In addition, user space is sparsely populated with valid pages. Only valid pages are able to map to physical pages. The `vm` subsystem attribute `vm-maxvas` specifies the maximum amount of valid virtual address space for a process (that is, the sum of all the valid pages). The default is 128000 pages (1 GB).

6.1.3.2 Translating Virtual Addresses to Physical Addresses

When a virtual page is touched (accessed), the virtual memory subsystem must locate the physical page and then translate the virtual address into a physical address. Each process has a **page table**, which is an array containing an entry for each current virtual-to-physical address translation. Page table entries have a direct relation to virtual pages (that is, virtual address 1 corresponds to page table entry 1) and contain a pointer to the physical page and protection information.

Figure 6–5 shows the translation of a virtual address into a physical address.

Figure 6–5: Virtual-to-Physical Address Translation



A process **resident set** is the complete set of all the virtual addresses that have been mapped to physical addresses (that is, all the pages that have been accessed during process execution). Resident set pages may be shared among multiple processes.

A process **working set** is the set of virtual addresses that are currently mapped to physical addresses. The working set is a subset of the resident set and represents a snapshot of the process resident set at one point in time.

6.1.3.3 Page Faulting

When an anonymous (nonfile-backed) virtual address is requested, the virtual memory subsystem must locate the physical page and make it available to the process. This occurs at different speeds, depending on whether the page is in memory or on disk (see Figure 1–1).

If a requested address is currently being used (that is, the address is in the active page list), it will have an entry in the page table. In this case, the PAL code loads the physical address into the translation lookaside buffer, which then passes the address to the CPU. Because this is a memory operation, it occurs quickly.

If a requested address is not active in the page table, the PAL lookup code issues a **page fault**, which instructs the virtual memory subsystem to locate the page and make the virtual-to-physical address translation in the page table.

There are different types of page faults:

- If a requested virtual address is being accessed for the first time, a **zero-filled-on-demand page fault** occurs. The virtual memory subsystem performs the following tasks:

  1. Allocates an available page of physical memory.
  2. Fills the page with zeros.
  3. Enters the virtual-to-physical address translation in the page table.

- If a requested virtual address has already been accessed and is located in the memory subsystem's internal data structures, a **short page fault** occurs. For example, if the physical address is located in the hash queue list or the page queue list, the virtual memory subsystem passes the address to the CPU and enters the virtual-to-physical address translation in the page table. This occurs quickly because it is a memory operation.

- If a requested virtual address has already been accessed, but the physical page has been reclaimed, the page contents will be found in swap space and a **page-in page fault** occurs.

  The virtual memory subsystem copies the contents of the page from swap space into the physical address and enters the virtual-to-physical address translation in the page table. Because this requires a disk I/O operation, it requires more time than a memory operation.

- If a process needs to modify a read-only virtual page, a **copy-on-write page fault** occurs. The virtual memory subsystem allocates an available page of physical memory, copies the read-only page into the new page, and enters the translation in the page table.

The virtual memory subsystem uses several techniques to improve process execution time and decrease the number of page faults:

- Mapping additional pages

  The virtual memory subsystem attempts to anticipate which pages the task will need next. Using an algorithm that checks which pages were most recently used, the number of available pages, and other factors, the subsystem maps additional pages along with the page that contains the requested address.

- Page coloring

  If possible, the virtual memory subsystem maps a process' entire resident set into the secondary cache and executes the entire task, text, and data within the cache.

  The `vm` subsystem attribute `private-cache-percent` specifies the percentage of the secondary cache that is reserved for anonymous memory and can be used for benchmarking. The default is to reserve 50 percent of the cache for anonymous memory and 50 percent for file-backed memory (shared). To cache more anonymous memory, increase the value of the `private-cache-percent` attribute.

6.1.4 Reclaiming Pages

Because memory resources are limited, the virtual memory subsystem must periodically reclaim pages. The free page list contains clean pages that are available to processes and the UBC. As the demand for memory increases, the list may become depleted. If the number of pages falls below a tunable limit, the virtual memory subsystem will reclaim the least-recently used pages from processes and the UBC to replenish the free list.

To reclaim pages, the virtual memory subsystem:

1. Prewrites modified pages to swap space, in an attempt to forestall a memory shortage. See Section 6.1.4.1 for more information.

2. Begins paging if the demand for memory is not satisfied, as follows:

   a. Reclaims pages that the UBC has borrowed and puts them on the free list.

   b. Reclaims the oldest inactive and UBC LRU pages from the active page list, moves the contents of the modified pages to swap space, and puts the clean pages on the free list.

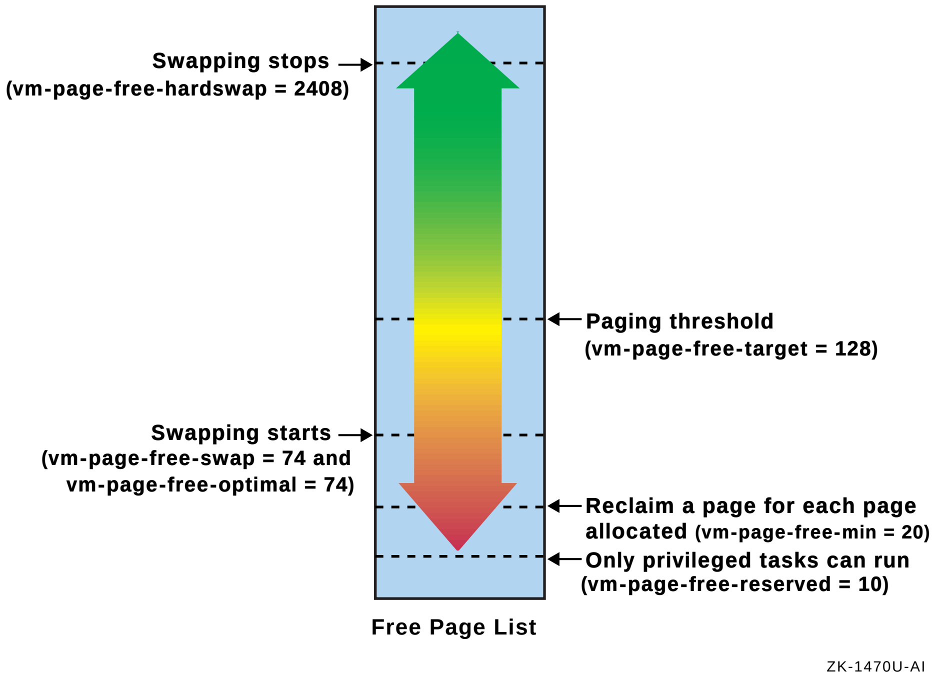
   c. More aggressively reclaims pages from the active list, if needed.

   See Section 6.1.4.2 for more information about paging.

3. Begins swapping if the demand for memory is not met. The virtual memory subsystem temporarily suspends processes and moves entire resident sets to swap space, which frees large numbers of pages. See Section 6.1.4.3 for information about swapping.

The point at which paging and swapping start and stop depends on the values of some `vm` subsystem attributes. Figure 6–6 shows the default values of these attributes.

Figure 6–6: Paging and Swapping Attributes – Default Values


Swapping stops
(vm-page-free-hardswap = 2408)
Paging threshold
(vm-page-free-target = 128)
Swapping starts
(vm-page-free-swap = 74 and
vm-page-free-optimal = 74)
Reclaim a page for each page
allocated (vm-page-free-min = 20)
Only privileged tasks can run
(vm-page-free-reserved = 10)
Free Page List
ZK-1470U-AI

Detailed descriptions of the attributes are as follows:

- `vm-page-free-target`—Paging starts when the number of pages on the free list is less than this value. Paging stops when the number of pages is equal to or more than this value. The default is 128 pages.
- `vm-page-free-min`—Specifies the threshold at which a page must be reclaimed for each page allocated (the default is 20 pages).
- `vm-page-free-swap`—Idle task swapping starts when the number of pages on the free list is less than this value for a period of time (the default is 74 pages).
- `vm-page-free-optimal`—Hard swapping starts when the number of pages on the free list is less than this value for five seconds (the default is 74 pages). The first processes to be swapped out include those with the lowest scheduling priority and those with the largest resident set size.
- `vm-page-free-hardswap`—Swapping stops when the number of pages on the free list is equal to or more than this value (the default is 2048 pages).
- `vm-page-free-reserved`—Only privileged tasks can get memory when the number of pages on the free list is less than this value (the default is 10 pages).

See Section 6.5 for information about modifying paging and swapping attributes.

The following sections describe the page reclamation procedure in detail.

6.1.4.1 Prewriting Modified Pages

The virtual memory subsystem attempts to prevent a memory shortage by prewriting modified pages to swap space.

When the virtual memory subsystem anticipates that the pages on the free list will soon be depleted, it prewrites to swap space the oldest modified (dirty) inactive pages. The value of the `vm` subsystem attribute `vm-page-prewrite-target` determines the number of pages that the subsystem will prewrite and keep clean. The default value is 256 pages.

In addition, when the number of modified UBC LRU pages exceeds the value of the `vm` subsystem attribute `vm-ubcdirtypercent`, the virtual memory subsystem prewrites to swap space the oldest modified UBC LRU pages. The default value of the `vm-ubcdirtypercent` attribute is 10 percent of the total UBC LRU pages.

To minimize the impact of `sync` (steady state flushes) when prewriting UBC pages, the `vm` subsystem attribute `ubc-maxdirtywrites` specifies the maximum number of disk writes that the kernel can perform each second. The default value is 5.

See Section 6.5.2 for information about modifying dirty page prewriting.

6.1.4.2 Reclaiming Memory by Paging

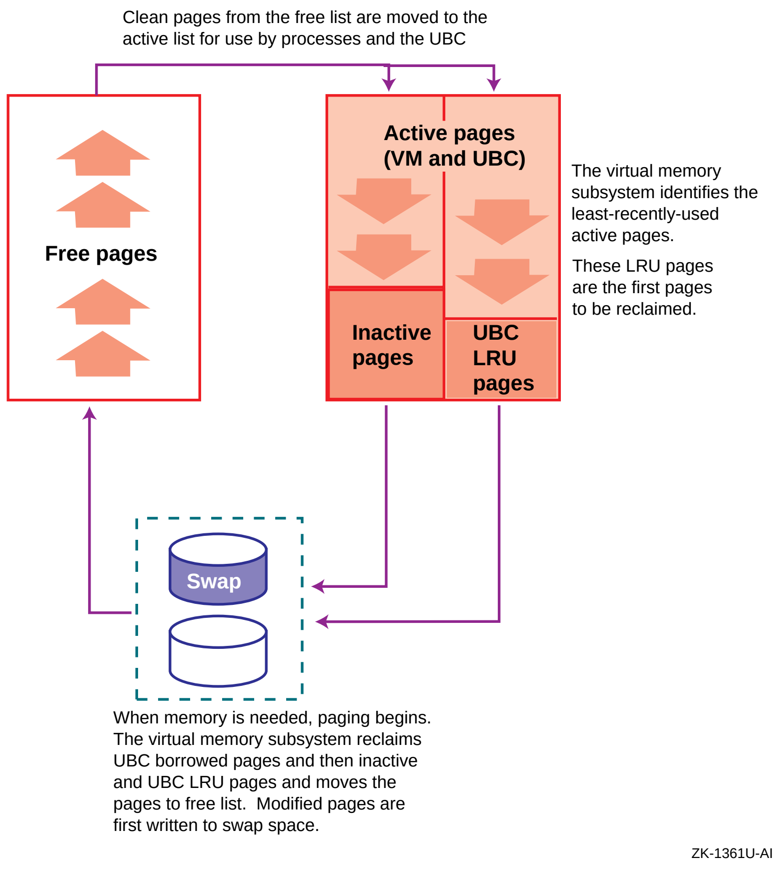
When the memory demand is high and the number of pages on the free page list falls below the value of the `vm` subsystem attribute `vm-page-free-target`, the virtual memory subsystem uses paging to replenish the free page list. The page-out daemon and task swapper daemon are extensions of the page reclamation code, which controls paging and swapping.

The paging process is as follows:

1. The page reclamation code activates the page-stealer daemon, which first reclaims the pages that the UBC has borrowed from the virtual memory subsystem, until the size of the UBC reaches the borrowing threshold that is specified by the value of the `ubc-borrowpercent` attribute (the default is 20 percent). Freeing borrowed UBC pages is a fast way to reclaim pages, because UBC pages are usually not modified. If the reclaimed pages are dirty (modified), their contents must be written to disk before the pages can be moved to the free page list.
2. If freeing UBC borrowed memory does not sufficiently replenish the free list, a **pageout** occurs. The page-stealer daemon reclaims the oldest inactive and UBC LRU pages from the active page list, moves the contents of the modified pages to swap space, and puts the clean pages on the free list.
3. Paging becomes increasingly aggressive if the number of free pages continues to decrease. If the number of pages on the free page list falls below the value of the `vm` subsystem attribute `vm-page-free-min` (the default is 20 pages), a page must be reclaimed for each page taken from the list.

Figure 6–7 shows the movement of pages during paging operations.

Figure 6–7: Paging Operation


Clean pages from the free list are moved to the active list for use by processes and the UBC
Free pages
Active pages (VM and UBC)
Inactive pages
UBC LRU pages
The virtual memory subsystem identifies the least-recently-used active pages.
These LRU pages are the first pages to be reclaimed.
Swap
When memory is needed, paging begins. The virtual memory subsystem reclaims UBC borrowed pages and then inactive and UBC LRU pages and moves the pages to free list. Modified pages are first written to swap space.
ZK-1361U-AI

Paging stops when the number of pages on the free list increases to the limit specified by the `vm` subsystem attribute `vm-page-free-target`. However, if paging individual pages does not sufficiently replenish the free list, swapping is used to free a large amount of memory (see Section 6.1.4.3).

6.1.4.3 Reclaiming Memory by Swapping

If there is a continuously high demand for memory, the virtual memory subsystem may be unable to replenish the free page list by reclaiming single pages. To dramatically increase the number of clean pages, the virtual memory subsystem uses swapping to suspend processes, which reduces the demand for physical memory.

The task swapper will **swap out** a process by suspending the process, writing its resident set to swap space, and moving the clean pages to the free page list. Swapping has a serious impact on system performance because a swapped out process cannot execute, and should be avoided on VLM systems and systems running large programs.

The point at which swapping starts and stops is controlled by a number of `vm` subsystem attributes, as follows:

- **Idle task swapping** begins when the number of pages on the free list falls below the value of the `vm-page-free-swap` attribute for a period of time (the default is 74 pages). The task swapper suspends all tasks that have been idle for 30 seconds or more.
- **Hard task swapping** begins when the number of pages on the free page list falls below the value of the `vm-page-free-optimal` attribute (the default is 74 pages) for more than five seconds. The task swapper suspends, one at a time, the tasks with the lowest priority and the largest resident set size.
- Swapping stops when the number of pages on the free list increases to the value of the `vm-page-free-hardswap` attribute (the default is 2048).
- A **swapin** occurs when the number of pages on the free list increases to the value of the `vm-page-free-optimal` attribute for a period of time. The task's working set is paged in from swap space and it can now execute. The value of the `vm-inswappedmin` attribute specifies the minimum amount of time, in seconds, that a task must remain in the inswapped state before it can be outswapped. The default value is 1 second.

You may be able to improve system performance by modifying the attributes that control when swapping starts and stops, as described in Section 6.5. Large-memory systems or systems running large programs should avoid paging and swapping, if possible.

Increasing the rate of swapping (swapping earlier during page reclamation) may increase throughput. As more processes are swapped out, fewer processes are actually executing and more work is done. Although increasing the rate of swapping moves long-sleeping threads out of memory and frees memory, it may degrade interactive response time because when an outswapped process is needed, it will have a long latency period.

Decreasing the rate of swapping (by swapping later during page reclamation) may improve interactive response time, but at the cost of throughput. See Section 6.5.1 for more information about changing the rate of swapping.

To facilitate the movement of data between memory and disk, the virtual memory subsystem uses synchronous and asynchronous swap buffers. The virtual memory subsystem uses these two types of buffers to immediately satisfy a page-in request without having to wait for the completion of a page-out request, which is a relatively slow process.

Synchronous swap buffers are used for page-in page faults and for swap outs. Asynchronous swap buffers are used for asynchronous pageouts and for prewriting modified pages. See Section 6.5.4 and Section 6.5.5 for swap buffer tuning information.

6.2 Configuring Swap Space for High Performance

Use the `swapon` command to display swap space, and to configure additional swap space after system installation. To make this additional swap space permanent, you must specify the swap file entry in the `/etc/fstab` file.

See Section 2.3.2.2 and Section 2.3.2.3 for information about swap space allocation modes and swap space requirements.

The following list describes how to configure swap space for high performance:

- Ensure that all your swap devices are configured when you boot the system, instead of adding swap space while the system is running.
- Use fast disks for swap space to decrease page-fault latency.
- Use disks that are not busy for swap space. Use the `iostat` command to determine which disks are not being used.
- Spread out swap space across multiple disks; do not put multiple swap partitions on the same disk. This makes paging and swapping more efficient and helps to prevent any single adapter, disk, or bus from becoming a bottleneck. The page reclamation code uses a form of disk striping (known as swap space interleaving) that improves performance when data is written to multiple disks.
- Spread out your swap disks across multiple I/O buses to prevent a single bus from becoming a bottleneck.
- Use the Logical Storage Manager (LSM) to stripe your swap disks.
- Use RAID 1 (mirroring) or RAID 5 for swap disks to provide data availability if a failure occurs. Mirroring may degrade performance for configurations that are paging heavily, because this increases the write load on the swap disks.

See the *System Administration* manual for more information about adding swap devices. See Chapter 8 for more information about configuring and tuning disks for high performance and availability.

6.3 Gathering Memory Information

Table 6–1 describes the tools that you can use to gather information about memory usage.

Table 6–1: Virtual Memory and UBC Monitoring Tools

Name	Use	Description
`sys_check`	Analyzes system configuration and displays statistics (Section 4.2)	Creates an HTML file that describes the system configuration, and can be used to diagnose problems. The `sys_check` utility checks kernel variable settings and memory and CPU resources, and provides performance data and lock statistics for SMP systems and kernel profiles. The `sys_check` utility calls various commands and utilities to perform a basic analysis of your configuration and kernel variable settings, and provides warnings and tuning recommendations if necessary. See `sys_check`(8) for more information.
`uerf`	Displays total system memory	Use the `uerf -r 300` command to determine the amount of memory on your system. The beginning of the listing shows the total amount of physical memory (including wired memory) and the amount of available memory. See `uerf`(8) for more information.
`vmstat`	Displays virtual memory and CPU usage statistics (Section 6.3.2)	Displays information about process threads, virtual memory usage (page lists, page faults, pageins, and pageouts), interrupts, and CPU usage (percentages of user, system and idle times). First reported are the statistics since boot time; subsequent reports are the statistics since a specified interval of time.
`ps`	Displays CPU and virtual memory usage by processes (Section 6.3.1)	Displays current statistics for running processes, including CPU usage, the processor and processor set, and the scheduling priority. The `ps` command also displays virtual memory statistics for a process, including the number of page faults, page reclamations, and pageins; the percentage of real memory (resident set) usage; the resident set size; and the virtual address size.

Table 6–1: Virtual Memory and UBC Monitoring Tools (cont.)

Name	Use	Description
`ipcs`	Displays IPC statistics	Displays interprocess communication (IPC) statistics for currently active message queues, shared-memory segments, semaphores, remote queues, and local queue headers. The information provided in the following fields reported by the `ipcs –a` command can be especially useful: `QNUM`, `CBYTES`, `QBYTES`, `SEGSZ`, and `NSEMS`. See `ipcs`(1) for more information.
`swapon`	Displays information about swap space utilization (Section 6.3.3)	Displays the total amount of allocated swap space, swap space in use, and free swap space, and also displays this information for each swap device. You can also use the `swapon` command to allocate additional swap space.
`(dbx) print vm_perfsum`	Reports virtual memory and UBC statistics (Section 6.3.4 and Section 6.3.5)	You can check virtual memory by using the `dbx print` command to examine the `vm_perfsum` data structure, which contains information about page faults, swap space, the free page list, and UBC page usage.
`memx`	Exercises system memory	Exercises memory by running a number of processes. You can specify the amount of memory to exercise, the number of processes to run, and a file for diagnostic output. Errors are written to a log file. See `memx`(8) for more information.
`shmx`	Exercises shared memory	Exercises shared memory segments by running a `shmxb` process. The `shmx` and `shmxb` processes alternate writing and reading the other process' data in the shared memory segments. You can specify the number of memory segments to test, the size of the segment, and a file for diagnostic output. Errors are written to a log file. See `shmx`(8) for more information.

The following sections describe some of these tools in detail.

6.3.1 Monitoring Memory by Using the ps Command

The `ps` command displays the current status of the system processes. You can use it to determine the current running processes (including users), their state, and how they utilize system memory. The command lists processes in order of decreasing CPU usage, so you can identify which processes are using the most CPU time.

The ps command provides only a snapshot of the system; by the time the command finishes executing, the system state has probably changed. In addition, one of the first lines of the command may refer to the ps command itself.

An example of the ps command is as follows:

```
# /usr/ucb/ps aux
USER  PID  %CPU %MEM   VSZ   RSS  TTY S    STARTED      TIME  COMMAND
chen  2225  5.0  0.3  1.35M  256K p9  U    13:24:58  0:00.36  cp /vmunix /tmp
root  2236  3.0  0.5  1.59M  456K p9  R  + 13:33:21  0:00.08  ps aux
sorn  2226  1.0  0.6  2.75M  552K p9  S  + 13:25:01  0:00.05  vi met.ps
root   347  1.0  4.0  9.58M  3.72 ??  S      Nov 07 01:26:44  /usr/bin/X11/X -a
root  1905  1.0  1.1  6.10M  1.01 ??  R    16:55:16  0:24.79  /usr/bin/X11/dxpa
mat   2228  0.0  0.5  1.82M  504K p5  S  + 13:25:03  0:00.02  more
mat   2202  0.0  0.5  2.03M  456K p5  S    13:14:14  0:00.23  -csh (csh)
root     0  0.0 12.7   356M  11.9 ??  R <  Nov 07 3-17:26:13  [kernel idle]
             1   2      3     4        5                  6        7
```

The ps command output includes the following information that you can use to diagnose CPU and virtual memory problems:

1. Percentage of CPU time usage (%CPU).
2. Percentage of real memory usage (%MEM).
3. Process virtual address size (VSZ)—This is the total amount of virtual memory allocated to the process (in bytes).
4. Real memory (resident set) size of the process (RSS)—This is the total amount of physical memory (in bytes) mapped to virtual pages (that is, the total amount of memory that the application has physically used). Shared memory is included in the resident set size figures; as a result, the total of these figures may exceed the total amount of physical memory available on the system.
5. Process status or state (S)—This specifies whether a process is in one of the following states:
   - Runnable (R)
   - Uninterruptible sleeping (U)
   - Sleeping (S)
   - Idle (I)
   - Stopped (T)
   - Halted (H)
   - Swapped out (W)
   - Has exceeded the soft limit on memory requirements (>)

- A process group leader with a controlling terminal (+)
- Has a reduced priority (N)
- Has a raised priority (<)

6 Current CPU time used (`TIME`), in the format *hh*:*mm*:*ss.ms*.

7 The command that is running (`COMMAND`).

From the output of the `ps` command, you can determine which processes are consuming most of your system's CPU time and memory resources and whether processes are swapped out. Concentrate on processes that are running or paging. Here are some concerns to keep in mind:

- If a process is using a large amount of memory (see the `RSS` and `VSZ` fields), the process may have excessive memory requirements. See Section 11.2 for information about decreasing an application's use of memory.
- Are duplicate processes running? Use the `kill` command to terminate any unnecessary processes. See `kill`(1) for more information.
- If a process is using a large amount of CPU time, it may be in an infinite loop. You may have to use the `kill` command to terminate the process and then correct the problem by making changes to its source code.

  You can also use the Class Scheduler to allocate a percentage of CPU time to a specific task or application (see Section 7.2.2) or lower the process' priority by using either the `nice` or `renice` command. These commands have no effect on memory usage by a process. See `nice`(8) or `renice`(8) for more information.
- Check the processes that are swapped out. Examine the `S` (state) field. A `W` entry indicates a process that has been swapped out. If processes are continually being swapped out, this could indicate a lack of memory resources. See Section 6.4 for information.

6.3.2 Monitoring Memory by Using the vmstat Command

The `vmstat` command shows the virtual memory, process, and CPU statistics for a specified time interval. The first line of the output is for all time since a reboot, and each subsequent report is for the last interval.

Invoke the `vmstat` command when the system is idle and also when the system is busy to compare the resulting data. You can use the `memx` memory exerciser to put a load on the memory subsystem.

An example of the vmstat command is as follows; output is provided in one-second intervals:

```
# /usr/ucb/vmstat 1
Virtual Memory Statistics: (pagesize = 8192)
procs        memory            pages                          intr        cpu
r  w  u   act  free wire  fault cow zero react pin pout   in  sy  cs  us sy  id
2 66 25  6417 3497 1570  155K  38K  50K    0  46K    0    4 290 165   0  2  98
4 65 24  6421 3493 1570   120    9   81    0    8    0  585 865 335  37 16  48
2 66 25  6421 3493 1570    69    0   69    0    0    0  570 968 368   8 22  69
4 65 24  6421 3493 1570    69    0   69    0    0    0  554 768 370   2 14  84
4 65 24  6421 3493 1570    69    0   69    0    0    0  865  1K 404   4 20  76
               [1]                                  [2]      [3]         [4]
```

The vmstat command includes information that you can use to diagnose CPU and virtual memory problems. The following fields are particularly important:

[1] Virtual memory information (memory): the number of pages that are on the active list, including inactive pages and UBC LRU pages (act); the number of pages on the free list (free), and the number of pages on the wired list (wire). Pages on the wired list cannot be reclaimed. See Section 6.1.1 for more information on page lists.

[2] The number of pages that have been paged out (pout).

[3] Interrupt information (intr), including the number of nonclock device interrupts per second (in), the number of system calls called per second (sy), and the number of task and thread context switches per second (cs).

[4] CPU usage information (cpu), including the percentage of user time for normal and priority processes (us), the percentage of system time (sy), and the percentage of idle time (id). User time includes the time the CPU spent executing library routines. System time includes the time the CPU spent executing system calls.

To use the vmstat command to diagnose a performance problem:

- Check the size of the free page list (free). Compare the number of free pages to the values for the active pages (act) and the wired pages (wire). The sum of the free, active, and wired pages should be close to the amount of physical memory in your system. Although the value for free should be small, if the value is consistently small (less than 128 pages) and accompanied by excessive paging and swapping, you may not have enough physical memory for your workload. See Section 6.4 for information about increasing memory resources.
- Examine the pout field. If the number of pageouts is consistently high, you may have insufficient memory. See Section 6.4 for information about increasing memory resources.

You also may have insufficient swap space or your swap space may be configured inefficiently. Use the `swapon -s` command to display your swap device configuration, and use the `iostat` command to determine which swap disk is being used the most. See Section 2.3.2.3 for information about configuring swap space.

- Check the user (`us`), system (`sy`), and idle (`id`) time split. You must understand how your applications use the system to determine the appropriate values for these times. The goal is to keep the CPU as productive as possible. Idle CPU cycles occur when no runnable processes exist or when the CPU is waiting to complete an I/O or memory request.

The following list describes how to interpret the values for user, idle, and system time:

- Idle time—A high percentage of idle time on one or more processors indicates either:
  - Threads are blocked because the CPU is waiting for some event or resource (for example, memory or I/O)
  - Threads are idle because the CPU is not busy

  If you have a high idle time and poor response time, and you are sure that your system has a typical load, one or more of the following problems may exist:
  - The hardware may have reached its capacity
  - One or more kernel data structures is being exhausted
  - You may have a hardware or kernel resource problem such as an application, disk I/O, or network bottleneck

  If the idle time percentage is very low but performance is acceptable, your system is utilizing its CPU efficiently.

- User time—A high percentage of user time can be a characteristic of a well-performing system. However, if the system has poor performance, a high percentage of user time may indicate a user code bottleneck, which can be caused by inefficient user code, insufficient processing power, or excessive memory latency or cache missing.

  Use profiling to determine which sections of code consume the most processing time. See Section 11.1 and the *Programmer's Guide* for more information on profiling.

  A high percentage of user time and a low percentage of idle time may indicate that your application code is consuming most of the CPU. You can optimize the application, or you may need a more powerful processor. See Section 7.2 for information on optimizing CPU resources.

- System time—A high percentage of system time may indicate a system bottleneck, which can be caused by excessive system calls, device interrupts, context switches, soft page faults, lock contention, or cache missing.

  A high percentage of system time and a low percentage of idle time may indicate that something in the application load is stimulating the system with high overhead operations. Such overhead operations could consist of high system call frequencies, high interrupt rates, large numbers of small I/O transfers, or large numbers of IPCs or network transfers. A high system time and low idle time may be caused by failing hardware. Use the `uerf` command to check your hardware.

  A high percentage of system time may also indicate that the system is thrashing; that is, the amount of memory available to the virtual memory subsystem has gotten so low that the system is spending all its time paging and swapping in an attempt to regain memory. A system that spends more than 50 percent of its time in system mode and idle mode may be doing a lot of paging and swapping I/O, and therefore may not have enough memory resources. See Section 6.4 for information about increasing memory resources.

If you have excessive page-in and page-out activity from a swap partition, the system may have a high physical memory commitment ratio. Excessive paging also can increase the miss rate for the secondary cache, and may be indicated by the following output:

- The output of the `ps` command shows high task swapping activity. See Section 6.3.1 for more information.
- The output of the `vmstat` shows a very low free page count or shows high page-in and page-out activity.
- The output of the `swapon` command shows excessive use of swap space. See Section 6.3.3 for more information.

The following command output may indicate that the size of the UBC is too small for your configuration:

- The output of the `vmstat` or `monitor` command shows excessive file system page in activity, but little or no page out activity or shows a very low free page count.
- The output of the `iostat` command shows little or no swap disk I/O activity or shows excessive file system I/O activity. See Section 8.2.1 for more information.

6.3.3 Monitoring Swap Space Usage by Using the swapon Command

Use the `swapon -s` command to display your swap device configuration. For each swap partition, the command displays the total amount of allocated swap space, the amount of swap space that is being used, and the amount of free swap space. This information can help you determine how your swap space is being utilized.

An example of the `swapon` command is as follows:

```
# /usr/sbin/swapon -s

Swap partition /dev/rz1b (default swap):
    Allocated space:        16384 pages (128MB)
    In-use space:           10452 pages ( 63%)
    Free space:              5932 pages ( 36%)

Swap partition /dev/rz4a:
    Allocated space:       128178 pages (1001MB)
    In-use space:           10242 pages (  7%)
    Free space:            117936 pages ( 92%)

Total swap allocation:
    Allocated space:       144562 pages (1.10GB)
    Reserved space:         34253 pages ( 23%)
    In-use space:           20694 pages ( 14%)
    Available space:       110309 pages ( 76%)
```

You can configure swap space when you first install the operating system, or you can add swap space at a later date. Application messages, such as the following, usually indicate that not enough swap space is configured into the system or that a process limit has been reached:

```
   “lack of paging space”
   “swap space below 10 percent free”
```

See Section 2.3.2.3 for information about swap space requirements. See Section 6.2 for information about adding swap space and distributing swap space for high performance.

6.3.4 Monitoring Memory by Using the dbx Debugger

You can check virtual memory by using the `dbx print` command to examine the `vm_perfsum` data structure. For example:

```
# /usr/ucb/dbx –k /vmunix /dev/mem
(dbx) print vm_perfsum
struct {
    vpf_pagefaults = 10079073
    vpf_kpagefaults = 103387
    vpf_cowfaults = 2696851
    vpf_cowsteals = 840487
    .
    .
    .
    vpf_allocatedpages = 7030
    vpf_vmwiredpages = 485
    vpf_ubcwiredpages = 0
    vpf_mallocpages = 924
    vpf_totalptepages = 307
    vpf_contigpages = 7
    vpf_rmwiredpages = 0
    vpf_ubcpages = 3211
    vpf_freepages = 128
    vpf_vmcleanpages = 256
    vpf_swapspace = 7879
}
(dbx)
```

Important fields in the previous example include the following:

- `vpf_pagefaults`—Number of hardware page faults
- `vpf_swapspace`—Number of pages of swap space not reserved
- `vpf_freepages`—Number of pages on the free list

To obtain additional information about the current use of memory, use the `dbx print` command to display the values of the following kernel variables:

- `vm_page_active_count`—Number of pages on the active list
- `vm_page_inactive_count`—Number of inactive pages
- `ubc_lru_page_count`—Number of UBC LRU pages

For example:

```
# /usr/ucb/dbx -k /vmunix /dev/mem
(dbx) print vm_page_active_count
708
```

See Chapter 6 for information on managing memory resources.

6.3.5 Monitoring the UBC by Using the dbx Debugger

The Unified Buffer Cache (UBC) is flushed by the `update` daemon. You can monitor the UBC usage lookup hit ratio by using the `dbx print` command to examine the `vm_perfsum`, `ufs_getapage_stats`, and `vm_tune` data structures.

The following example shows part of the `vm_perfsum` data structure:

```
# /usr/ucb/dbx -k /vmunix /dev/mem
(dbx) print vm_perfsum
struct {
    vpf_pagefaults = 10079139
    vpf_kpagefaults = 103387
    vpf_cowfaults = 2696861
    vpf_cowsteals = 840499
    vpf_zfod = 2332612
    vpf_kzfod = 103217
    vpf_pgiowrites = 28526
    .
    .
    .
    vpf_ubcalloc = 678788
    vpf_ubcpagepushes = 51
    vpf_ubcdirtywra = 8
    vpf_ubcreclaim = 0
    vpf_ubcpagesteal = 330624
    vpf_ubclookups = 7880454
    vpf_ubclookuphits = 7472308
    vpf_allocatedpages = 7030
    vpf_vmwiredpages = 489
    vpf_ubcwiredpages = 0
    vpf_mallocpages = 924
    vpf_totalptepages = 319
    vpf_contigpages = 7
    vpf_rmwiredpages = 0
    vpf_ubcpages = 3179
    vpf_freepages = 128
    vpf_vmcleanpages = 256
    vpf_swapspace = 7877
}
(dbx)
```

Important fields include the following:

- The `vpf_pgiowrites` field specifies the number of I/O operations for pageouts generated by the page stealing daemon.
- The `vpf_ubcalloc` field specifies the number of times the UBC had to allocate a page from the virtual memory free page list to satisfy memory demands.
- The `vpf_ubcpages` field specifies the number of pages of physical memory that the UBC is using to cache file data. If the UBC is using significantly more than half of the physical memory and the paging rate is high (`vpf_pgiowrites` field), you may want to reduce the amount of memory available to the UBC by decreasing the value of the `ubc-maxpercent` attribute. See Section 6.4.4 for information.

- The vpf_ubclookuphits field specifies the UBC hit rate.

You can also monitor the UBC by using the dbx print command to examine the ufs_getapage_stats data structure. For example:

```
# /usr/ucb/dbx -k /vmunix /dev/mem
(dbx) print ufs_getapage_stats
struct {
    read_looks = 2059022
    read_hits = 2022488
    read_miss = 36506
}
(dbx)
```

To calculate the hit rate, divide the value of the read_hits field by the value of the read_looks field. A good hit rate is a rate above 95 percent. In the previous example, the hit rate is approximately 98 percent.

You can also check the UBC by using the dbx print command to examine the vm_tune data structure. For example:

```
# /usr/ucb/dbx -k /vmunix /dev/mem
(dbx) print vm_tune
struct {
    vt_cowfaults = 4
    vt_mapentries = 200
    vt_maxvas = 1073741824
    vt_maxwire = 16777216
    vt_heappercent = 7
    vt_anonklshift = 17
    vt_anonklpages = 1
    vt_vpagemax = 16384
    vt_segmentation = 1
    vt_ubcpagesteal = 24
    vt_ubcdirtypercent = 10
    vt_ubcseqstartpercent = 50
    vt_ubcseqpercent = 10
    vt_csubmapsize = 1048576
    vt_ubcbuffers = 256
    vt_syncswapbuffers = 128
    vt_asyncswapbuffers = 4
    vt_clustermap = 1048576
    vt_clustersize = 65536
    vt_zone_size = 0
    vt_kentry_zone_size = 16777216
    vt_syswiredpercent = 80
    vt_inswappedmin = 1
}
(dbx)
```

Important fields include the `vt_ubcseqpercent` and `vt_ubcseqstartpercent` fields. The values of these fields are used to prevent a large file from completely filling the UBC, which limits the amount of memory available to processes.

When copying large files, the source and destination objects in the UBC will grow very large (up to all of the available physical memory). Reducing the value of the `vm-ubcseqpercent` attribute decreases the number of UBC pages that will be used to cache a large sequentially accessed file. The value represents the percentage of UBC memory that a sequentially accessed file can consume before it starts reusing UBC memory. The value imposes a resident set size limit on a file. See Section 9.2.7 for more information.

6.4 Tuning to Provide More Memory to Processes

If you have insufficient memory for process execution, you may be able to increase the memory that is available to processes by tuning various kernel subsystem attributes. Some of the recommendations for increasing the memory available to processes may impact UBC operation and file system caching.

Table 6–2 shows the recommendations for increasing memory resources to processes and lists the performance benefits as well as tradeoffs.

Table 6–2: Memory Resource Tuning Guidelines

Recommendation	Performance Benefit	Tradeoff
Reduce the number of processes running at the same time (Section 6.4.1)	Decreases CPU load and demand for memory	System performs less work
Reduce the static size of the kernel (Section 6.4.2)	Decreases demand for memory	Not all functionality may be available
Reduce dynamically wired memory (Section 6.4.3)	Decreases demand for memory	None
Reduce the amount of physical memory available to the UBC (Section 6.4.4)	Provides more memory resources to processes	May degrade file system performance
Decrease the size of the AdvFS buffer cache (Section 6.4.5)	Provides more memory resources to processes	May degrade AdvFS performance on systems that open and reuse files
Decrease the size of the metadata buffer cache (Section 6.4.6)	Provides more memory resources to processes	May degrade UFS performance on small systems

Table 6–2: Memory Resource Tuning Guidelines (cont.)

Recommendation	Performance Benefit	Tradeoff
Decrease the size of the namei cache (Section 6.4.7)	Decreases demand for memory	May slow lookup operations and degrade file system performance
Increase the percentage of memory reserved for kernel allocations (Section 6.4.8)	Enables large database programs to run	Applicable only to large database applications
Reduce process memory requirements (Section 11.2.6)	Decreases demand for memory	Program may not run optimally

The following sections describe the recommendations that will increase the memory available to processes in detail.

6.4.1 Reducing the Number of Processes Running Simultaneously

You can improve performance and reduce the demand for memory by running fewer applications simultaneously. Use the `at` or the `batch` command to run applications at offpeak hours.

See `at`(1) for more information.

6.4.2 Reducing the Static Size of the Kernel

You can reduce the static size of the kernel by deconfiguring any unnecessary subsystems. Use the `sysconfig` command to display the configured subsystems and to delete subsystems. Be sure not to remove any subsystems or functionality that is vital to your environment.

See Section 4.4 for information about modifying kernel subsystem attributes.

6.4.3 Reducing Dynamically Wired Memory

You can reduce the amount of dynamically wired memory by reducing the value of the `vm` subsystem attribute `vm-syswiredpercent`. The default value is 80 percent.

You can also reduce dynamically wired memory by allocating more kernel resources to processes (for example, by increasing the value of the `proc` subsystem attribute `maxusers`). See Section 5.1 for information.

See Section 4.4 for information about modifying kernel subsystem attributes.

6.4.4 Decreasing the Memory Available to the UBC

The UBC and processes compete for the memory that is not wired by the kernel. You may be able to improve process performance by reducing the percentage of virtual memory that is available for the UBC. This will increase the amount of memory available to processes, which may reduce the paging and swapping rate.

Reducing the memory allocated to the UBC may adversely affect I/O performance because the UBC will hold less file system data, which results in more disk I/O operations. Therefore, do not significantly decrease the maximum size of the UBC.

The maximum amount of virtual memory that can be allocated to the UBC is specified by the `vm` subsystem attribute `ubc-maxpercent`. The default is 100 percent. The minimum amount of memory that can be allocated to the UBC is specified by the `vm` subsystem attribute `ubc-minpercent`. The default is 10 percent. These default values are appropriate for most configurations, including Internet servers.

Use the `vmstat` command to determine whether the system is paging excessively. Use the `dbx print` command to periodically examine the `vm_perfsum` data structure, especially the `vpf_pgiowrites` and `vpf_ubcalloc` fields. The page-out rate may shrink if pageouts greatly exceed UBC allocations.

If the page out rate is high and you are not using the file system heavily, decreasing the value of the `ubc-maxpercent` attribute may reduce the rate of paging and swapping. Start with the default value of 100 percent and decrease the value in increments of 10. If the values of the `ubc-maxpercent` and `ubc-minpercent` attributes are close together, you may seriously degrade I/O performance or cause the system to page excessively.

You also may be able to prevent paging by increasing the percentage of memory that the UBC borrows from the virtual memory subsystem. To do this, decrease the value of the `ubc-borrowpercent` attribute so that less memory remains in the UBC when page reclamation begins. Although this can reduce the UBC effectiveness, it may improve the system response time when memory is low. The value of the `ubc-borrowpercent` attribute can range from 0 to 100. The default value is 20 percent.

See Section 4.4 for information about modifying kernel subsystem attributes.

6.4.5 Decreasing the Size of the AdvFS Buffer Cache

To free memory resources, you may want to decrease the amount of memory allocated to the AdvFS buffer cache. Decreasing the cache size also decreases the overhead associated with managing the cache. The `advfs` subsystem attribute `AdvfsCacheMaxPercent` determines the maximum amount of memory that can be used for the AdvFS buffer cache. The default is 7 percent of physical memory. The minimum is 1 percent, and the maximum is 30 percent.

If you are not using AdvFS or if you do not reuse many files, decrease the cache size to 1 percent. If you are using AdvFS, but you have a VLM system, you may also want to decrease the cache size.

However, decreasing the size of the AdvFS buffer cache may adversely affect AdvFS I/O performance if you access and then reuse many files.

See Section 4.4 for information about modifying kernel subsystem attributes.

6.4.6 Decreasing the Size of the Metadata Buffer Cache

The metadata buffer cache contains recently accessed UFS and CDFS metadata. If you have a high cache hit rate, you may want to decrease the size of the metadata buffer cache. This will increase the amount of memory that is available to the virtual memory subsystem. However, decreasing the size of the cache may degrade UFS performance.

The `vfs` subsystem attribute `bufcache` specifies the percentage of physical memory that the kernel wires for the metadata buffer cache. The default size of the metadata buffer cache is 3 percent of physical memory. You can decrease the value of the `bufcache` attribute to a minimum of 1 percent.

For VLM systems and systems that use only AdvFS, set the value of the `bufcache` attribute to 1 percent.

See Section 4.4 for information about modifying kernel subsystem attributes.

6.4.7 Decreasing the Size of the namei Cache

The namei cache is used by all file systems to map file pathnames to inodes. Monitor the cache by using the `dbx print` command to examine the `nchstats` data structure.

To free memory resources, decrease the number of elements in the namei cache by decreasing the value of the `vfs` subsystem attribute name-cache-size. The default value is 2*nvnode*11/10. The maximum value is 2*max-vnodes*11/10.

Make sure that decreasing the size of the namei cache does not degrade file system performance.

See Section 4.4 for information about modifying kernel subsystem attributes.

6.4.8 Increasing the Memory Reserved for Kernel Allocations

If you are running a large database application, you may receive the following console message:

```
malloc_wait:X : no space in map.
```

If you receive this message, you may want to increase the size of the kernel malloc map by increasing the percentage of physical memory reserved for kernel memory allocations that are less than or equal to the page size (8 KB). To do this, increase the value of the generic subsystem attribute kmemreserve-percent.

The default value of the kmemreserve-percent attribute is 0, which means that the percentage of reserved physical memory will be 0.4 percent of available memory or 256, whichever is the smallest value. Increase the value of the kmemreserve-percent attribute by increments of 25 until the message no longer appears.

In addition, you may want to increase the value of the kmemreserve-percent attribute if the output of the vmstat command shows dropped packets under the fail_nowait heading. This may occur under a heavy network load.

See Section 4.4 for information about modifying kernel subsystem attributes.

6.5 Tuning Paging and Swapping Operation

You may be able to improve performance by modifying paging and swapping operations. VLM systems should avoid paging and swapping.

Table 6–3 describes the recommendations for controlling paging and swapping and lists the performance benefits and any tradeoffs.

Table 6–3: Paging and Swapping Tuning Guidelines

Action	Performance Benefit	Tradeoff
Increase the rate of swapping (Section 6.5.1)	Increases process throughput	Decreases interactive response performance
Decrease the rate of swapping (Section 6.5.1)	Improves process interactive response performance	Decreases process throughput
Increase the rate of dirty page prewriting (Section 6.5.2)	Prevents drastic performance degradation when memory is exhausted	Decreases peak workload performance
Decrease the rate of dirty page prewriting (Section 6.5.2)	Improves peak workload performance	May cause drastic performance degradation when memory is exhausted
Increase the size of the page-in and page-out clusters (Section 6.5.3)	Improves peak workload performance	Decreases total system workload performance
Decrease the size of the page-in and page-out clusters (Section 6.5.3)	Improves total system workload performance	Decreases peak workload performance
Increase the swap device I/O queue depth for pageins and swapouts (Section 6.5.4)	Increases overall system throughput	Consumes memory
Decrease the swap device I/O queue depth for pageins and swapouts (Section 6.5.4)	Improves the interactive response time and frees memory	Decreases system throughput
Increase the swap device I/O queue depth for pageouts (Section 6.5.5)	Frees memory and increases throughput	Decreases interactive response performance
Decrease the swap device I/O queue depth for pageouts (Section 6.5.5)	Improves interactive response time	Consumes memory
Increase the paging threshold (Section 6.5.6)	Maintains performance when free memory is exhausted	May waste memory
Enable aggressive swapping (Section 6.5.7)	Improves system throughput	Degrades interactive response performance

The following sections describe the recommendations for controlling paging and swapping in detail.

6.5.1 Changing the Rate of Swapping

Swapping has a drastic impact on system performance. You can modify kernel subsystem attributes to control when swapping begins and ends. VLM systems and systems running large programs should avoid swapping.

Increasing the rate of swapping (swapping earlier during page reclamation), moves long-sleeping threads out of memory, frees memory, and increases throughput. As more processes are swapped out, fewer processes are actually executing and more work is done. However, when an outswapped process is needed, it will have a long latency, so increasing the rate of swapping will degrade interactive response time.

To increase the rate of swapping, increase the value of the `vm` subsystem attribute `vm-page-free-optimal` (the default is 74 pages). Increase the value only by 2 pages at a time. Do not specify a value that is more than the value of the `vm` subsystem attribute `vm-page-free-target`.

If you decrease the rate of swapping (swap later during page reclamation), you will improve interactive response time, but at the cost of throughput. To decrease the rate of swapping, decrease the value of the `vm-page-free-optimal` attribute by 2 pages at a time. Do not specify a value that is less than the value of the `vm` subsystem attribute `vm-page-free-min` (the default is 20).

See Section 4.4 for information about modifying kernel subsystem attributes.

6.5.2 Controlling Dirty Page Prewriting

The virtual memory subsystem attempts to prevent a memory shortage by prewriting modified pages to swap space. When the virtual memory subsystem anticipates that the pages on the free list will soon be depleted, it prewrites to swap space the oldest modified (dirty) pages on the inactive list. To reclaim a page that has been prewritten, the virtual memory subsystem only needs to validate the page.

Increasing the rate of dirty page prewriting will reduce peak workload performance, but it will prevent a drastic performance degradation when memory is exhausted. Decreasing the rate will improve peak workload performance, but it will cause a drastic performance degradation when memory is exhausted.

You can control the rate of dirty page prewriting by modifying the values of the `vm` subsystem attributes `vm-page-prewrite-target` and `vm-ubcdirtypercent`.

The `vm-page-prewrite-target` attribute specifies the number of virtual memory pages that the subsystem will prewrite and keep clean. The default value is 256 pages. To increase the rate of virtual memory dirty page prewriting, increase the value of the `vm-page-prewrite-target` attribute from the default value (256) by increments of 64 pages.

The `vm-ubcdirtypercent` attribute specifies the percentage of UBC LRU pages that can be modified before the virtual memory subsystem prewrites the dirty UBC LRU pages. The default value is 10 percent of the total UBC LRU pages (that is, 10 percent of the UBC LRU pages must be dirty before the UBC LRU pages are prewritten). To increase the rate of UBC LRU dirty page prewriting, decrease the value of the `vm-ubcdirtypercent` attribute by increments of 1 percent.

In addition, you may want to minimize the impact of I/O spikes caused by the `sync` function when prewriting UBC LRU dirty pages. The value of the `vm` subsystem attribute `ubc-maxdirtywrites` specifies the maximum number of disk writes that the kernel can perform each second. The default value of the `ubc-maxdirtywrites` attribute is five I/O operations per second.

To minimize the impact of `sync` (steady state flushes) when prewriting dirty UBC LRU pages, increase the value of the `ubc-maxdirtywrites` attribute.

See Section 4.4 for information about modifying kernel subsystem attributes.

6.5.3 Modifying the Size of the Page-In and Page-Out Clusters

The virtual memory subsystem reads in and writes out additional pages in an attempt to anticipate pages that it will need.

The `vm` subsystem attribute `vm-max-rdpgio-kluster` specifies the maximum size of an anonymous page-in cluster. The default value is 16 KB (2 pages). If you increase the value of this attribute, the system will spend less time page faulting because more pages will be in memory. This will increase the peak workload performance, but will consume more memory and decrease the total system workload performance.

Decreasing the value of the `vm-max-rdpgio-kluster` attribute will conserve memory and increase the total system workload performance, but will increase paging and decrease the peak workload performance.

The `vm` subsystem attribute `vm-max-wrpgio-kluster` specifies the maximum size of an anonymous page-out cluster. The default value is 32 KB (4 pages). Increasing the value of this attribute improves the peak workload performance and conserves memory, but causes more pageins and decreases the total system workload performance.

Decreasing the value of the `vm-max-wrpgio-kluster` attribute improves the total system workload performance and decreases the number of pageins, but decreases the peak workload performance and consumes more memory.

See Section 4.4 for information about modifying kernel subsystem attributes.

6.5.4 Modifying the Swap I/O Queue Depth for Pageins and Swapouts

Synchronous swap buffers are used for page-in page faults and for swapouts. The `vm` subsystem attribute `vm-syncswapbuffers` specifies the maximum swap device I/O queue depth for pageins and swapouts. The value should be equal to the approximate number of simultaneously running processes that the system can easily handle. The default is 128.

Increasing the swap device I/O queue depth increases overall system throughput, but it consumes memory.

Decreasing the swap device I/O queue depth decreases memory demands and improves interactive response time, but it decreases overall system throughput.

See Section 4.4 for information about modifying kernel subsystem attributes.

6.5.5 Modifying the Swap I/O Queue Depth for Pageouts

Asynchronous swap buffers are used for asynchronous pageouts and for prewriting modified pages. The `vm` subsystem attribute `vm-asyncswapbuffers` controls the maximum depth of the swap device I/O queue for pageouts.

The value of the `vm-asyncswapbuffers` attribute should be the approximate number of I/O transfers that a swap device can handle at one time. The default value is 4.

Increasing the queue depth will free memory and increase the overall system throughput.

Decreasing the queue depth will use more memory, but it will improve the interactive response time.

If you are using LSM, you may want to increase the page-out rate. Be careful if you increase the value of the `vm-asyncswapbuffers` attribute, because this will cause page-in requests to lag asynchronous page-out requests.

See Section 4.4 for information about modifying kernel subsystem attributes.

6.5.6 Increasing the Paging Threshold

The `vm` subsystem attribute `vm-page-free-target` specifies the minimum number of pages on the free list before paging starts. The default value is 128 pages.

If you have sufficient memory resources, you may want to increase the value of the `vm-page-free-target` attribute. Increasing the paging threshold will increase paging activity, but it may improve performance when free memory is exhausted. However, an excessively high value can waste memory.

If you want to increase the value of the `vm-page-free-target` attribute, start at the default value and then double the value. If you have up to 1 GB of memory, you may want to use a value of 256. If you have up to 4 GB of memory, you may want to use a value of 768. Do not specify a value that is more than 1024 pages or 8 MB.

Do not decrease the value of the `vm-page-free-target` attribute unless you have a lot of memory, or unless you experience a serious performance degradation when free memory is exhausted.

See Section 4.4 for information about modifying kernel subsystem attributes.

6.5.7 Enabling Aggressive Task Swapping

You can enable the `vm` subsystem attribute `vm-aggressive-swap` (set the value to 1) to allow the virtual memory subsystem to aggressively swap out processes when memory is needed. This improves system throughput, but it degrades the interactive response performance.

By default, the `vm-aggressive-swap` attribute is disabled (set to 0), which results in less aggressive swapping. In this case, processes are swapped in at a faster rate than if aggressive swapping is enabled.

See Section 4.4 for information about modifying kernel subsystem attributes.

6.6 Tuning to Reserve Physical Memory for Shared Memory

Granularity hints allow you to reserve a portion of dynamically wired physical memory at boot time for shared memory. This functionality allows the translation lookaside buffer to map more than a single page and enables shared page table entry functionality, which will cause fewer buffer misses.

On typical database servers, using granularity hints provides a 2 to 4 percent run-time performance gain that reduces the shared memory detach time. In most cases, use the Segmented Shared Memory (SSM) functionality (the default) instead of the granularity hints functionality.

To enable granularity hints, you must specify a value for the `vm` subsystem attribute `gh-chunks`. In addition, to make granularity hints more effective, modify applications to ensure that both the shared memory segment starting address and size are aligned on an 8-MB boundary.

Section 6.6.1 and Section 6.6.2 describe how to enable granularity hints.

6.6.1 Tuning the Kernel to Use Granularity Hints

To use granularity hints, you must specify the number of 4-MB chunks of physical memory to reserve for shared memory at boot time. This memory cannot be used for any other purpose and cannot be returned to the system or reclaimed.

To reserve memory for shared memory, specify a nonzero value for the `gh-chunks` attribute. For example, if you want to reserve 4 GB of memory, specify 1024 for the value of `gh-chunks` (1024 * 4 MB = 4 GB). If you specify a value of 512, you will reserve 2 GB of memory.

The value you specify for the `gh-chunks` attribute depends on your database application. Do not reserve an excessive amount of memory, because this decreases the memory available to processes and the UBC.

Note

If you enable granularity hints, disable the use of segmented shared memory by setting the value of the `ipc` subsystem attribute `ssm-threshold` attribute to zero.

You can determine if you have reserved the appropriate amount of memory. For example, you can initially specify 512 for the value of the `gh-chunks` attribute. Then, invoke the following sequence of `dbx` commands while running the application that allocates shared memory:

```
# /usr/ucb/dbx -k /vmunix /dev/mem

(dbx) px &gh_free_counts
0xfffffc0000681748
(dbx) 0xfffffc0000681748/4X
fffffc0000681748:  0000000000000402 0000000000000004
fffffc0000681758:  0000000000000000 0000000000000002
(dbx)
```

The previous output shows the following:

- The first number (402) specifies the number of 512-page chunks (4 MB).
- The second number (4) specifies the number of 64-page chunks.
- The third number (0) specifies the number of 8-page chunks.
- The fourth number (2) specifies the number of 1-page chunks.

To save memory, you can reduce the value of the `gh-chunks` attribute until only one or two 512-page chunks are free while the application that uses shared memory is running.

The following `vm` subsystem attributes also affect granularity hints:

- `gh-min-seg-size`

  Specifies the shared memory segment size above which memory is allocated from the memory reserved by the `gh-chunks` attribute. The default is 8 MB.

- `gh-fail-if-no-mem`

  When set to 1 (the default), the `shmget` function returns a failure if the requested segment size is larger than the value specified by the `gh-min-seg-size` attribute, and if there is insufficient memory in the `gh-chunks` area to satisfy the request.

  If the value of the `gh-fail-if-no-mem` attribute is 0, the entire request will be satisfied from the pageable memory area if the request is larger than the amount of memory reserved by the `gh-chunks` attribute.

In addition, messages will display on the system console indicating unaligned size and attach address requests. The unaligned attach messages are limited to one per shared memory segment.

See Section 4.4 for information about modifying kernel subsystem attributes.

6.6.2 Modifying Applications to Use Granularity Hints

You can make granularity hints more effective by making both the shared memory segment starting address and size aligned on an 8-MB boundary.

To share Level 3 page table entries, the shared memory segment attach address (specified by the shmat function) and the shared memory segment size (specified by the shmget function) must be aligned on an 8-MB boundary. This means that the lowest 23 bits of both the address and the size must be zero.

The attach address and the shared memory segment size is specified by the application. In addition, System V shared memory semantics allow a maximum shared memory segment size of 2 GB minus 1 byte. Applications that need shared memory segments larger than 2 GB can construct these regions by using multiple segments. In this case, the total shared memory size specified by the user to the application must be 8-MB aligned. In addition, the value of the shm-max attribute, which specifies the maximum size of a System V shared memory segment, must be 8-MB aligned.

If the total shared memory size specified to the application is greater than 2 GB, you can specify a value of 2139095040 (or 0x7f800000) for the value of the shm-max attribute. This is the maximum value (2 GB minus 8 MB) that you can specify for the shm-max attribute and still share page table entries.

Use the following dbx command sequence to determine if page table entries are being shared:

```
# /usr/ucb/dbx -k /vmunix /dev/mem

(dbx) p *(vm_granhint_stats *)&gh_stats_store
 struct {
     total_mappers = 21
     shared_mappers = 21
     unshared_mappers = 0
     total_unmappers = 21
     shared_unmappers = 21
     unshared_unmappers = 0
     unaligned_mappers = 0
     access_violations = 0
     unaligned_size_requests = 0
     unaligned_attachers = 0
     wired_bypass = 0
     wired_returns = 0
 }
 (dbx)
```

For the best performance, the shared_mappers kernel variable should be equal to the number of shared memory segments, and the unshared_mappers, unaligned_attachers, and unaligned_size_requests variables should be zero.

Because of how shared memory is divided into shared memory segments, there may be some unshared segments. This occurs when the starting address or the size is aligned on an 8-MB boundary. This condition may be unavoidable in some cases. In many cases, the value of total_unmappers will be greater than the value of total_mappers.

Shared memory locking changes a lock that was a single lock into a hashed array of locks. The size of the hashed array of locks can be modified by modifying the value of the `vm` subsystem attribute `vm-page-lock-count`. The default value is zero.

7 Managing CPU Performance

You may be able to improve performance by optimizing CPU resources. This chapter describes how to perform the following tasks:

- Obtain information about CPU performance (Section 7.1)
- Improve CPU performance (Section 7.2)

7.1 Gathering CPU Performance Information

Table 7–1 describes the tools you can use to gather information about CPU usage.

Table 7–1: CPU Monitoring Tools

Name	Use	Description
`sys_check`	Analyzes system configuration and displays statistics (Section 4.2)	Creates an HTML file that describes the system configuration, and can be used to diagnose problems. The utility checks kernel variable settings and memory and CPU resources, and provides performance data and lock statistics for SMP systems and kernel profiles. The `sys_check` utility performs a basic analysis of your configuration and kernel variable settings and provides warnings and tuning recommendations if necessary. See `sys_check`(8) for more information.
`ps`	Displays CPU and virtual memory usage by processes (Section 6.3.1)	Displays current statistics for running processes, including CPU usage, the processor and processor set, and the scheduling priority. The `ps` command also displays virtual memory statistics for a process, including the number of page faults, page reclamations, and pageins; the percentage of real memory (resident set) usage; the resident set size; and the virtual address size.

Table 7–1: CPU Monitoring Tools (cont.)

Name	Use	Description
`Process Tuner`	Displays CPU and virtual memory usage by processes	Displays current statistics for running processes. Invoke the Process Tuner from the CDE Application Manager to display a list of processes and their characteristics, display the processes running for yourself or all users, display and modify process priorities, or send a signal to a process. While monitoring processes, you can select parameters to view (percent of CPU usage, virtual memory size, state, and `nice` priority) and also sort the view.
`vmstat`	Displays virtual memory and CPU usage statistics (Section 6.3.2)	Displays information about process threads, virtual memory usage (page lists, page faults, pageins, and pageouts), interrupts, and CPU usage (percentages of user, system and idle times). First reported are the statistics since boot time; subsequent reports are the statistics since a specified interval of time.
`monitor`	Collects performance data	Collects a variety of performance data on a running system and either displays the information in a graphical format or saves it to a binary file. The `monitor` command is available on the Tru64 UNIX Freeware CD-ROM. See `ftp://gatekeeper.dec.com/pub/DEC` for information.
`top`	Provides a continuous report on the system	Provides continuous reports on the state of the system, including a list of the processes using the most CPU resources. The `top` command is available on the Tru64 UNIX Freeware CD-ROM. See `ftp://eecs.nwu.edu/pub/top` for information.
`ipcs`	Displays IPC statistics	Displays interprocess communication (IPC) statistics for currently active message queues, shared-memory segments, semaphores, remote queues, and local queue headers. The information provided in the following fields reported by the `ipcs –a` command can be especially useful: `QNUM`, `CBYTES`, `QBYTES`, `SEGSZ`, and `NSEMS`. See `ipcs`(1) for more information.

Table 7–1: CPU Monitoring Tools (cont.)

Name	Use	Description
`uptime`	Displays the system load average (Section 7.1.3)	Displays the number of jobs in the run queue for the last 5 seconds, the last 30 seconds, and the last 60 seconds. The `uptime` command also shows the number of users logged into the system and how long a system has been running.
`w`	Reports system load averages and user information	Displays the current time, the amount of time since the system was last started, the users logged in to the system, and the number of jobs in the run queue for the last 5 seconds, 30 seconds, and 60 seconds. The `w` command also displays information about system users, including login and process information. See `w`(1) for more information.
`xload`	Monitors the system load average	Displays the system load average in a histogram that is periodically updated. See `xload`(1X) for more information.
`(kdbx) cpustat`	Reports CPU statistics (Section 7.1.4)	Displays CPU statistics, including the percentages of time the CPU spends in various states.
`(kdbx) lockstats`	Reports lock statistics (Section 7.1.5)	Displays lock statistics for each lock class on each CPU in the system.

The following sections describe some of these commands in detail.

7.1.1 Monitoring CPU Usage by Using the ps Command

The `ps` command displays a snapshot of the current status of the system processes. You can use it to determine the current running processes (including users), their state, and how they utilize system memory. The command lists processes in order of decreasing CPU usage so you can identify which processes are using the most CPU time.

See Section 6.3.1 for detailed information about using the `ps` command to diagnose CPU performance problems.

7.1.2 Monitoring CPU Statistics by Using the vmstat Command

The `vmstat` command shows the virtual memory, process, and CPU statistics for a specified time interval. The first line of output displays statistics since reboot time; each subsequent line displays statistics since the specified time interval.

See Section 6.3.2 for detailed information about the using the `vmstat` command to diagnose performance problems.

7.1.3 Monitoring the Load Average by Using the uptime Command

The `uptime` command shows how long a system has been running and the load average. The load average counts jobs that are waiting for disk I/O, and applications whose priorities have been changed with either the `nice` or the `renice` command. The load average numbers give the average number of jobs in the run queue for the last 5 seconds, the last 30 seconds, and the last 60 seconds.

An example of the `uptime` command is as follows:

```
# /usr/ucb/uptime
1:48pm  up 7 days,  1:07,  35 users,  load average: 7.12, 10.33, 10.31
```

The command output displays the current time, the amount of time since the system was last started, the number of users logged into the system, and the load averages for the last 5 seconds, the last 30 seconds, and the last 60 seconds.

From the command output, you can determine whether the load is increasing or decreasing. An acceptable load average depends on your type of system and how it is being used. In general, for a large system, a load of 10 is high, and a load of 3 is low. Workstations should have a load of 1 or 2.

If the load is high, look at what processes are running with the `ps` command. You may want to run some applications during offpeak hours.

You can also lower the priority of applications with the `nice` or `renice` command to conserve CPU cycles. See `nice`(1) and `renice`(8)for more information.

7.1.4 Checking CPU Usage by Using the kdbx Debugger

The `kdbx` debugger `cpustat` extension displays CPU statistics, including the percentages of time the CPU spends in the following states:

- Running user-level code
- Running system-level code
- Running at a priority set with the `nice` function
- Idle
- Waiting (idle with input or output pending)

The cpustat extension to the kdbx debugger can help application developers determine how effectively they are achieving parallelism across the system.

By default, the kdbx cpustat extension displays statistics for all CPUs in the system. For example:

```
# /usr/bin/kdbx -k /vmunix /dev/mem
(kdbx)cpustat
 Cpu   User (%)    Nice (%) System (%)  Idle (%)   Wait (%)
===== ========== ========== ========== ========== ==========
    0       0.23       0.00       0.08      99.64       0.05
    1       0.21       0.00       0.06      99.68       0.05
```

See the *Kernel Debugging* manual and kdbx(8) for more information.

7.1.5 Checking Lock Usage by Using the kdbx Debugger

The kdbx debugger lockstats extension displays lock statistics for each lock class on each CPU in the system, including the following information:

- Address of the structure
- Class of the lock for which lock statistics are being recorded
- CPU for which the lock statistics are being recorded
- Number of instances of the lock
- Number of times that processes have tried to get the lock
- Number of times that processes have tried to get the lock and missed
- Percentage of time that processes miss the lock
- Total time that processes have spent waiting for the lock
- Maximum amount of time that a single process has waited for the lock
- Minimum amount of time that a single process has waited for the lock

For example:

```
# /usr/bin/kdbx -k /vmunix /dev/mem
(kdbx)lockstats
```

See the *Kernel Debugging* manual and kdbx(8) for more information.

7.2 Improving CPU Performance

A system must be able to efficiently allocate the available CPU cycles among competing processes to meet the performance needs of users and applications. You may be able to improve performance by optimizing the CPU usage.

Table 7–2 describes the recommendations for improving CPU performance.

Table 7–2: Primary CPU Performance Improvement Guidelines

Recommendations	Performance Benefit	Tradeoff
Add processors (Section 7.2.1)	Increases CPU resources	Applicable only for multiproccessing systems and may impact virtual memory performance
Use the Class Scheduler (Section 7.2.2)	Allocates CPU resources to critical applications	None
Prioritize jobs (Section 7.2.3)	Ensures that important applications have the highest priority	None
Schedule jobs at offpeak hours (Section 7.2.4)	Distributes the system load	None
Stop the `advfsd` daemon (Section 7.2.5)	Decreases demand for CPU power	Applicable only if you are not using the AdvFS graphical user interface
Use hardware RAID (Section 7.2.6)	Relieves the CPU of disk I/O overhead and provides disk I/O performance improvements	Increases costs

The following sections describe how to optimize your CPU resources. If optimizing CPU resources does not solve the performance problem, you may have to upgrade your CPU to a faster processor.

7.2.1 Adding Processors

Multiprocessing systems allow you to expand the computing power of a system by adding processors. Workloads that benefit most from multiprocessing have multiple processes or multiple threads of execution that can run concurrently, such as database management system (DBMS) servers, Internet servers, mail servers, and compute servers.

You may be able to improve the performance of a multiprocessing system that has only a small percentage of idle time by adding processors. Before you add processors, you must ensure that a performance problem is not caused by the virtual memory or I/O subsystems. For example, increasing the number of processors will not improve performance in a system that lacks sufficient memory resources.

In addition, increasing the number of processors may increase the demands on your I/O and memory subsystems and could cause bottlenecks.

If you add processors and your system is metadata-intensive (that is, it opens large numbers of small files and accesses them repeatedly), you can improve the performance of synchronous write operations by using Prestoserve (see Section 2.4.8), or by using a RAID controller with a write-back cache (see Section 8.4).

7.2.2 Using the Class Scheduler

Use the Class Scheduler to allocate a percentage of CPU time to specific tasks or applications. This allows you to reserve CPU time for important processes, while limiting CPU usage by less critical processes.

To use class scheduling, group together processes into classes and assign each class a percentage of CPU time. You can also manually assign a class to any process.

The Class Scheduler allows you to display statistics on the actual CPU usage for each class.

See `class_scheduling`(4), `class_admin`(8), `runclass`(1), and `classcntl`(2) for more information about the Class Scheduler.

7.2.3 Prioritizing Jobs

You can prioritize jobs so that important applications are run first. Use the `nice` command to specify the priority for a command. Use the `renice` command to change the priority of a running process.

See `nice`(1) and `renice`(8) for more information.

7.2.4 Scheduling Jobs at Offpeak Hours

You can schedule jobs so that they run at offpeak hours (use the `at` and `cron` commands) or when the load level permits (use the `batch` command). This can relieve the load on the CPU and the memory and disk I/O subsystems.

See `at`(1) and `cron`(8) for more information.

7.2.5 Stopping the advfsd Daemon

The `advfsd` daemon allows Simple Network Management Protocol (SNMP) clients such as Netview or Performance Manager (PM) to request AdvFS file system information. If you are not using the AdvFS graphical user interface (GUI), you can free CPU resources and prevent the `advfsd` daemon from periodically scanning disks by stopping the `advfsd` daemon.

To prevent the `advfsd` daemon from starting at boot time, rename `/sbin/rc3.d/S53advfsd` to `/sbin/rc3.d/T53advfsd`. To immediately stop the daemon, use the `kill -9` *`pid`* command, where *`pid`* specifies the daemon's process identification number (PID).

7.2.6 Using Hardware RAID to Relieve the CPU of I/O Overhead

RAID controllers can relieve the CPU of the disk I/O overhead, in addition to providing many disk I/O performance-enhancing features. See Section 8.4 for more information about hardware RAID.

8

Managing Disk Storage Performance

There are various ways that you can manage your disk storage. Depending on your performance and availability needs, you can use static disk partitions, the Logical Storage Manager (LSM), hardware RAID, or a combination of these solutions.

The disk storage configuration can have a significant impact on system performance, because disk I/O is used for file system operations and also by the virtual memory subsystem for paging and swapping.

You may be able to improve disk I/O performance by following the configuration and tuning guidelines described in this chapter, which describes how to perform the following tasks:

- Improve performance by efficiently distributing the disk I/O load (Section 8.1)
- Obtain information about the disk storage configuration and performance (Section 8.2)
- Manage LSM performance (Section 8.3)
- Improve hardware RAID subsystem performance (Section 8.4)
- Tune the Common Access Method (CAM) (Section 8.5)

To configure a disk storage subsystem that will meet your performance and availability needs, you must first understand your workload resource model, as described in Section 2.1.

8.1 Distributing the Disk I/O Load

Distributing the disk I/O load across devices helps to prevent a single disk, controller, or bus from becoming a bottleneck and also allows simultaneous I/O operations.

For example, if you have 16 GB of disk storage, you may get better performance from sixteen 1-GB disks rather than four 4-GB disks. More spindles (disks) may allow more simultaneous operations. For random I/O operations, 16 disks may be simultaneously seeking instead of 4 disks. For large sequential data transfers, 16 data streams can be simultaneously working instead of 4 data streams.

RAID 0 (disk striping) enables you to efficiently distribute disk data across the disks in a stripe set. See Section 8.3.3 and Section 8.4 for more information.

If you are using file systems, place the most frequently used file systems on different disks and optimally on different buses. Directories containing executable files or temporary files are often frequently accessed (for example, `/var`, `/usr`, and `/tmp`). If possible, place `/usr` and `/tmp` on different disks.

Guidelines for distributing disk I/O also apply to swap devices. See Section 6.2 for more information about configuring swap devices for high performance.

8.2 Gathering Basic Disk Information

Table 8–1 describes the tools you can use to obtain information about basic disk activity and usage.

Table 8–1: Basic Disk Monitoring Tools

Name	Use	Description
`sys_check`	Analyzes system configuration and displays statistics (Section 4.2)	Creates an HTML file that describes the system configuration, and can be used to diagnose problems. The `sys_check` utility checks kernel variable settings and memory and CPU resources, and provides performance data and lock statistics for SMP systems and kernel profiles. The `sys_check` utility calls other commands and utilities to perform a basic analysis of your configuration and kernel variable settings and provides warnings and tuning recommendations if necessary. See `sys_check`(8) for more information.
`iostat`	Displays disk and CPU usage (Section 8.2.1)	Displays transfer statistics for each disk, and the percentage of time the system has spent in user mode, in user mode running low priority (`nice`) processes, in system mode, and in idle mode.
`(dbx) print nchstats`	Reports namei cache statistics (Section 9.1.2)	Reports namei cache statistics, including hit rates.

Table 8–1: Basic Disk Monitoring Tools (cont.)

Name	Use	Description
`(dbx) print xpt_qhead`, `ccmn_bp_head`, and `xpt_cb_queue`	Reports Common Access Method (CAM) statistics (Section 8.2.2)	Reports CAM statistics, including information about buffers and completed I/O operations.
`diskx`	Tests disk driver functionality	Reads and writes data to disk partitions. The `diskx` exerciser analyzes data transfer performance, verifies the `disktab` database file entry, and tests reads, writes, and seeks. The `diskx` exerciser can destroy the contents of a partition. See `diskx`(8) for more information.

8.2.1 Displaying Disk Usage with the iostat Command

The `iostat` command reports I/O statistics for terminals, disks, and the CPU that you can use to diagnose disk I/O performance problems. The first line of the output is the average since boot time, and each subsequent report is for the last interval. You can also specify a disk name in the command line to output information only about that disk.

An example of the `iostat` command is as follows; output is provided in one-second intervals:

```
# /usr/ucb/iostat 1
      tty     fd0      rz0      rz1      dk3     cpu
 tin tout bps tps  bps tps  bps tps  bps tps  us ni sy id
   1   73   0   0   23   2   37   3    0   0   5  0 17 79
   0   58   0   0   47   5  204  25    0   0   8  0 14 77
   0   58   0   0    8   1   62   1    0   0  27  0 27 46
   0   58   0   0    8   1    0   0    0   0  22  0 31 46
```

The `iostat` command output displays the following information:

- For each disk, (`rz`*n*), the number of kilobytes transferred per second (`bps`) and the number of transfers per second (`tps`).
- For the system (`cpu`), the percentage of time the CPU has spent in user state running processes either at their default priority or preferred priority (`us`), in user mode running processes at a less favored priority (`ni`), in system mode (`sy`), and in idle mode (`id`). This information enables you to determine how disk I/O is affecting the CPU. User mode includes the time the CPU spent executing library routines. System mode includes the time the CPU spent executing system calls.

The `iostat` command can help you to do the following:

- Determine which disk is being used the most and which is being used the least. This information will help you determine how to distribute your file systems and swap space. Use the `swapon -s` command to determine which disks are used for swap space.
- If the `iostat` command output shows a lot of disk activity and a high system idle time, the system may be disk bound. You may need to balance the disk I/O load, defragment disks, or upgrade your hardware.
- If a disk is doing a large number of transfers (the `tps` field) but reading and writing only small amounts of data (the `bps` field), examine how your applications are doing disk I/O. The application may be performing a large number of I/O operations to handle only a small amount of data. You may want to rewrite the application if this behavior is not necessary.

8.2.2 Monitoring CAM by Using the dbx Debugger

The operating system uses the Common Access Method (CAM) as the operating system interface to the hardware. CAM maintains the following `dbx` data structures:

- `xpt_qhead`—Contains information about the current size of the buffer pool free list (`xpt_nfree`), the current number of processes waiting for buffers (`xpt_wait_cnt`), and the total number of times that processes had to wait for free buffers (`xpt_times_wait`).

  For example:

  ```
  # /usr/ucb/dbx –k /vmunix /dev/mem
  (dbx) print xpt_qhead
  struct {
      xws = struct {
          x_flink = 0xffffffff81f07400
          x_blink = 0xffffffff81f03000
          xpt_flags = 2147483656
          xpt_ccb = (nil)
          xpt_nfree = 300
          xpt_nbusy = 0
      }
      xpt_wait_cnt = 0
      xpt_times_wait = 2
      xpt_ccb_limit = 1048576
      xpt_ccbs_total = 300
      x_lk_qhead = struct {
          sl_data = 0
          sl_info = 0
          sl_cpuid = 0
          sl_lifms = 0
      }
  ```

```
}
(dbx)
```

If the value for the xpt_wait_cnt field is not zero, CAM has run out of buffer pool space. If this situation persists, you may be able to eliminate the problem by changing one or more of CAM's I/O attributes (see Section 8.5).

- ccmn_bp_head—Provides statistics on the buffer structure pool. This pool is used for raw I/O to disk. The information provided is the current size of the buffer structure pool (num_bp) and the wait count for buffers (bp_wait_cnt).

  For example:

  ```
  # /usr/ucb/dbx -k /vmunix /dev/mem
  (dbx) print ccmn_bp_head
  struct {
      num_bp = 50
      bp_list = 0xffffffff81f1be00
      bp_wait_cnt = 0
  }
  (dbx)
  ```

  If the value for the bp_wait_cnt field is not zero, CAM has run out of buffer pool space. If this situation persists, you may be able to eliminate the problem by changing one or more of the CAM subsystem attributes (see Section 8.5).

- xpt_cb_queue—Contains the actual link list of the I/O operations that have been completed and are waiting to be passed back to the peripheral drivers (cam_disk or cam_tape).

  For example:

  ```
  # /usr/ucb/dbx -k /vmunix /dev/mem
  (dbx) print xpt_cb_queue
  struct {
      flink = 0xfffffc00004d6828
      blink = 0xfffffc00004d6828
      flags = 0
      initialized = 1
      count = 0
      cplt_lock = struct {
          sl_data = 0
          sl_info = 0
          sl_cpuid = 0
          sl_lifms = 0
      }
  }
  (dbx)
  ```

The `count` field specifies the number of I/O operations that have been completed and are ready to be passed back to a peripheral device driver. Normally, this value is 0 or 1. If the value of `count` is temporarily greater than 1, it may indicate that a large number of I/O operations are completing simultaneously. If the value is consistently greater than 1, it may indicate a problem.

8.3 Managing Logical Storage Manager Performance

The Logical Storage Manager (LSM) can improve system performance and provide high data availability with little additional overhead. LSM also provides you with online storage management features and enhanced performance information and statistics. Although any type of system can benefit from LSM, it is especially suited for configurations with large numbers of disks or configurations that regularly add storage.

LSM allows you to set up a shared storage pool that consists of multiple disks. You can create virtual disks (LSM volumes) from this pool of storage, according to your performance and capacity needs. LSM volumes are used in the same way as disk partitions. You can create UFS file systems and AdvFS file domains and filesets on an LSM volume, use a volume as a raw device, or create LSM volumes on top of RAID storage sets. You can also use LSM on swap disks.

LSM provides you with flexible and easy management for large storage configurations. Because there is no direct correlation between a virtual disk and a physical disk, file system or raw I/O can span disks, as needed. In addition, you can easily add disks to and remove disks from the pool, balance the load, and perform other storage management tasks.

LSM provides more cost-effective RAID functionality than a hardware RAID subsystem. When LSM is used to stripe or mirror disks, it is sometimes referred to as **software RAID**. LSM configurations are less complex than hardware RAID.

LSM supports the following basic disk management features:

- Pool of storage
- Load balancing by transparently moving data across disks
- Disk concatenation (creating a large volume from multiple disks)
- Detailed LSM performance information from the `volstat` command

The following advanced LSM disk management features require a license:

- RAID 0 (disk striping)
- RAID 1 (disk mirroring)

- Block-change logging (BCL), which improves the mirrored volume recovery rate
- Striped swap disks
- Graphical user interface (GUI) for easy disk management and detailed performance information

To obtain the best LSM performance, you must follow the configuration and tuning guidelines. The following sections contain information about:

- Guidelines for disks, disk groups, and databases (Section 8.3.1)
- Guidelines for mirrored disks (Section 8.3.2)
- Guidelines for striped disks (Section 8.3.3)
- Monitoring the LSM configuration and performance (Section 8.3.4)
- Improving LSM performance (Section 8.3.5)

See the *Logical Storage Manager* manual for detailed information about using LSM.

8.3.1 Basic LSM Configuration Recommendations

There are general recommendations that you can use to configure LSM disks, disk groups, and databases for high performance. Each LSM disk group maintains a configuration database, which includes detailed information about mirrored and striped disks and volume, plex, and subdisk records. How you configure your LSM disks, disk groups, and databases determines the flexibility and performance of your LSM configuration.

Table 8–2 describes the LSM disk, disk group, and database configuration recommendations and lists performance benefits as well as tradeoffs.

Table 8–2: LSM Disk, Disk Group, and Database Configuration Guidelines

Recommendation	Benefit	Tradeoff
Initialize your LSM disks as sliced disks (Section 8.3.1.1)	Provides greater storage configuration flexibility	None
Make the `rootdg` disk group a sufficient size (Section 8.3.1.2)	Ensures sufficient space for disk group information	None
Use a sufficient private region size for each disk in a disk group (Section 8.3.1.3)	Ensures sufficient space for database copies	Large private regions require more disk space
Make the private regions in a disk group the same size (Section 8.3.1.4)	Efficiently utilizes the configuration space	None

Table 8–2: LSM Disk, Disk Group, and Database Configuration Guidelines (cont.)

Recommendation	Benefit	Tradeoff
Organize disks into different disk groups according to function (Section 8.3.1.5)	Allows you to move disk groups between systems	Reduces flexibility when configuring volumes
Use an appropriate size and number of database and log copies (Section 8.3.1.6)	Ensures database availability and improves performance	None
Place disks containing database and log copies on different buses (Section 8.3.1.7)	Improves availability	Cost of additional hardware

The following sections describe the previous recommendations in detail.

8.3.1.1 Initializing LSM Disks as Sliced Disks

Initialize your LSM disks as sliced disks, instead of as simple disks. A sliced disk provides greater storage configuration flexibility because the entire disk is under LSM control. The disk label for a sliced disk contains information that identifies the partitions containing the private and the public regions. In contrast, simple disks have both public and private regions in the same partition.

8.3.1.2 Sizing the rootdg Disk Group

You must make sure that the `rootdg` disk group has an adequate size, because the disk group's configuration database contains records for disks outside of the `rootdg` disk group, in addition to the ordinary disk-group configuration information. For example, the `rootdg` configuration database includes disk-access records that define all disks under LSM control.

The `rootdg` disk group must be large enough to contain records for the disks in all the disk groups. See Table 8–3 for more information.

8.3.1.3 Sizing Private Regions

You must make sure that the private region for each disk has an adequate size. LSM keeps disk media label and configuration database copies in each disk's private region.

A private region must be large enough to accommodate the size of the LSM database copies. In addition, the maximum number of LSM objects (disks, subdisks, volumes, and plexes) in a disk group depends on an adequate private region size. However, a large private region requires more disk space. The default private region size is 1024 blocks, which is usually adequate for configurations using up to 128 disks per disk group.

8.3.1.4 Making Private Regions in a Disk Group the Same Size

The private region of each disk in a disk group should be the same size to efficiently utilize the configuration space. One or two LSM configuration database copies can be stored in a disk's private region.

When you add a new disk to existing an LSM disk group, the size of the private region on the new disk is determined by the private region size of the other disks in the disk group. As you add more disks to a disk group, the `voldiskadd` utility reduces the number of configuration copies and log copies that are initialized for the new disks. See `voldiskadd`(8) for more information.

8.3.1.5 Organizing Disks in Disk Groups

You may want to organize disks in disk groups according to their function. This enables disk groups to be moved between systems, and decreases the size of the LSM configuration database for each disk group. However, using multiple disk groups reduces flexibility when configuring volumes.

8.3.1.6 Choosing the Correct Number and Size of the Database and Log Copies

Each disk group maintains a configuration database, which includes detailed information about mirrored and striped disks and volume, plex, and subdisk records. The LSM subsystem's overhead primarily involves managing the kernel change logs and copies of the configuration databases.

LSM performance is affected by the size and the number of copies of the configuration database and the kernel change log. They determine the amount of time it takes for LSM to start up, for changes to the configuration to occur, and for the LSM disks to fail over in a cluster.

Usually, each disk in a disk group contains one or two copies of both the kernel change log and the configuration database. Disk groups consisting of more than eight disks should not have copies on all disks. Always use four to eight copies.

The number of kernel change log copies must be the same as the number of configuration database copies. For the best performance, the number of copies must be the same on each disk that contains copies.

Table 8–3 describes the guidelines for configuration database and kernel change log copies.

Table 8–3: LSM Database and Kernel Change Log Guidelines

Disks Per Disk Group	Size of Private Region (in Blocks)	Configuration and Kernel Change Log Copies Per Disk
1 to 3	512	Two copies in each private region
4 to 8	512	One copy in each private region
9 to 32	512	One copy on four to eight disks, zero copies on remaining disks
33 to 128	1024	One copy on four to eight disks, zero copies on remaining disks
129 to 256	1536	One copy on four to eight disks, zero copies on remaining disks
257 or more	2048	One copy on four to eight disks, zero copies on remaining disks

8.3.1.7 Distributing the Database and Logs Across Different Buses

For disk groups with large numbers of disks, place the disks that contain configuration database and kernel change log copies on different buses. This provides you with better performance and higher availability.

8.3.2 LSM Mirrored Volume Configuration Recommendations

Use LSM mirrored volumes (RAID 1) for high data availability. A plex is a set of data. To mirror a volume, you must set up at least two plexes. If a physical disk fails, the plex containing the failed disk becomes temporarily unavailable, but the remaining plexes are still available.

Mirroring can also improve read performance. However, a write to a mirrored volume results in parallel writes to each plex, so mirroring will degrade disk write performance. Environments whose disk I/O operations are predominantly reads obtain the best performance results from mirroring. See Table 8–4 for mirrored volume guidelines.

Use block-change logging (BCL) to improve the mirrored volume recovery rate when a system failure occurs by reducing the synchronization time. If BCL is enabled and a write is made to a mirrored plex, BCL identifies the block numbers that have changed and then stores the numbers on a logging subdisk. BCL is not used for reads.

BCL is enabled if two or more plexes in a mirrored volume have a logging subdisk associated with them. Only one logging subdisk can be associated with a plex.

BCL can add some overhead to your system and degrade the mirrored volume's write performance. However, the impact is less for systems under a heavy I/O load, because multiple writes to the log are batched into a single write. See Table 8–5 for BCL configuration guidelines.

Note

BCL will be replaced by dirty region logging (DRL) in a future release.

You may want to combine mirroring (RAID 1) with striping (RAID 0) to provide high availability and balance the disk I/O load. See Section 8.3.3 for striping guidelines.

Table 8–4 describes LSM mirrored volume configuration recommendations and lists performance benefits as well as tradeoffs.

Table 8–4: LSM Mirrored Volume Guidelines

Recommendation	Benefit	Tradeoff
Map mirrored plexes across different buses (Section 8.3.2.1)	Improves performance and increases availability	None
Use the appropriate read policy (Section 8.3.2.2)	Efficiently distributes reads	None
Attach up to eight plexes to the same volume (Section 8.3.2.3)	Improves performance for read-intensive workloads and increases availability	Uses disk space inefficiently
Use a symmetrical configuration (Section 8.3.2.4)	Provides more predictable performance	None
Use block-change logging (BCL) (Table 8–5)	Improves mirrored volume recovery rate	May decrease write performance
Stripe the mirrored volumes (Table 8–6)	Improves disk I/O performance and balances I/O load	Increases management complexity

Table 8–5 describes LSM block-change logging (BCL) configuration recommendations and lists performance benefits as well as tradeoffs.

Table 8–5: LSM Block-Change Logging Guidelines

Recommendation	Benefit	Tradeoff
Configure multiple logging subdisks (Section 8.3.2.5)	Improves recovery time	Requires additional disks
Use a write-back cache for logging subdisks (Section 8.3.2.6)	Minimizes BCLs write degradation	Cost of hardware RAID subsystem

Table 8–5: LSM Block-Change Logging Guidelines (cont.)

Recommendation	Benefit	Tradeoff
Use the appropriate BCL subdisk size (Section 8.3.2.7)	Enables migration to dirty region logging	None
Place logging subdisks on infrequently used disks (Section 8.3.2.8)	Helps to prevent disk bottlenecks	None
Use solid-state disks for logging subdisks (Section 8.3.2.9)	Minimizes BCL's write degradation	Cost of solid-state disks

The following sections describe the previous LSM mirrored volume and BCL recommendations in detail.

8.3.2.1 Mirroring Volumes Across Different Buses

Putting each mirrored plex on a different bus improves performance by enabling simultaneous I/O operations. Mirroring across different buses also increases availability by protecting against bus and adapter failure.

8.3.2.2 Choosing a Read Policy for a Mirrored Volume

To provide optimal performance for different types of mirrored volumes, LSM supports the following read policies:

- Round-robin read

  Satisfies read operations to the volume in a round-robin manner from all plexes in the volume.

- Preferred read

  Satisfies read operations from one specific plex (usually the plex with the highest performance).

- Select

  Selects a default read policy based on the plex associations to the volume. If the mirrored volume contains a single, enabled, striped plex, the default is to prefer that plex. For any other set of plex associations, the default is to use a round-robin policy.

If one plex exhibits superior performance, either because the plex is striped across multiple disks or because it is located on a much faster device, then set the read policy to preferred read for that plex. By default, a mirrored volume with one striped plex should have the striped plex configured as the preferred read. Otherwise, you should use the round-robin read policy.

8.3.2.3 Using Multiple Plexes in a Mirrored Volume

To improve performance for read-intensive workloads, up to eight plexes can be attached to the same mirrored volume. However, this configuration does not use disk space efficiently.

8.3.2.4 Using a Symmetrical Configuration

A symmetrical mirrored disk configuration provides predictable performance and easy management. Use the same number of disks in each mirrored plex. For mirrored striped volumes, you can stripe across half of the available disks to form one plex and across the other half to form the other plex.

8.3.2.5 Using Multiple BCL Subdisks

Using multiple block-change logging (BCL) subdisks will improve recovery time after a failure.

8.3.2.6 Using a Write-Back Cache with LSM

To minimize BCL's impact on write performance, use LSM in conjunction with a RAID subsystem that has a write-back cache. Typically, the BCL performance degradation is more significant on systems with few writes than on systems with heavy write loads.

8.3.2.7 Sizing BCL Subdisks

To support migration from BCL to dirty region logging (DRL), which will be supported in a future release, use the appropriate BCL subdisk size.

If you have less than 64 GB of disk space under LSM control, calculate the subdisk size by multiplying 1 block by each gigabyte of storage. If the result is an odd number, add 1 block; if the result is an even number, add 2 blocks.

For example, if you have 1 GB (or less) of space, use a 2-block subdisk. If you have 2 GB (or 3 GB) of space, use a 4-block subdisk.

If you have more than 64 GB of disk space under LSM control, use a 64-block subdisk.

8.3.2.8 Placing BCL Logging Subdisks on Infrequently Used Disks

Place logging subdisks on infrequently used disks. Because these subdisks are frequently written, do not put them on busy disks. In addition, do not configure BCL subdisks on the same disks as the volume data, because this will cause head seeking or thrashing.

8.3.2.9 Using Solid-State Disks for BCL Subdisks

If persistent (nonvolatile) solid-state disks are available, use them for logging subdisks.

8.3.3 LSM Striped Volume Configuration Recommendations

Striping volumes (RAID 0) with LSM enables parallel I/O streams to operate concurrently on separate devices, which distributes the disk I/O load and improves performance. Striping is especially effective for applications that perform large sequential data transfers or multiple, simultaneous I/O operations.

Striping distributes data in fixed-size portions (stripes) across the disks in a volume. The stripes are interleaved across the striped plex's subdisks, which are located on different disks to evenly distribute the disk I/O.

The performance benefit of striping depends on the stripe width, which is the number of blocks in a stripe, and how your users and applications perform I/O. Bandwidth increases with the number of disks across which a plex is striped.

You can combine mirroring (RAID 1) with striping to obtain high availability. However, mirroring will decrease write performance. See Section 8.3.2 for mirroring guidelines.

Table 8–6 describes the LSM striped volume configuration recommendations and lists performance benefits as well as tradeoffs.

Table 8–6: LSM Striped Volume Guidelines

Recommendation	Benefit	Tradeoff
Use multiple disks in a striped volume (Section 8.3.3.1)	Improves performance	Decreases volume reliability
Distribute subdisks across different disks and buses (Section 8.3.3.2)	Improves performance and increases availability	None
Use the appropriate stripe width (Section 8.3.3.3)	Improves performance	None

Table 8–6: LSM Striped Volume Guidelines (cont.)

Recommendation	Benefit	Tradeoff
Avoid splitting small data transfers (Section 8.3.3.3)	Improves the performance of volumes that quickly receive multiple data transfers	May use disk space inefficiently
Split large individual data transfers (Section 8.3.3.3)	Improves the performance of volumes that receive large data transfers	Decreases throughput

The following sections describe the previous LSM striped volume configuration recommendations in detail.

8.3.3.1 Increasing the Number of Disks in a Striped Volume

Increasing the number of disks in a striped volume can increase the bandwidth, depending on the applications and file systems you are using and on the number of simultaneous users. However, this reduces the effective mean-time-between-failures (MTBF) of the volume. If this reduction is a problem, use both mirroring and striping. See Section 8.3.2 for mirroring guidelines.

8.3.3.2 Distributing Striped Volume Subdisks Across Different Buses

Distribute the subdisks of a striped volume across different buses. This improves performance and helps to prevent a single bus from becoming a bottleneck.

8.3.3.3 Choosing the Correct Stripe Width

The performance benefit of striping depends on the size of the stripe width and the characteristics of the I/O load. Stripes of data are allocated alternately and evenly to the subdisks of a striped plex. A striped plex consists of a number of equal-sized subdisks located on different disks.

The number of blocks in a stripe determines the stripe width. LSM uses a default stripe width of 64 KB (or 128 sectors), which works well in most environments.

Use the `volstat` command to determine the number of data transfer splits. For volumes that receive only small I/O transfers, you may not want to use striping because disk access time is important. Striping is most beneficial for large data transfers.

To improve performance of large sequential data transfers, use a stripe width that will divide each individual data transfer and distribute the blocks equally across the disks.

To improve the performance of multiple simultaneous small data transfers, make the stripe width the same size as the data transfer. However, an excessively small stripe width can result in poor system performance.

If you are striping mirrored volumes, ensure that the stripe width is the same for each plex.

8.3.4 Gathering LSM Information

Table 8–7 describes the tools you can use to obtain information about the Logical Storage Manager (LSM).

Table 8–7: LSM Monitoring Tools

Name	Use	Description
`volprint`	Displays LSM configuration information (Section 8.3.4.1)	Displays information about LSM disk groups, disk media, volumes, plexes, and subdisk records. It does not display disk access records. See `volprint`(8) for more information.
`volstat`	Monitors LSM performance statistics (Section 8.3.4.2)	Displays performance statistics since boot time for all LSM objects (volumes, plexes, subdisks, and disks). These statistics include information about read and write operations, including the total number of operations, the number of failed operations, the number of blocks read or written, and the average time spent on the operation in a specified interval of time. The `volstat` utility also can reset the I/O statistics. See `volstat`(8) for more information.
`voltrace`	Tracks LSM operations (Section 8.3.4.3)	Sets I/O tracing masks against one or all volumes in the LSM configuration and logs the results to the LSM default event log, `/dev/volevent`. The utility also formats and displays the tracing mask information and can trace the following ongoing LSM events: requests to logical volumes, requests that LSM passes to the underlying block device drivers, and I/O events, errors, and recoveries. See `voltrace`(8) for more information.

Table 8–7: LSM Monitoring Tools (cont.)

Name	Use	Description
`volwatch`	Monitors LSM events (Section 8.3.4.4)	Monitors LSM for failures in disks, volumes, and plexes, and sends mail if a failure occurs. The `volwatch` script starts automatically when you install LSM. See `volwatch`(8) for more information.
`dxlsm`	Monitors LSM objects (Section 8.3.4.5)	Using the Analyze menu, displays information about LSM disks, volumes, and subdisks. See `dxlsm`(8) for more information.

The following sections describe some of these commands in detail.

8.3.4.1 Displaying LSM Configuration Information by Using the volprint Utility

The `volprint` utility displays information from records in the LSM configuration database. You can select the records to be displayed by name or by using special search expressions. In addition, you can display record association hierarchies, so that the structure of records is more apparent.

Use the `volprint` utility to display disk group, disk media, volume, plex, and subdisk records. Invoke the `voldisk list` command to display disk access records or physical disk information.

The following example uses the `volprint` utility to show the status of the `voldev1` volume:

```
# /usr/sbin/volprint -ht voldev1
DG NAME        GROUP-ID
DM NAME        DEVICE       TYPE     PRIVLEN  PUBLEN   PUBPATH
V  NAME        USETYPE      KSTATE   STATE    LENGTH   READPOL  PREFPLEX
PL NAME        VOLUME       KSTATE   STATE    LENGTH   LAYOUT   ST-WIDTH MODE
SD NAME        PLEX         PLOFFS   DISKOFFS LENGTH   DISK-NAME    DEVICE

v  voldev1     fsgen        ENABLED  ACTIVE   804512   SELECT   -
pl voldev1-01  voldev1      ENABLED  ACTIVE   804512   CONCAT   -        RW
sd rz8-01      voldev1-01   0        0        804512   rz8          rz8
pl voldev1-02  voldev1      ENABLED  ACTIVE   804512   CONCAT   -        RW
sd dev1-01     voldev1-02   0        2295277  402256   dev1         rz9
sd rz15-02     voldev1-02   402256   2295277  402256   rz15         rz15
```

See `volprint`(8) for more information.

8.3.4.2 Monitoring LSM Performance Statistics by Using the volstat Utility

The `volstat` utility provides information about activity on volumes, plexes, subdisks, and disks under LSM control. It reports statistics that reflect the activity levels of LSM objects since boot time.

The amount of information displayed depends on which options you specify to the `volstat` utility. For example, you can display statistics for a specific

LSM object, or you can display statistics for all objects at one time. If you specify a disk group, only statistics for objects in that disk group are displayed. If you do not specify a particular disk group, the `volstat` utility displays statistics for the default disk group (`rootdg`).

You can also use the `volstat` utility to reset the base statistics to zero. This can be done for all objects or for only specified objects. Resetting the statistics to zero before a particular operation makes it possible to measure the subsequent impact of that operation.

The following example uses the `volstat` utility to display statistics on LSM volumes:

```
# /usr/sbin/volstat
OPERATIONS       BLOCKS        AVG TIME(ms)
TYP NAME        READ   WRITE    READ    WRITE   READ   WRITE
vol archive      865     807    5722     3809   32.5    24.0
vol home        2980    5287    6504    10550   37.7   221.1
vol local      49477   49230  507892   204975   28.5    33.5
vol src        79174   23603  425472   139302   22.4    30.9
vol swapvol    22751   32364  182001   258905   25.3   323.2
```

See `volstat`(8) for more information.

8.3.4.3 Tracking LSM Operations by Using the voltrace Utility

The `voltrace` utility reads an event log (`/dev/volevent`) and prints formatted event log records to standard output. Using the `voltrace` utility, you can set event trace masks to determine the type of events to track. For example, you can trace I/O events, configuration changes, or I/O errors.

The following example uses the `voltrace` utility to display status on all new events:

```
# /usr/sbin/voltrace -n -e all
18446744072623507277 IOTRACE 439: req 3987131 v:rootvol p:rootvol-01 \
  d:root_domain s:rz3-02 iot write lb 0 b 63120 len 8192 tm 12
18446744072623507277 IOTRACE 440: req 3987131 \
  v:rootvol iot write lb 0 b 63136 len 8192 tm 12
```

See `voltrace`(8) for more information.

8.3.4.4 Monitoring LSM Events by Using the volwatch Script

The `volwatch` shell script is automatically started when you install LSM. This script sends mail to root if certain LSM configuration events occur, such as a plex detach caused by a disk failure. The script sends mail to root by default. You also can specify another mail recipient.

See `volwatch`(8) for more information.

8.3.4.5 Monitoring LSM by Using the dxlsm Graphical User Interface

The LSM Visual Administrator, the `dxlsm` graphical user interface (GUI), allows you to graphically manipulate LSM objects and manage the LSM configuration. The `dxlsm` GUI also includes an Analyze menu that allows you to display statistics about volumes, LSM disks, and subdisks. The information is graphically displayed, using colors and patterns on the disk icons, and numerically, using the `Analysis Statistics` form.

You can use the `Analysis Parameters` form to customize the displayed information.

See the *Logical Storage Manager* manual and `dxlsm`(8X) for more information.

8.3.5 Improving LSM Performance

You may be able to improve LSM performance by modifying an LSM subsystem attribute or by performing some administrative tasks. Be sure you have followed the configuration guidelines that are described in Section 8.3.1, Section 8.3.2, and Section 8.3.3.

You can improve LSM performance as follows:

- Increase the maximum number of LSM volumes

  For large systems, increase the value of the `lsm` subsystem attribute `max-vol`, which specifies the maximum number of volumes per system. The default is 1024; you can increase it to 4096. See Section 4.4 for information about modifying kernel subsystem attributes.

- Balance the I/O load

  LSM allows you to achieve a fine level of granularity in data placement, because LSM provides a way for volumes to be distributed across multiple disks. After measuring actual data-access patterns, you can adjust the placement of file systems.

  You can reassign data to specific disks to balance the I/O load among the available storage devices. You can reconfigure volumes on line after performance patterns have been established without adversely impacting volume availability.

- Stripe frequently accessed data

  If you have frequently accessed file systems or databases, you can realize significant performance benefits by striping the data across multiple disks, which increases bandwidth to this data. See Section 8.3.3 for information.

- Set the preferred read policy to the fastest mirrored plex

  If one plex of a mirrored volume exhibits superior performance, either because the disk is being striped or concatenated across multiple disks, or because it is located on a much faster device, then set the read policy to the preferred read policy for that plex. By default, a mirrored volume with one striped plex should be configured with the striped plex as the preferred read. See Section 8.3.2.2 for more information.

- You may be able to improve the performance of LSM systems that use large amounts of memory or disks by increasing the value of the `volinfo.max_io` kernel variable. See Section 4.4.6 for information about using `dbx` to display and modify kernel variables.

8.4 Managing Hardware RAID Subsystem Performance

Hardware RAID subsystems provide RAID functionality for high performance and high availability, relieve the CPU of disk I/O overhead, and enable you to connect many disks to a single I/O bus. There are various types of hardware RAID subsystems with different performance and availability features, but they all include a RAID controller, disks in enclosures, cabling, and disk management software.

RAID storage solutions range from low-cost backplane RAID array controllers to cluster-capable RAID array controllers that provide extensive performance and availability features, such as write-back caches and complete component redundancy.

Hardware RAID subsystems use disk management software, such as the RAID Configuration Utility (RCU) and the StorageWorks Command Console (SWCC) utility, to manage the RAID devices. Menu-driven interfaces allow you to select RAID levels.

Use hardware RAID to combine multiple disks into a single storage set that the system sees as a single unit. A storage set can consist of a simple set of disks, a striped set, a mirrored set, or a RAID set. You can create LSM volumes, AdvFS file domains, or UFS file systems on a storage set, or you can use the storage set as a raw device.

All hardware RAID subsystems provide you with the following features:

- A RAID controller that relieves the CPU of the disk I/O overhead
- Increased disk storage capacity

  Hardware RAID subsystems allow you to connect a large number of disks to a single I/O bus. In a typical storage configuration, you use a SCSI bus connected to an I/O bus slot to attach disks to a system. However, systems have limited I/O bus slots, and you can connect only a limited number of disks to a SCSI bus (eight for SCSI-2 and sixteen for SCSI-3).

Hardware RAID subsystems contain multiple internal SCSI buses and host bus adapters, and require only one I/O bus to connect the subsystem to a system.

- Read cache

  A read cache improves I/O read performance by holding data that it anticipates the host will request. If a system requests data that is already in the read cache (a cache hit), the data is immediately supplied without having to read the data from disk. Subsequent data modifications are written both to disk and to the read cache (write-through caching).

- Write-back cache

  Hardware RAID subsystems support (as a standard or an optional feature) a write-back cache, which can improve I/O write performance while maintaining data integrity. A write-back cache decreases the latency of many small writes, and can improve Internet server performance because writes appear to be executed immediately. Applications that perform few writes will not benefit from a write-back cache.

  With write-back caching, data intended to be written to disk is temporarily stored in the cache, consolidated, and then periodically written (flushed) to disk for maximum efficiency. I/O latency is reduced by consolidating contiguous data blocks from multiple host writes into a single unit.

  A write-back cache must be battery-backed to protect against data loss and corruption.

- RAID support

  All hardware RAID subsystems support RAID 0 (disk striping), RAID 1 (disk mirroring), and RAID 5. High-performance RAID array subsystems also support RAID 3 and dynamic parity RAID. See Section 1.3.3.1 for information about RAID levels.

- Non-RAID disk array capability or "just a bunch of disks" (JBOD)

- Component hot swapping and hot sparing

  Hot swap support allows you to replace a failed component while the system continues to operate. Hot spare support allows you to automatically use previously installed components if a failure occurs.

- Graphical user interface (GUI) for easy management and monitoring

There are various types of hardware RAID subsystems, which provide different degrees of performance and availability at various costs:

- Backplane RAID array storage subsystems

  These entry-level subsystems, such as those utilizing the RAID Array 230/Plus storage controller, provide a low-cost hardware RAID solution and are designed for small and midsize departments and workgroups.

  A backplane RAID array storage controller is installed in an I/O bus slot, either a PCI bus slot or an EISA bus slot, and acts as both a host bus adapter and a RAID controller.

  Backplane RAID array subsystems provide RAID functionality (0, 1, 0+1, and 5), an optional write-back cache, and hot swap functionality.

- High-performance RAID array subsystems

  These subsystems, such as the RAID Array 450 subsystem, provide extensive performance and availability features and are designed for client/server, data center, and medium to large departmental environments.

  A high-performance RAID array controller, such as an HSZ50 controller, is connected to a system through a FWD SCSI bus and a high-performance host bus adapter installed in an I/O bus slot.

  High-performance RAID array subsystems provide RAID functionality (0, 1, 0+1, 3, 5, and dynamic parity RAID), dual-redundant controller support, scalability, storage set partitioning, multipath concurrent access, a standard battery-backed write-back cache, and hot-swappable components.

- Enterprise Storage Arrays (ESA)

  These preconfigured high-performance hardware RAID subsystems, such as the RAID Array 7000, provide the highest performance, availability, and disk capacity of any RAID subsystem. They are used for high transaction-oriented applications and high bandwidth decision-support applications.

  ESAs support all major RAID levels, including dynamic parity RAID; fully redundant and hot-swappable components; a standard battery-backed write-back cache; and centralized storage management.

See the Compaq *Systems & Options Catalog* for detailed information about hardware RAID subsystem features.

Table 8–8 describes the hardware RAID subsystem configuration recommendations and lists performance benefits as well as tradeoffs.

Table 8–8: Hardware RAID Subsystem Configuration Guidelines

Recommendation	Performance Benefit	Tradeoff
Evenly distribute disks in a storage set across different buses (Section 8.4.1)	Improves performance and helps to prevent bottlenecks	None
Configure devices for multipath concurrent access	Improves throughput and increases availability	Cost of devices that support this feature
Use disks with the same data capacity in each storage set (Section 8.4.2)	Simplifies storage management	None
Use an appropriate stripe size (Section 8.4.3)	Improves performance	None
Mirror striped sets (Section 8.4.4)	Provides availability and distributes disk I/O performance	Increases configuration complexity and may decrease write performance
Use a write-back cache (Section 8.4.5)	Improves write performance	Cost of hardware
Use dual-redundant RAID controllers (Section 8.4.6)	Improves performance, increases availability, and prevents I/O bus bottlenecks	Cost of hardware
Install spare disks (Section 8.4.7)	Improves availability	Cost of disks
Replace failed disks promptly (Section 8.4.7)	Improves performance	None

The following sections describe some of these guidelines. See your RAID subsystem documentation for detailed configuration information.

8.4.1 Distributing Storage Set Disks Across Buses

You can improve performance and help to prevent bottlenecks by distributing storage set disks evenly across different buses.

In addition, make sure that the first member of each mirrored set is on a different bus.

8.4.2 Using Disks with the Same Data Capacity

Use disks with the same capacity in the same storage set. This simplifies storage management.

8.4.3 Choosing the Correct Stripe Size

You must understand how your applications perform disk I/O before you can choose the stripe (chunk) size that will provide the best performance benefit. See Section 2.1 for information about determining a resource model for your system.

If the stripe size is large compared to the average I/O size, each disk in a stripe set can respond to a separate data transfer. I/O operations can then be handled in parallel, which increases sequential write performance and throughput. This can improve performance for environments that perform large numbers of I/O operations, including transaction processing, office automation, and file services environments, and for environments that perform multiple random read and write operations.

If the stripe size is smaller than the average I/O operation, multiple disks can simultaneously handle a single I/O operation, which can increase bandwidth and improve sequential file processing. This is beneficial for image processing and data collection environments. However, do not make the stripe size so small that it will degrade performance for large sequential data transfers.

For example, if you use an 8-KB stripe size, small data transfers will be distributed evenly across the member disks, but a 64-KB data transfer will be divided into at least eight data transfers.

If your applications are doing I/O to a raw device and not a file system, use a stripe size that distributes a single data transfer evenly across the member disks. For example, if the typical I/O size is 1 MB and you have a four-disk array, you could use a 256-KB stripe size. This would distribute the data evenly among the four member disks, with each doing a single 256-KB data transfer in parallel.

For small file system I/O operations, use a stripe size that is a multiple of the typical I/O size (for example, four to five times the I/O size). This will help to ensure that the I/O is not split across disks.

You may want to choose a stripe size that will prevent any particular range of blocks from becoming a bottleneck. For example, if an application often uses a particular 8-KB block, you may want to use a stripe size that is slightly larger or smaller than 8 KB or is a multiple of 8 KB to force the data onto a different disk.

8.4.4 Mirroring Striped Sets

Striped disks improve I/O performance by distributing the disk I/O load. Mirroring striped disks provides high availability, but can decrease write performance, because each write operation results in two disk writes.

8.4.5 Using a Write-Back Cache

RAID subsystems support, either as a standard or an optional feature, a nonvolatile (battery-backed) write-back cache that can improve disk I/O performance while maintaining data integrity. A write-back cache improves performance for systems that perform large numbers of writes. Applications that perform few writes will not benefit from a write-back cache.

With write-back caching, data intended to be written to disk is temporarily stored in the cache and then periodically written (flushed) to disk for maximum efficiency. I/O latency is reduced by consolidating contiguous data blocks from multiple host writes into a single unit.

A write-back cache improves performance because writes appear to be executed immediately. If a failure occurs, upon recovery the RAID controller detects any unwritten data that still exists in the write-back cache and writes the data to disk before enabling normal controller operations.

A write-back cache must be battery-backed to protect against data loss and corruption.

If you are using an HSZ40 or HSZ50 RAID controller with a write-back cache, the following guidelines may improve performance:

- Set `CACHE_POLICY` to B.
- Set `CACHE_FLUSH_TIMER` to a minimum of 45 (seconds).
- Enable the write-back cache (`WRITEBACK_CACHE`) for each unit, and set the value of `MAXIMUM_CACHED_TRANSFER_SIZE` to a minimum of 256.

See the RAID subsystem documentation for more information about using the write-back cache.

8.4.6 Using Dual-Redundant Controllers

If supported by your RAID subsystem, you can use a dual-redundant controller configuration and balance the number of disks across the two controllers. This can improve performance, increase availability, and prevent I/O bus bottlenecks.

8.4.7 Using Spare Disks to Replace Failed Disks

Install predesignated spare disks on separate controller ports and storage shelves. This will help you to maintain data availability and recover quickly if a disk failure occurs.

8.5 Tuning CAM

The Common Access Method (CAM) is the operating system interface to the hardware. CAM maintains pools of buffers that are used to perform I/O. Each buffer takes approximately 1 KB of physical memory. Monitor these pools (see Section 8.2.2) and tune them if necessary.

You can modify the following `io` subsystem attributes to improve CAM performance:

- `cam_ccb_pool_size`—The initial size of the buffer pool free list at boot time. The default is 200.
- `cam_ccb_low_water`—The number of buffers in the pool free list at which more buffers are allocated from the kernel. CAM reserves this number of buffers to ensure that the kernel always has enough memory to shut down runaway processes. The default is 100.
- `cam_ccb_increment`—The number of buffers either added or removed from the buffer pool free list. Buffers are allocated on an as-needed basis to handle immediate demands, but are released in a more measured manner to guard against spikes. The default is 50.

If the I/O pattern associated with your system tends to have intermittent bursts of I/O operations (I/O spikes), increasing the values of the `cam_ccb_pool_size` and `cam_ccb_increment` attributes may improve performance.

9 Managing File System Performance

The Tru64 UNIX operating system supports various file system options that have different performance features and functionality.

This chapter describes how to perform the following tasks:

- Gather information about all types of file systems (Section 9.1)
- Apply tuning recommendations that are applicable to all types of file systems (Section 9.2)
- Manage Advanced File System (AdvFS) performance (Section 9.3)
- Manage UNIX File System (UFS) performance (Section 9.4)
- Manage Network File System (NFS) performance (Section 9.5)

9.1 Gathering File System Information

The following sections describe how to use tools to monitor general file system activity and describe some general file system tuning guidelines.

9.1.1 Monitoring the Unified Buffer Cache

The Unified Buffer Cache (UBC) uses a portion of physical memory to cache most-recently accessed UFS file system data for reads and writes and for page faults from mapped file regions, in addition to AdvFS metadata and user data. The UBC competes with processes for this portion of physical memory, so the amount of memory allocated to the UBC can affect overall system performance.

See Section 6.3.5 for information about using `dbx` to check the UBC. See Section 9.2 for information on how to tune the UBC.

9.1.2 Checking the namei Cache with the dbx Debugger

The namei cache is used by UFS, AdvFS, CD-ROM File System (CDFS), and NFS to store recently used file system pathname/inode number pairs. It also stores inode information for files that were referenced but not found. Having this information in the cache substantially reduces the amount of searching that is needed to perform pathname translations.

To check the namei cache, use the `dbx print` command to examine the `nchstats` data structure. Consider the following example:

```
# /usr/ucb/dbx -k /vmunix /dev/mem
(dbx) print nchstats
struct {
    ncs_goodhits = 9748603
    ncs_neghits = 888729
    ncs_badhits = 23470
    ncs_falsehits = 69371
    ncs_miss = 1055430
    ncs_long = 4067
    ncs_pass2 = 127950
    ncs_2passes = 195763
    ncs_dirscan = 47
}
(dbx)
```

Examine the `ncs_goodhits` (found a pair), `ncs_neghits` (found a pair that did not exist), and `ncs_miss` (did not find a pair) fields to determine the hit rate. The hit rate should be above 80 percent (`ncs_goodhits` plus `ncs_neghits` divided by the sum of the `ncs_goodhits`, `ncs_neghits`, `ncs_miss`, and `ncs_falsehits`).

See Section 9.2.1 for information on how to improve the namei cache hit rate and lookup speeds.

9.2 Tuning File Systems

You may be able to improve I/O performance by modifying some kernel attributes that affect all file system performance.

General file system tuning often involves tuning the Virtual File System (VFS), which provides a uniform interface that allows common access to files, regardless of the file system on which the files reside.

To successfully improve file system performance, you must understand how your applications and users perform I/O, as described in Section 2.1. Because file systems share memory with processes, you should also understand virtual memory operation, as described in Chapter 6.

Table 9–1 describes the guidelines for general file system tuning and lists the performance benefits as well as the tradeoffs. There are also specific guidelines for AdvFS and UFS file systems. See Section 9.3 and Section 9.4 for information.

Table 9–1: General File System Tuning Guidelines

Action	Performance Benefit	Tradeoff
Increase the size of the namei cache (Section 9.2.1)	Improves cache lookup operations	Consumes memory
Increase the size of the hash chain table for the namei cache (Section 9.2.2)	Improves cache lookup operations	Consumes memory
Increase the memory allocated to the UBC (Section 9.2.3)	Improves file system performance	May cause excessive paging and swapping
Decrease the amount of memory borrowed by the UBC (Section 9.2.4)	Improves file system performance	Decreases the memory available for processes and may decrease system response time
Increase the minimum size of the UBC (Section 9.2.5)	Improves file system performance	Decreases the memory available for processes
Increase the UBC write device queue depth (Section 9.2.6)	Increases overall file system throughput and frees memory	Decreases interactive response performance
Decrease the UBC write device queue depth (Section 9.2.6)	Improves interactive response time	Consumes memory
Increase the amount of UBC memory used to cache a large file (Section 9.2.7)	Improves large file performance	May allow a large file to consume all the pages on the free list
Decrease the amount of UBC memory used to cache a large file (Section 9.2.7)	Prevents a large file from consuming all the pages on the free list	May degrade large file performance
Disable flushing to disk file read access times (Section 9.2.8)	Improves file system performance for proxy servers	Jeopardizes the integrity of read access time updates and violates POSIX standards
Use Prestoserve to cache only file system metadata (Section 9.2.9)	Improves performance for applications that access large amounts of file system metadata	Prestoserve is not supported in a cluster or for nonfile system I/O operations
Increase the size of the Prestoserve buffer hash table (Section 9.2.10)	Decreases Prestoserve lock contention	Prestoserve is not supported in a cluster or for nonfile system I/O operations

Table 9–1: General File System Tuning Guidelines (cont.)

Action	Performance Benefit	Tradeoff
Cache more vnodes on the free list (Section 9.2.11)	Improves cache lookup operations	Consumes memory
Increase the amount of time for which vnodes are kept on the free list (Section 9.2.12)	Improves cache lookup operations	None
Delay vnode deallocation (Section 9.2.13)	Improves namei cache lookup operations	Consumes memory
Accelerate vnode deallocation (Section 9.2.14)	Speeds the freeing of memory	Reduces the efficiency of the namei cache
Disable vnode deallocation (Section 9.2.15)	Optimizes processing time	Consumes memory

The following sections describe these guidelines in detail.

9.2.1 Increasing the Size of the namei Cache

The namei cache is used by all file systems to map file pathnames to inodes. Monitor the cache by using the `dbx print` command to examine the `nchstats` data structure. The miss rate (misses / (good + negative + misses)) should be less than 20 percent.

To make lookup operations faster, increase the size of the namei cache by increasing the value of the `maxusers` attribute (the recommended way), as described in Section 5.1, or by increasing the value of the `vfs` subsystem attribute `name-cache-size` (the default value is 1029).

Increasing the value of the `maxusers` or `name-cache-size` attribute allocates more system resources for use by the kernel, but also increases the amount of wired memory consumed by the kernel.

Note that many benchmarks perform better with a large namei cache.

9.2.2 Increasing the Size of the Hash Chain Table for the namei Cache

Increasing the size of the hash chain table for the namei cache distributes the namei cache elements and reduces the time needed for linear searches, which can improve lookup speeds. The `vfs` subsystem attribute `name-cache-hash-size` specifies the size of the hash chain table, in table elements, for the namei cache.

The default value of the name-cache-hash-size attribute is the value of the name-cache-size attribute divided by 8 and rounded up to the next power of 2, or 8192, whichever is the highest value.

You can change the value of the name-cache-hash-size attribute so that each hash chain has three or four name cache entries. To determine an appropriate value for the name-cache-hash-size attribute, divide the value of the vfs subsystem attribute name-cache-size by 3 or 4 and then round the result to a power of 2.

For example, if the value of name-cache-size is 1029, dividing 1029 by 4 produces a value of 257. Based on this calculation, you could specify 256 (2 to the power of 8) for the value of the name-cache-hash-size attribute.

9.2.3 Increasing Memory for the UBC

The Unified Buffer Cache (UBC) shares with processes the memory that is not wired. The UBC caches UFS file system data for reads and writes, AdvFS metadata and file data, and Memory File System (MFS) data. Performance is improved if the cached data is later reused and a disk operation is avoided.

If you reuse data, be sure to allocate enough memory to the UBC to improve the chance that data will be found in the cache. An insufficient amount of memory allocated to the UBC can impair file system performance. However, the performance of an application that generates a lot of random I/O will not be improved by a large UBC, because the next access location for random I/O cannot be predetermined.

To increase the maximum amount of memory allocated to the UBC, you can increase the value of the vm subsystem attribute ubc-maxpercent. The default value is 100 percent, which should be appropriate for most configurations, including Internet servers.

Be sure that allocating more memory to the UBC does not cause excessive paging and swapping.

See Section 6.1.2.2 for information about UBC memory allocation.

9.2.4 Decreasing the Amount of Borrowed Memory

The UBC borrows all physical memory above the value of the vm subsystem attribute ubc-borrowpercent and up to the value of the ubc-maxpercent attribute.

Increasing the value of the ubc-borrowpercent attribute allows more memory to remain in the UBC when page reclamation begins. This can increase the UBC cache effectiveness, but may degrade system response time when a low-memory condition occurs. If `vmstat` output shows excessive paging but few or no pageouts, you may want to increase borrowing threshold.

The value of the `ubc-borrowpercent` attribute can range from 0 to 100. The default value is 20 percent.

See Section 6.1.2.2 for information about UBC memory allocation.

9.2.5 Increasing the Minimum Size of the UBC

Increasing the minimum size of the UBC will prevent large programs from completely filling the UBC. For I/O servers, you may want to raise the value of the `vm` subsystem attribute `ubc-minpercent` to ensure that enough memory is available for the UBC. The default value is 10 percent.

Because the UBC and processes share virtual memory, increasing the minimum size of the UBC may cause the system to page excessively. In addition, if the values of the `vm` subsystem attributes `ubc-maxpercent` and `ubc-minpercent` are close together, you may degrade I/O performance.

To ensure that the value of the `ubc-minpercent` is appropriate, use the `vmstat` command to examine the page-out rate. See Section 6.3.2 for information.

See Section 6.1.2.2 for information about UBC memory allocation.

9.2.6 Modifying the UBC Write Device Queue Depth

The UBC uses a buffer to facilitate the movement of data between memory and disk. The `vm` subsystem attribute `vm-ubcbuffers` specifies the maximum file system device I/O queue depth for writes. The default value is 256.

Increasing the UBC write device queue depth frees memory and increases the overall file system throughput.

Decreasing the UBC write device queue depth increases memory demands, but it improves the interactive response time.

9.2.7 Controlling Large File Caching

If a large file completely fills the UBC, it may take all of the pages on the free page list, which may cause the system to page excessively. The `vm` subsystem attribute `vm-ubcseqpercent` specifies the maximum amount of memory allocated to the UBC that can be used to cache a file. The default value is 10 percent of memory allocated to the UBC.

The `vm` subsystem attribute `vm-ubcseqstartpercent` specifies the size of the UBC as a percentage of physical memory, at which time the virtual memory subsystem starts stealing the UBC LRU pages for a file to satisfy the demand for pages. The default is 50 percent of physical memory.

Increasing the value of the `vm-ubcseqpercent` attribute will improve the performance of a large single file, but will decrease the remaining amount of memory.

Decreasing the value of the `vm-ubcseqpercent` attribute will increase the available memory, but will degrade the performance of a large single file.

To force the system to reuse the pages in the UBC instead of taking pages from the free list, perform the following tasks:

- Make the maximum size of the UBC greater than the size of the UBC as a percentage of percentage of memory. That is, the value of the `vm` subsystem attribute `ubc-maxpercent` (the default is 100 percent) must be greater than the value of the `vm-ubcseqstartpercent` attribute (the default is 50 percent).
- Make the value of the `vm-ubcseqpercent` attribute, which specifies the size of a file as a percentage of the UBC, greater than a referenced file. The default value of the `vm-ubcseqpercent` attribute is 10 percent.

For example, using the default values, the UBC would have to be larger than 50 percent of all memory and a file would have to be larger than 10 percent of the UBC (that is, the file size would have to be at least 5 percent of all memory) in order for the system to reuse the pages in the UBC.

On large-memory systems that are doing a lot of file system operations, you may want to lower the `vm-ubcseqstartpercent` value to 30 percent. Do not specify a lower value unless you decrease the size of the UBC. In this case, do not change the value of the `vm-ubcseqpercent` attribute.

9.2.8 Disabling File Read Access Time Flushing

When a `read` system call is made to a file system's files, the default behavior is for the file system to update both the in-memory file access time and the on-disk stat structure, which contains most of the file information that is returned by the `stat`(2) system call.

You can improve file system performance for proxy servers by specifying, at mount time, that the file system update only the in-memory file access time when a read system call is made to a file. The file system will update the on-disk stat structure only if the file is modified.

To enable this functionality, use the `mount` command with the `noatimes` option. See `read`(2) and `mount`(8) for more information.

Updating only the in-memory file access time for reads can improve proxy server response time by decreasing the number of disk I/O operations. However, this behavior jeopardizes the integrity of read access time updates and violates POSIX standards. Do not use this functionality if it will affect utilities that use read access times to perform tasks, such as migrating files to different devices.

9.2.9 Using Prestoserve to Cache Only File System Metadata

Prestoserve can improve the overall run-time performance for systems that perform large numbers of synchronous writes. The `prmetaonly` attribute controls whether Prestoserve caches only UFS and AdvFS file system metadata, instead of both metadata and synchronous write data (the default). If the attribute is set to 1 (enabled), Prestoserve caches only file system metadata.

Caching only metadata may improve the performance of applications that access many small files or applications that access a large amount of file-system metadata but do not reread recently written data.

9.2.10 Increasing the Size of the Prestoserve Buffer Hash Table

If the contention on the Prestoserve lock (`presto_lock`) is high (for example, the miss rate is a few percentage points), you may be able to improve throughput by increasing the value of the `presto` subsystem attribute `presto-buffer-hash-size`. This will decrease Prestoserve lock contention.

The default value of the `presto-buffer-hash-size` attribute is 256 bytes. The minimum value is 0; the maximum value is 64 KB.

9.2.11 Caching More Free vnodes

You can increase the minimum number of vnodes on the free list to cache more free vnodes and improve the performance of cache lookup operations. However, increasing the minimim number of vnodes will consume memory resources.

The `vfs` subsystem attribute `min-free-vnodes` specifies the minimum number of vnodes. The default value of the `min-free-vnodes` attribute is either 150 or the value of the `nvnode` kernel variable, whichever is greater.

If the value of `min-free-vnodes` is larger than the value of `max-vnodes`, vnode deallocations will not occur.

If the value of `min-free-vnodes` is close to the value of the `max-vnodes` attribute, vnode deallocation will not be effective. If the value of `min-free-vnodes` must be close to the value of `max-vnodes`, you may want to disable vnode deallocation (see Section 9.2.15).

Disabling vnode deallocation does not free memory, because memory used by the vnodes is not returned to the system. On systems that need to reclaim the memory used by vnodes, make sure that the value of `min-free-vnodes` is significantly lower than the value of `max-vnodes`.

See Section 5.5.1 for information about modifying `max-vnodes`.

9.2.12 Increasing the Time vnodes Remain on the Free List

You can increase the value of the `vfs` subsystem attribute `vnode-age` to increase the amount of time for which vnodes are kept on the free list. This increases the possibility that the vnode will be successfully looked up. The default value for `vnode-age` is 120 seconds on 32-MB or larger systems and 2 seconds on 24-MB systems.

9.2.13 Delaying the Deallocation of vnodes

To delay the deallocation of vnodes, increase the value of the `vfs` subsystem attribute `namei-cache-valid-time`. The default value is 1200. This can improve namei cache lookup operations, but it consumes memory resources.

9.2.14 Accelerating the Deallocation of vnodes

To accelerate the deallocation of vnodes, decrease the value of the `vfs` subsystem attribute `namei-cache-valid-time`. The default value is 1200. This causes vnodes to be deallocated from the namei cache at a faster rate and returns memory to the operating system, but it also reduces the efficiency of the cache.

9.2.15 Disabling vnode Deallocation

To optimize processing time, disable vnode deallocation by setting the value of the `vfs` subsystem attribute `vnode-deallocation-enable` to zero. Disabling vnode deallocation does not free memory, because memory used by the vnodes is not returned to the system.

You may want to disable vnode allocation if the value of the `vfs` subsystem attribute `min-free-vnodes` is close to the value of the `max-vnodes` attribute. See Section 5.5.1 for information about modifying `max-vnodes`.

9.3 Managing Advanced File System Performance

The Advanced File System (AdvFS) provides file system features beyond those of a traditional UFS file system. Unlike the rigid UFS model in which the file system directory hierarchy (tree) is bound tightly to the physical storage, AdvFS consists of two distinct layers: the directory hierarchy layer and the physical storage layer.

The AdvFS decoupled file system structure enables you to manage the physical storage layer apart from the directory hierarchy layer. This means that you can move files between a defined group of disk volumes without changing file pathnames. Because the pathnames remain the same, the action is completely transparent to end users.

AdvFS allows you to put multiple volumes (disks, LSM volumes, or RAID storage sets) in a file domain and distribute the filesets and files across the volumes. A file's blocks usually reside together on the same volume, unless the file is striped or the volume is full. Each new file is placed on the successive volume by using round-robin scheduling.

AdvFS provides the following features:

- High-performance file system

  AdvFS uses an extent-based file allocation scheme that consolidates data transfers, which increases sequential bandwidth and improves performance for large data transfers. AdvFS performs large reads from disk when it anticipates a need for sequential data. AdvFS also performs large writes by combining adjacent data into a single data transfer.

- Fast file system recovery

  Rebooting after a system interruption is extremely fast, because AdvFS uses write-ahead logging, instead of the `fsck` utility, as a way to check for and repair file system inconsistencies. The recovery speed depends on the number of uncommitted records in the log, not the amount of data in the fileset; therefore, reboots are quick and predictable.

- Online file system management
- File domain defragmentation capability
- Support for large files and file systems
- User quotas
- Support for the `salvage` command, which allows you to recover file data from damaged AdvFS file domains

The optional AdvFS utilities, which are licensed separately, provide the following features:

- Pool of storage that allows you to add, remove, and back up disks without disrupting users or applications.
- Disk spanning filesets
- Ability to recover deleted files

  Users can retrieve their own unintentionally deleted files from predefined trashcan directories, without assistance from system administrators.
- I/O load balancing across disks
- Online fileset resizing
- Online file migration across disks
- File-level striping

  File-level striping may improve I/O bandwidth (transfer rates) by distributing file data across multiple disk volumes.
- Graphical user interface (GUI) that simplifies disk and file system administration, provides status, and alerts you to potential problems

The following sections describe how to perform these tasks:

- Understand AdvFS I/O queues and access structures (Section 9.3.1)
- Use the AdvFS guidelines in order to set up a high-performance configuration (Section 9.3.2)
- Obtain information about the AdvFS performance (Section 9.3.3)
- Improve AdvFS performance by tuning the subsystem (Section 9.3.4)

See the *AdvFS Guide to File System Administration* for detailed information about setting up and managing AdvFS.

9.3.1 Understanding AdvFS Operation

AdvFS is a file system option that provides many file management and performance features. You can use AdvFS instead of UFS to organize and manage your files. An AdvFS file domain can consist of multiple volumes, which can be UNIX block devices (entire disks), disk partitions, LSM logical volumes, or RAID storage sets. AdvFS filesets can span all the volumes in the file domain.

The AdvFS Utilities product, which is licensed separately from the operating system, extends the capabilities of the AdvFS file system.

The following sections describe AdvFS I/O queues and access structures.

9.3.1.1 AdvFS I/O Queues

At boot time, the system reserves a percentage of static wired physical memory for the AdvFS buffer cache, which is the part of the UBC that holds the most recently accessed pages of AdvFS file data and metadata. A disk operation is avoided if the data is later reused and the page is still in the cache (a buffer cache hit). This can improve AdvFS performance.

The amount of memory that can be allocated to the AdvFS buffer cache is specified by the `advfs` subsystem attribute `AdvfsCacheMaxPercent`. The default value is 7 percent of physical memory. See Section 6.1.2.3 for information about how the system allocates memory to the AdvFS buffer cache.

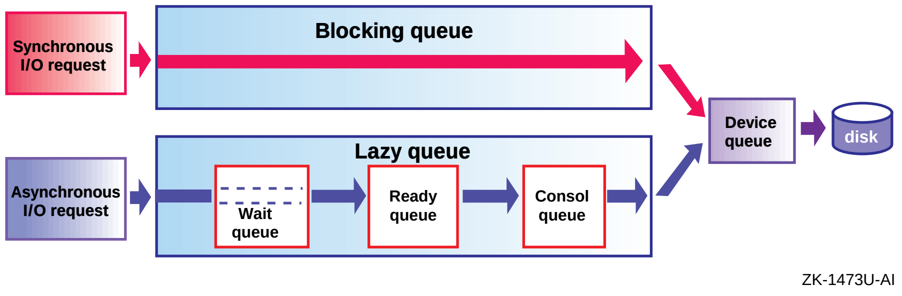
For each AdvFS volume, I/O requests are sent either to the blocking queue, which caches synchronous I/O requests, or to the lazy queue, which caches asynchronous I/O requests. Both the blocking queue and the lazy queue feed I/O requests to the device queue.

A synchronous I/O request is one that must be written to disk before the write is considered successful and the application can continue. This ensures data reliability because the write is not stored in memory to be later written to disk. Therefore, I/O requests on the blocking queue cannot be asynchronously removed, because the I/O must complete.

Asynchronous I/O requests are cached in the lazy queue and periodically flushed to disk in portions that are large enough to allow the disk drivers to optimize the order of the write.

Figure 9–1 shows the movement of synchronous and asynchronous I/O requests through the AdvFS I/O queues.

Figure 9–1: AdvFS I/O Queues



When an asynchronous I/O request enters the lazy queue, it is assigned a time stamp. The lazy queue is a pipeline that contains a sequence of queues through which an I/O request passes: the wait queue (if applicable), the ready queue, and the consol queue. An AdvFS buffer cache hit can occur while an I/O request is in any part of the lazy queue.

Detailed descriptions of the wait, ready, and consol queues are as follows:

- wait queue—Asynchronous I/O requests that are waiting for an AdvFS transaction log write to complete first enter the wait queue. Each file domain has a transaction log that keeps track of fileset activity for all filesets in the file domain, and ensures AdvFS metadata consistency if a crash occurs.

  AdvFS uses write-ahead logging, which requires that when metadata is modified, the transaction log write must complete before the actual metadata is written. This ensures that AdvFS can always use the transaction log to create a consistent view of the file system metadata. After the transaction log is written, I/O requests are moved from the wait queue to the ready queue.

- ready queue—Asynchronous I/O requests that are not waiting for an AdvFS transaction log write to complete enter the ready queue, where they are sorted and held until the size of the ready queue reaches the value specified by the `AdvfsReadyQLim` attribute, or until the `update` daemon flushes the data. The default value of the `AdvfsReadyQLim` attribute is 16,384 512-byte blocks (8 MB).

  You can modify the size of the ready queue for all AdvFS volumes by changing the value of the `AdvfsReadyQLim` attribute. You can modify the size for a specific AdvFS volume by using the `chvol -t` command.

  You can disable data caching in the ready queue and allow I/O requests to bypass the ready queue. To do this, specify a value of 0 for the `AdvfsReadyQLim` attribute However, this is not recommended. See Section 9.3.4.5 for information about tuning the ready queue.

- consol queue—I/O requests are moved from the ready queue to the consol queue, which feeds the device queue.

Both the consol queue and the blocking queue feed the device queue, where logically contiguous I/O requests are consolidated into larger I/Os before they are sent to the device driver. The size of the device queue affects the amount of time it takes to complete a synchronous (blocking) I/O operation. AdvFS issues several types of blocking I/O operations, including AdvFS metadata and log data operations.

The `AdvfsMaxDevQLen` attribute limits the total number of I/O requests on the AdvFS device queue. The default value is 24 requests. When the number of requests exceeds this value, only synchronous requests from the blocking queue are accepted onto the device queue.

Although the default value of the `AdvfsMaxDevQLen` attribute is appropriate for most configurations, you may need to modify this value. However, only increase the default value if devices are not being kept busy. Make sure that increasing the size of the device queue does not cause a decrease in response time. See Section 9.3.4.6 for more information about tuning the AdvFS device queue.

Use the `advfsstat` command to show the AdvFS queue statistics.

9.3.1.2 AdvFS Access Structures

If your users or applications open and then reuse many files, you may be able to improve AdvFS performance by modifying how the system allocates AdvFS access structures. AdvFS access structures are in-memory data structures that AdvFS uses to cache low-level information about files that are currently open and files that were opened but are now closed. Caching open file information can enhance AdvFS performance if the open files are later reused.

At boot time, the system reserves for AdvFS access structures a percentage of the physical memory that is not wired by the kernel or applications. Out of this pool of reserved memory, the system allocates a number of access structures and places them on the access structure free list. When a file is opened, an access structure is taken from the access structure free list. Access structures are allocated and deallocated according to the kernel configuration and workload demands.

There are two attributes that control the allocation of AdvFS access structures:

- The `AdvfsAccessMaxPercent` attribute controls the amount of pageable memory (`malloc`) that is reserved for AdvFS access structures. The default value is 80 percent of pageable memory.
- The `AdvfsPreallocAccess` specifies the number of AdvFS access structures that the system allocates at startup time. The default and minimum values are 128. The maximum value is either 65536 or the value of the `AdvfsAccessMaxPercent` attribute, whichever is the smallest value.

You may be able to improve AdvFS performance by modifying the previous attributes and allocating more memory for AdvFS access structures. However, this will reduce the amount of memory available to processes and may cause excessive paging and swapping.

If you do not use AdvFS or if your workload does not reuse AdvFS files, do not allocate a large amount of memory for access structures. If you have a large-memory system, you may want to decrease the amount of memory reserved for AdvFS access structures.

See Section 9.3.4.3 for information about tuning access structures.

9.3.2 AdvFS Configuration Guidelines

You will obtain the best performance if you carefully plan your AdvFS configuration. Table 9–2 lists AdvFS configuration guidelines and performance benefits as well as tradeoffs.

Table 9–2: AdvFS Configuration Guidelines

Action	Performance Benefit	Tradeoff
Use multiple-volume file domains (Section 9.3.2.1)	Improves throughput and simplifies management	Increases chance of domain failure and may cause a log bottleneck
Use several file domains instead of one large domain (Section 9.3.2.1)	Prevents log from becoming a bottleneck	Increases maintenance complexity
Place transaction log on fast or uncongested volume (Section 9.3.2.2)	Prevents log from becoming a bottleneck	None
Stripe files across different disks and, if possible, different buses (Section 9.3.2.4)	Improves sequential read and write performance	Increases chance of domain failure
Use quotas (Section 9.3.2.5)	Controls file system space utilization	None

The following sections describe these AdvFS configuration guidelines in detail.

9.3.2.1 Using Multiple-Volume File Domains

Using multiple-volume file domains allows greater control over your physical resources, and may improve a fileset's total throughput. However, be sure that the log does not become a bottleneck. Multiple-volume file domains improve performance because AdvFS generates parallel streams of output using multiple device consolidation queues.

In addition, using only a few file domains instead of using many file domains reduces the overall management effort, because fewer file domains require less administration. However, a single volume failure within a file domain renders the entire file domain inaccessible. Therefore, the more volumes that you have in your file domain the greater the risk that a file domain will fail.

It is recommended that you use a maximum of 12 volumes in each file domain. However, to reduce the risk of file domain failure, limit the number of volumes per file domain to three or mirror data with LSM or hardware RAID.

For multiple-volume domains, make sure that busy files are not located on the same volume. Use the `migrate` command to move files across volumes.

9.3.2.2 Improving the Transaction Log Performance

Each file domain has a transaction log that tracks fileset activity for all filesets in the file domain, and ensures AdvFS metadata consistency if a crash occurs. The AdvFS file domain transaction log may become a bottleneck if the log resides on a congested disk or bus, or if the file domain contains many filesets.

To prevent the log from becoming a bottleneck, put the log on a fast, uncongested volume. You may want to put the log on a disk that contains only the log. See Section 9.3.4.12 for information on moving an existing transaction log.

To make the transaction log highly available, use LSM or hardware RAID to mirror the log.

9.3.2.3 Improving Bitmap Metadata Table Performance

The AdvFS fileset data structure (metadata) is stored in a file called the bitfile metadata table (BMT). Each volume in a domain has a BMT that describes the file extents on the volume. If a domain has multiple volumes of the same size, files will be distributed evenly among the volumes.

The BMT is the equivalent of the UFS inode table. However, the UFS inode table is statically allocated, while the BMT expands as more files are added to the domain. Each time that AdvFS needs additional metadata, the BMT grows by a fixed size (the default is 128 pages). As a volume becomes increasingly fragmented, the size by which the BMT grows may be described by several extents.

To monitor the BMT, use the `vbmtpg` command and examine the number of mcells (`freeMcellCnt`). The value of `freeMcellCnt` can range from 0 to 22. A volume with 1 free mcell has very little space in which to grow the BMT. See `vbmtpg`(8) for more information.

You can also invoke the `showfile` command and specify *`mount_point/`*`.tags/M-10` to examine the BMT extents on the first domain volume that contains the fileset mounted on the specified mount point. To examine the extents of the other volumes in the domain, specify `M-16`, `M-24`, and so on. If the extents at the end of the BMT are smaller than the extents at the beginning of the file, the BMT is becoming fragmented. See `showfile`(8) for more information.

If you are prematurely out of BMT disk space, you may be able to eliminate the problem by defragmenting the file domain that contains the volume. See `defragment`(8) for more information.

Table 9–3 provides some BMT sizing guidelines for the number of pages to preallocate for the BMT, and the number of pages by which the BMT extent size grows. The BMT sizing depends on the maximum number of files you expect to create on a volume.

Table 9–3: BMT Sizing Guidelines

Estimated Maximum Number of Files on a Volume	Number of Pages to Preallocate	Number of Pages to Grow Extent
< 50,000	3600	128
100,000	7200	256
200,000	14,400	512
300,000	21,600	768
400,000	28,800	1024
800,000	57,600	2048

You can modify the number of extent pages by which the BMT grows when a file domain is created or when a volume is added to the domain. If you use the `mkfdmn -x` or the `addvol -x` command when there is a large amount of free space on a disk, as files are created the BMT will expand by the specified number of pages and those pages will be in one extent. As the disk becomes more fragmented, the BMT will still expand, but the pages will not be contiguous and will require more extents. Eventually, the BMT will run out of its limited number of extents even though the growth size is large.

To prevent this problem, you can preallocate space for the BMT when the file domain is created, or when a volume is added to the domain. If you use the `mkfdmn -p` or the `addvol -p` command, the preallocated BMT is described in one extent. All subsequent growth will be able to utilize nearly all of the limited number of BMT extents. See `mkfdmn`(8) and `addvol`(8).

Do not overallocate BMT space because the disk space cannot be used for other purposes. However, too little BMT space will eventually cause the BMT to grow by a fixed amount. The disk may be fragmented and the growth will require multiple extents.

9.3.2.4 Striping Files

You may be able to use the AdvFS `stripe` utility to improve the read and write performance of an individual file by spreading file data evenly across different disks in a file domain. For the maximum performance benefit, stripe files across disks on different I/O buses.

Striping files, instead of striping entire disks, is useful if an application continually accesses only a few specific files. Do not stripe both a file and the disk on which it resides.

The `stripe` utility directs a zero-length file (a file with no data written to it yet) to be distributed evenly across a specified number of volumes. As data is appended to the file, the data is spread across the volumes. The size of each data segment (also called the stripe or chunk size) is 64 KB (65,536 bytes). AdvFS alternates the placement of the segments on the disks in a sequential pattern. For example, the first 64 KB of the file is written to the first volume, the second 64 KB is written to the next volume, and so on.

See `stripe`(8) for more information.

Note

Distributing data across multiple volumes decreases data availability, because one volume failure makes the entire file domain unavailable. To make striped files highly available, you can mirror the disks on which the file is striped.

To determine if you should stripe files, use the `iostat` utility, as described in Section 8.2.1. The blocks per second and I/O operations per second should be cross-checked with the disk's bandwidth capacity. If the disk access time is slow, in comparison to the stated capacity, then file striping may improve performance.

The performance benefit of striping also depends on the size of the average I/O transfer in relation to the data segment (stripe) size, in addition to how your users and applications perform disk I/O.

9.3.2.5 Using AdvFS Quotas

AdvFS quotas allow you to track and control the amount of physical storage that a user, group, or fileset consumes. AdvFS eliminates the slow reboot activities associated with UFS quotas. In addition, AdvFS quota information is always maintained, but quota enforcement can be activated and deactivated.

For information about AdvFS quotas, see the *AdvFS Administration* manual.

9.3.3 Gathering AdvFS Information

Table 9–4 describes the tools you can use to obtain information about AdvFS.

Table 9–4: AdvFS Monitoring Tools

Name	Use	Description
`advfsstat`	Displays AdvFS performance statistics (Section 9.3.3.1)	Allows you to obtain extensive AdvFS performance information, including buffer cache, fileset, volume, and bitfile metadata table (BMT) statistics, for a specific interval of time.
`advscan`	Identifies disks in a file domain (Section 9.3.3.2)	Locates pieces of AdvFS file domains on disk partitions and in LSM disk groups.
`showfdmn`	Displays detailed information about AdvFS file domains and volumes (Section 9.3.3.3)	Allows you to determine if files are evenly distributed across AdvFS volumes. The `showfdmn` utility displays information about a file domain, including the date created and the size and location of the transaction log, and information about each volume in the domain, including the size, the number of free blocks, the maximum number of blocks read and written at one time, and the device special file. For multivolume domains, the utility also displays the total volume size, the total number of free blocks, and the total percentage of volume space currently allocated.
`showfile`	Displays information about files in an AdvFS fileset (Section 9.3.3.4)	Displays detailed information about files (and directories) in an AdvFS fileset. The `showfile` command allows you to check a file's fragmentation. A low performance percentage (less than 80 percent) indicates that the file is fragmented on the disk. The `showfile` command also displays the extent map of each file. An extent is a contiguous area of disk space that AdvFS allocates to a file. Simple files have one extent map; striped files have an extent map for every stripe segment. The extent map shows whether the entire file or only a portion of the file is fragmented.

Table 9–4: AdvFS Monitoring Tools (cont.)

Name	Use	Description
`showfsets`	Displays AdvFS fileset information for a file domain (Section 9.3.3.5)	Displays information about the filesets in a file domain, including the fileset names, the total number of files, the number of free blocks, the quota status, and the clone status. The `showfsets` command also displays block and file quota limits for a file domain or for a specific fileset in the domain.
`verify`	Checks the AdvFS on-disk metadata structures	Checks AdvFS on-disk structures such as the BMT, the storage bitmaps, the tag directory and the frag file for each fileset. It verifies that the directory structure is correct and that all directory entries reference a valid file (tag) and that all files (tags) have a directory entry.
`fsx`	Exercises file systems	Exercises AdvFS and UFS file systems by creating, opening, writing, reading, validating, closing, and unlinking a test file. Errors are written to a log file. See `fsx`(8) for more information.

The following sections describe some of these commands in detail.

9.3.3.1 Monitoring AdvFS Performance Statistics by Using the advfsstat Command

The `advfsstat` command displays various AdvFS performance statistics and monitors the performance of AdvFS domains and filesets. Use this command to obtain detailed information, especially if the `iostat` command output indicates a disk bottleneck (see Section 8.2.1).

The `advfsstat` command displays detailed information about a file domain, including information about the AdvFS buffer cache, fileset vnode operations, locks, the namei cache, and volume I/O performance. The command reports information in units of one disk block (512 bytes) for each interval of time (the default is one second). You can use the `-i` option to output information at specific time intervals.

The following example of the `advfsstat -v 2` command shows the I/O queue statistics for the specified volume:

```
# /usr/sbin/advfsstat -v 2 test_domain
vol1
  rd  wr  rg  arg  wg  awg  blk  wlz  rlz  con  dev
  54   0  48  128   0    0    0    1    0    0   65
```

The previous example shows the following fields:

- Read and write requests—Compare the number of read requests (rd) to the number of write requests (wr). Read requests are blocked until the read completes, but write requests will not block the calling thread, which increases the throughput of multiple threads.
- Consolidated reads and writes—You may be able to improve performance by consolidating reads and writes. The consolidated read values (rg and arg) and write values (wg and awg) indicate the number of disparate reads and writes that were consolidated into a single I/O to the device driver. If the number of consolidated reads and writes decreases compared to the number of reads and writes, AdvFS may not be consolidating I/O.
- I/O queue values—The blk, wlz, rlz, con, and dev fields can indicate potential performance issues. The con value specifies the number of entries on the consolidate queue. These entries are ready to be consolidated and moved to the device queue. The device queue value (dev) shows the number of I/O requests that have been issued to the device controller. The system must wait for these requests to complete.

  If the number of I/O requests on the device queue increases continually and you experience poor performance, applications may be I/O bound on this device. You may be able to eliminate the problem by adding more disks to the domain or by striping disks.

You can monitor the type of requests that applications are issuing by using the advfsstat command's -f option to display fileset vnode operations. You can display the number of file creates, reads, and writes and other operations for a specified domain or fileset. For example:

```
# /usr/sbin/advfsstat -i 3 -f 2 scratch_domain fset1
  lkup  crt geta read writ fsnc dsnc   rm   mv rdir  mkd  rmd link
     0    0    0    0    0    0    0    0    0    0    0    0    0
     4    0   10    0    0    0    0    2    0    2    0    0    0
     0    0    0    0    0    0    0    0    0    0    0    0    0
     0    0    0    0    0    0    0    0    0    0    0    0    0
    24    8   51    0    9    0    0    3    0    0    4    0    0
  1201  324 2985    0  601    0    0  300    0    0    0    0    0
  1275  296 3225    0  655    0    0  281    0    0    0    0    0
  1217  305 3014    0  596    0    0  317    0    0    0    0    0
  1249  304 3166    0  643    0    0  292    0    0    0    0    0
  1175  289 2985    0  601    0    0  299    0    0    0    0    0
   779  148 1743    0  260    0    0  182    0   47    0    4    0
     0    0    0    0    0    0    0    0    0    0    0    0    0
     0    0    0    0    0    0    0    0    0    0    0    0    0
```

See advfsstat(8) for more information.

Note that it is difficult to link performance problems to some statistics such as buffer cache statistics. In addition, lock performance that is related to lock statistics cannot be tuned.

9.3.3.2 Identifying Disks in an AdvFS File Domain by Using the advscan Command

The advscan command locates pieces of AdvFS domains on disk partitions and in LSM disk groups. Use the advscan command when you have moved disks to a new system, have moved disks around in a way that has changed device numbers, or have lost track of where the domains are.

You can specify a list of volumes or disk groups with the advscan command to search all partitions and volumes. The command determines which partitions on a disk are part of an AdvFS file domain.

You can also use the advscan command for repair purposes if you deleted the /etc/fdmns directory, deleted a directory domain under /etc/fdmns, or deleted some links from a domain directory under /etc/fdmns.

You can run the advscan command to rebuild all or part of your /etc/fdmns directory, or you can manually rebuild it by supplying the names of the partitions in a domain.

The following example scans two disks for AdvFS partitions:

```
# /usr/advfs/advscan rz0 rz5

Scanning disks  rz0 rz5
Found domains:
usr_domain
                Domain Id       2e09be37.0002eb40
                Created         Thu Jun 26 09:54:15 1998
                Domain volumes          2
                /etc/fdmns links        2
                Actual partitions found:
                                        rz0c
                                        rz5c
```

For the following example, the rz6 file domains were removed from /etc/fdmns. The advscan command scans device rz6 and re-creates the missing domains.

```
# /usr/advfs/advscan -r rz6

Scanning disks  rz6
Found domains:
*unknown*
                Domain Id       2f2421ba.0008c1c0
                Created         Mon Jan 20 13:38:02 1998

                Domain volumes          1
                /etc/fdmns links        0
                Actual partitions found:
                                        rz6a*


*unknown*
                Domain Id       2f535f8c.000b6860
                Created         Tue Feb 25 09:38:20 1998

                Domain volumes          1
                /etc/fdmns links       0

                Actual partitions found:
                                        rz6b*


Creating /etc/fdmns/domain_rz6a/
        linking rz6a

Creating /etc/fdmns/domain_rz6b/
        linking rz6b
```

See advscan(8) for more information.

9.3.3.3 Checking AdvFS File Domains by Using the showfdmn Command

The showfdmn command displays the attributes of an AdvFS file domain and detailed information about each volume in the file domain.

The following example of the showfdmn command displays domain information for the usr file domain:

```
% /sbin/showfdmn usr

               Id              Date Created  LogPgs  Domain Name
2b5361ba.000791be  Tue Jan 12 16:26:34 1998     256  usr

Vol 512-Blks    Free % Used  Cmode  Rblks  Wblks  Vol Name
 1L   820164  351580    57%     on    256    256  /dev/disk/rz0d
```

See showfdmn(8) for more information about the output of the command.

9.3.3.4 Displaying AdvFS File Information by Using the showfile Command

The showfile command displays the full storage allocation map (extent map) for one or more files in an AdvFS fileset. An extent is a contiguous area of disk space that AdvFS allocates to a file.

The following example of the `showfile` command displays the AdvFS characteristics for all of the files in the current working directory:

```
# /usr/sbin/showfile *

       Id  Vol  PgSz  Pages  XtntType  Segs  SegSz    I/O  Perf File
  22a.001    1    16      1    simple    **     **  async  50%  Mail
    7.001    1    16      1    simple    **     **  async  20%  bin
  1d8.001    1    16      1    simple    **     **  async  33%  c
 1bff.001    1    16      1    simple    **     **  async  82%  dxMail
  218.001    1    16      1    simple    **     **  async  26%  emacs
  1ed.001    1    16      0    simple    **     **  async 100%  foo
  1ee.001    1    16      1    simple    **     **  async  77%  lib
  1c8.001    1    16      1    simple    **     **  async  94%  obj
  23f.003    1    16      1    simple    **     **  async 100%  sb
 170a.008    1    16      2    simple    **     **  async  35%  t
    6.001    1    16     12    simple    **     **  async  16%  tmp
```

The `I/O` column specifies whether write operations are forced to be synchronous. See Section 9.3.4.10 for information.

The following example of the `showfile` command shows the characteristics and extent information for the `tutorial` file, which is a simple file:

```
# /usr/sbin/showfile -x tutorial

        Id  Vol  PgSz  Pages  XtntType  Segs  SegSz    I/O  Perf    File
 4198.800d    2    16     27    simple    **     **  async   66% tutorial

     extentMap: 1
          pageOff    pageCnt    vol    volBlock    blockCnt
                0          5      2      781552          80
                5         12      2      785776         192
               17         10      2      786800         160
       extentCnt: 3
```

The `Perf` entry shows the efficiency of the file-extent allocation, expressed as a percentage of the optimal extent layout. A high value, such as 100 percent, indicates that the AdvFS I/O subsystem is highly efficient. A low value indicates that files may be fragmented.

See `showfile`(8) for more information about the command output.

9.3.3.5 Displaying the AdvFS Filesets in a File Domain by Using the showfsets Command

The `showfsets` command displays the AdvFS filesets (or clone filesets) and their characteristics in a specified domain.

The following is an example of the `showfsets` command:

```
# /sbin/showfsets dmn
mnt
        Id           : 2c73e2f9.000f143a.1.8001
```

```
          Clone is     : mnt_clone
          Files        :       79,  limit =     1000
          Blocks  (1k) :      331,  limit =    25000
          Quota Status : user=on  group=on

mnt_clone
          Id           : 2c73e2f9.000f143a.2.8001
          Clone of     : mnt
          Revision     : 1
```

See `showfsets`(8) for information about the options and output of the command.

9.3.4 Tuning AdvFS

After you configure AdvFS, you may be able to tune it to improve performance. To successfully improve performance, you must understand how your applications and user perform file system I/O, as described in Section 2.1.

Table 9–5 lists AdvFS tuning guidelines and performance benefits as well as tradeoffs. In addition, the recommendations described in Table 9–1 apply to AdvFS configurations.

Table 9–5: AdvFS Tuning Guidelines

Action	Performance Benefit	Tradeoff
Decrease the size of the metadata buffer cache to 1 percent (Section 6.4.6)	Improves performance for systems that use only AdvFS	None
Increase the percentage of memory allocated for the AdvFS buffer cache (Section 9.3.4.1)	Improves AdvFS performance if data reuse is high	Consumes memory
Increase the number of AdvFS buffer hash chains (Section 9.3.4.2)	Speeds lookup operations and decreases CPU usage	Consumes memory
Increase the memory reserved for AdvFS access structures (Section 9.3.4.3)	Improves AdvFS performance for systems that open and reuse files	Decreases the memory available to the virtual memory subsystem and the UBC
Defragment file domains (Section 9.3.4.4)	Improves read and write performance	None

Table 9–5: AdvFS Tuning Guidelines (cont.)

Action	Performance Benefit	Tradeoff
Increase the amount of data cached in the ready queue (Section 9.3.4.5)	Improves asynchronous write performance	May cause I/O spikes or increase the number of lost buffers if a crash occurs
Decrease the maximum number of I/O requests on the device queue (Section 9.3.4.6)	Decreases the time to complete synchronous I/O requests and improves response time	May cause I/O spikes
Decrease the I/O transfer read-ahead size (Section 9.3.4.7)	Improves performance for `mmap` page faulting	None
Disable the flushing of dirty pages mapped with the `mmap` function during a `sync` call (Section 9.3.4.8)	May improve performance for applications that manage their own flushing	None
Consolidate I/O transfers (Section 9.3.4.9)	Improves AdvFS performance	None
Force all AdvFS file writes to be synchronous (Section 9.3.4.10)	Ensures that data is successfully written to disk	May degrade file system performance
Prevent partial writes (Section 9.3.4.11)	Ensures that system crashes do not cause partial disk writes	May degrade asynchronous write performance
Move the transaction log to a fast or uncongested volume (Section 9.3.4.12)	Prevents log from becoming a bottleneck	None
Balance files across volumes in a file domain (Section 9.3.4.13)	Improves performance and evens the future distribution of files	None
Migrate frequently used or large files to different file domains (Section 9.3.4.14)	Improves I/O performance	None

The following sections describe the AdvFS tuning recommendations in detail.

9.3.4.1 Modifying the Size of the AdvFS Buffer Cache

The `advfs` subsystem attribute `AdvfsCacheMaxPercent` specifies the maximum percentage of physical memory that can be used to cache AdvFS file data. Caching AdvFS data can improve I/O performance only if the cached data is reused.

If data reuse is high, you may be able to improve AdvFS performance by increasing the percentage of memory allocated to the AdvFS buffer cache. To do this, increase the value of the `AdvfsCacheMaxPercent` attribute. The default is 7 percent of memory, and the maximum is 30 percent. If you increase the value of the `AdvfsCacheMaxPercent` attribute and experience no performance benefit, return to the original value.

Note that the AdvFS buffer cache cannot be more than 50 percent of the UBC.

Increasing the memory allocated to the AdvFS buffer cache will decrease the amount of memory available for processes; make sure that you do not cause excessive paging and swapping. Use the `vmstat` command to check virtual memory statistics, as described in Section 6.3.2.

If your workload does not reuse AdvFS data or if you have more than 2 GB of memory, you may want to decrease the size of the AdvFS buffer cache. The minimum value is 1 percent of physical memory. This can improve performance, because it decreases the overhead associated with managing the cache and also frees memory.

See Section 4.4 for information about modifying kernel subsystem attributes.

9.3.4.2 Increasing the Number of AdvFS Buffer Hash Chains

The hash chain table for the AdvFS buffer cache is used to locate pages of AdvFS file data in memory. The table contains a number of hash chains, which contain elements that point to pages of file system data that have already been read into memory. When a `read` or `write` system call is done for a particular offset within an AdvFS file, the system sequentially searches the appropriate hash chain to determine if the file data is already in memory.

The value of the `advfs` subsystem attribute `AdvfsCacheHashSize` specifies the number of hash chains on the table. The default value is either 8192 KB or 10 percent of the size of the AdvFS buffer cache (rounded up to the next power of 2), whichever is the smallest value. The minimum value is 1024 KB. The maximum value is either 65536 or the size of the AdvFS buffer cache, whichever is the smallest value. The `AdvfsCacheMaxPercent` attribute specifies the size of the AdvFS buffer cache (see Section 9.3.4.1).

If you have more than 4 GB of memory, you may want to increase the value of the `AdvfsCacheHashSize` attribute, which will increase the number of hash chains on the table. The more hash chains on the table, the shorter the hash chains. Short hash chains contain less elements to search, which results in fast seach speeds and decreases CPU usage.

For example, you can double the default value of the AdvfsCacheHashSize attribute if the system is experiencing high CPU system time, or if a kernel profile shows high percentage of CPU usage in the find_page routine.

Increasing the size of the AdvFS buffer cache hash table will increase the amount of kernel wired memory in the system.

See Section 4.4 for information about modifying kernel subsystem attributes.

9.3.4.3 Increasing the Memory Reserved for Access Structures

At boot time, the system reserves a percentage of pageable memory (memory that is not wired by the kernel or applications) for AdvFS access structures. If your system opens and then reuses many files (for example, if you have a proxy server), you may be able to improve AdvFS performance by increasing the number of AdvFS access structures that the system places on the access structure free list at startup time.

AdvFS access structures are in-memory data structures that AdvFS uses to cache low-level information about files that are currently open, and files that were opened but are now closed. Increasing the number of access structures on the free list allows more open file information (metadata) to remain in the cache, which can improve AdvFS performance if the files are reused. See Section 9.3.1.2 for more information about access structures.

Use the advfs subsystem attribute AdvfsPreallocAccess to modify the number of AdvFS access structures that the system allocates at startup time. The default and minimum values are 128 if you have a mounted AdvFS fileset. The maximum value is either 65536 or the value of the advfs subsystem attribute AdvfsAccessMaxPercent, whichever is the smallest value.

The AdvfsAccessMaxPercent attribute specifies the maximum percentage of pageable memory (malloc pool) that can be reserved for AdvFS access structures. The minimum value is 5 percent of pageable memory, and the maximum value is 95 percent. The default value is 80 percent.

Increasing the value of the AdvfsAccessMaxPercent attribute allows you to allocate more memory resources for access structures, which may improve AdvFS performance on systems that open and reuse many files. However, increasing the memory available for access structures will decrease the memory that is available to processes, which may cause excessive paging and swapping.

Decreasing the value of the AdvfsAccessMaxPercent attribute frees pageable memory but you will be able to allocate less memory for AdvFS access structures, which may degrade AdvFS performance on systems that open and reuse many files.

See Section 4.4 for information about modifying kernel subsystem attributes.

9.3.4.4 Defragmenting a File Domain

AdvFS attempts to store file data in a collection of contiguous blocks (a file extent) on a disk. If all data in a file is stored in contiguous blocks, the file has one file extent. However, as files grow, contiguous blocks on the disk may not be available to accommodate the new data, so the file must be spread over discontiguous blocks and multiple file extents.

File fragmentation degrades read and write performance because many disk addresses must be examined to access a file. In addition, if a domain has a large number of small files, you may prematurely run out of disk space due to fragmentation.

Use the `defragment` utility to reduce the amount of file fragmentation in a file domain by attempting to make the files more contiguous, which reduces the number of file extents. The utility does not affect data availability and is transparent to users and applications. Striped files are not defragmented.

Use the `defragment` utility with the `–v` and `–n` options to show the amount of file fragmentation.

You can improve the efficiency of the defragmenting process by deleting any unneeded files in the file domain before running the `defragment` utility. See `defragment`(8) for more information.

9.3.4.5 Increasing the Amount of Data Cached in the Ready Queue

AdvFS caches asynchronous I/O requests in the AdvFS buffer cache. If the cached data is later reused, pages can be retrieved from memory, and a disk operation is avoided.

Asynchronous I/O requests are sorted in the ready queue and remain there until the size of the queue reaches the value specified by the `AdvfsReadyQLim` attribute or until the `update` daemon flushes the data. The default value of the `AdvfsReadyQLim` attribute is 16,384 512-byte blocks (8 MB). See Section 9.3.1.1 for more information about AdvFS queues.

You can modify the size of the ready queue for all AdvFS volumes by changing the value of the `AdvfsReadyQLim` attribute. You can modify the size for a specific AdvFS volume by using the `chvol -t` command. See `chvol`(8) for more information.

If you have high data reuse (data is repeatedly read and written), you may want to increase the size of the ready cache. This can increase the number of AdvFS buffer cache hits. If you have low data reuse, you can decrease the threshold, but it is recommended that you use the default value.

If you change the size of the ready queue and performance does not improve, return to the original value.

Although you can specify a value of 0 for the `AdvfsReadyQLim` attribute to disable data caching in the ready queue and allow I/O requests to bypass the ready queue, this is not recommended.

See Section 4.4 for information about modifying kernel subsystem attributes.

9.3.4.6 Decreasing the Maximum Number of I/O Requests on the Device Queue

Small, logically contiguous synchronous and asynchronous AdvFS I/O requests are consolidated into larger I/O requests on the device queue, before they are sent to the device driver. See Section 9.3.1.1 for more information about AdvFS queues.

The `AdvfsMaxDevQLen` attribute controls the maximum number of I/O requests on the device queue. When the number of requests on the queue exceeds this value, only synchronous requests are accepted onto the device queue. The default value of the `AdvfsMaxDevQLen` attribute is 24 requests.

Although the default value of the `AdvfsMaxDevQLen` attribute is appropriate for most configurations, you may need to modify this value. Increase the default value of the `AdvfsMaxDevQLen` attribute only if devices are not being kept busy. A guideline is to specify a value for the `AdvfsMaxDevQLen` attribute that is less than or equal to the average number of I/O operations that can be performed in 0.5 seconds.

Make sure that increasing the size of the device queue does not cause a decrease in response time. To calculate response time, multiply the value of the `AdvfsMaxDevQLen` attribute by the average I/O latency time for your disks.

Decreasing the size of the device queue decreases the amount of time it takes to complete a synchronous (blocking) I/O operation and can improve response time.

If you do not want to limit the number of requests on the device queue, set the value of the `AdvfsMaxDevQLen` attribute to 0 (zero), although this behavior is not recommended.

See Section 4.4 for information about modifying kernel subsystem attributes.

9.3.4.7 Decreasing the I/O Transfer Size

AdvFS reads and writes data by a fixed number of 512-byte blocks. The default value depends on the disk driver's reported preferred transfer size. For example, a common default value is either 128 blocks or 256 blocks.

Use the `chvol` command with the `-r` option to change the read-ahead size. You may be able to improve performance for `mmap` page faulting and reduce read-ahead paging and cache dilution by decreasing the read-ahead size.

Use the `chvol` command with the `-w` option to change the write-consolidation size. See `chvol`(8) for more information.

If the disk is fragmented so that the pages of a file are not sequentially allocated, reduce fragmentation by using the `defragment` utility. See `defragment`(8) for more information.

9.3.4.8 Disabling the Flushing of Modified mmapped Pages

The AdvFS buffer cache can contain modified data due to a `write` system call or a memory write reference after an `mmap` system call. The `update` daemon runs every 30 seconds and issues a `sync` call for every fileset mounted with read and write access.

The `AdvfsSyncMmapPages` attribute controls whether modified (dirty) mmapped pages are flushed to disk during a `sync` system call. If the `AdvfsSyncMmapPages` attribute is set to 1 (the default), the modified mmapped pages are asynchronously written to disk. If the `AdvfsSyncMmapPages` attribute is set to 0, modified mmapped pages are not written to disk during a `sync` system call.

If your applications manage their own `mmap` page flushing, set the value of the `AdvfsSyncMmapPages` attribute to zero.

See `mmap`(2) and `msync`(2) for more information.

See Section 4.4 for information about modifying kernel subsystem attributes.

9.3.4.9 Consolidating I/O Transfers

By default, AdvFS consolidates a number of I/O transfers into a single, large I/O transfer, which can improve AdvFS performance. To enable the consolidation of I/O transfers, use the `chvol` command with the `–c on` option.

It is recommended that you not disable the consolidation of I/O transfers. See `chvol`(8) for more information.

9.3.4.10 Forcing Synchronous Writes

By default, asynchronous write requests are cached in the AdvFS buffer cache, and the `write` system call then returns a success value. The data is written to disk at a later time (asynchronously).

You can use the `chfile -l on` command to force all write requests to a specified AdvFS file to be synchronous. If you enable forced synchronous writes on a file, data must be successfully written to disk before the `write` system call will return a success value. This behavior is similar to the behavior associated with a file that has been opened with the `O_SYNC` option; however, forcing synchronous writes persists across `open()` calls.

Forcing all writes to a file to be synchronous ensures that the write has completed when the `write` system call returns a success value. However, it may degrade performance.

A file cannot have both forced synchronous writes enabled and atomic write data logging enabled. See Section 9.3.4.11 for more information.

Use the `chfile` command to determine whether forced synchronous writes or atomic write data logging is enabled. Use the `chfile -l off` command to disable forced synchronous writes (the default behavior).

9.3.4.11 Preventing Partial Data Writes

AdvFS writes data to disk in 8-KB chunks. By default, and in accordance with POSIX standards, AdvFS does not guarantee that all or part of the data will actually be written to disk if a crash occurs during or immediately after the write. For example, if the system crashes during a write that consists of two 8-KB chunks of data, only a portion (anywhere from 0 to 16 KB) of the total write may have succeeded. This can result in partial data writes and inconsistent data.

To prevent partial writes if a system crash occurs, use the `chfile -L on` command to enable atomic write data logging for a specified file.

By default, each file domain has a transaction log file that tracks fileset activity and ensures that AdvFS can maintain a consistent view of the file system metadata if a crash occurs. If you enable atomic write data logging on a file, data from a write call will be written to the transaction log file before it is written to disk. If a system crash occurs during or immediately after the write call, upon recovery, the data in the log file can be used to reconstruct the write. This guarantees that each 8-KB chunk of a write either is completely written to disk or is not written to disk.

For example, if atomic write data logging is enabled and a crash occurs during a write that consists of two 8-KB chunks of data, the write can have three possible states: none of the data is written, 8 KB of the data is written, or 16 KB of data is written.

Atomic write data logging may degrade AdvFS write performance because of the extra write to the transaction log file. In addition, a file that has atomic write data logging enabled cannot be memory mapped by using the `mmap` system call.

A file cannot have both forced synchronous writes enabled (see Section 9.3.4.10) and atomic write data logging enabled. However, you can enable atomic write data logging on a file and also open the file with an `O_SYNC` option. This ensures that the write is synchronous, but also prevents partial writes if a crash occurs.

Use the `chfile` command to determine if forced synchronous writes or atomic write data logging is enabled. Use the `chfile -L off` command to disable atomic write data logging (the default).

To enable atomic write data logging on AdvFS files that are NFS mounted, the NFS property list daemon, `proplistd`, must be running on the NFS client and the fileset must be mounted on the client by using the `mount` command's `proplist` option.

If atomic write data logging is enabled and you are writing to a file that has been NFS mounted, the offset into the file must be on an 8-KB page boundary, because NFS performs I/O on 8-KB page boundaries.

You can also activate and deactivate atomic data logging by using the `fcntl` system call. In addition, both the `chfile` and `fcntl` can be used on an NFS client to activate or deactivate this feature on a file that resides on the NFS server.

9.3.4.12 Moving the Transaction Log

Make sure that the AdvFS transaction log resides on an uncongested disk and bus or performance may be degraded.

Use the `showfdmn` command to determine the current location of the transaction log. In the `showfdmn` command display, the letter `L` displays next to the volume that contains the log.

If the transaction log becomes a bottleneck, use the `switchlog` command to relocate the transaction log of the specified file domain to a faster or less congested volume in the same domain. See `switchlog`(8) and `showfdmn`(8) for more information.

In addition, you can divide the file domain into several smaller file domains. This will cause each domain's transaction log to handle transactions for fewer filesets.

9.3.4.13 Balancing a Multivolume File Domain

If the files in a multivolume domain are not evenly distributed, performance may be degraded. Use the `balance` utility to distribute the percentage of used space evenly between volumes in a multivolume file domain. This improves performance and evens the distribution of future file allocations. Files are moved from one volume to another until the percentage of used space on each volume in the domain is as equal as possible.

The `balance` utility does not affect data availability and is transparent to users and applications. If possible, use the `defragment` utility before you balance files.

The `balance` utility does not generally split files. Therefore, file domains with very large files may not balance as evenly as file domains with smaller files. See `balance`(8) for more information.

To determine if you need to balance your files across volumes, use the `showfdmn` command to display information about the volumes in a domain. The `% used` field shows the percentage of volume space that is currently allocated to files or metadata (fileset data structure). See `showfdmn`(8) for more information.

9.3.4.14 Migrating Files Within a File Domain

Performance may degrade if too many frequently accessed or large files reside on the same volume in a multivolume file domain. You can improve I/O performance by altering the way files are mapped on the disk.

Use the `migrate` utility to move frequently accessed or large files to different volumes in the file domain. You can specify the volume where a file is to be moved, or allow the system to pick the best space in the file domain. You can migrate either an entire file or specific pages to a different volume.

Using the `balance` utility after migrating files may cause the files to move to a different volume. See `balance`(8) for more information.

In addition, a file that is migrated is defragmented at the same time, if possible. Defragmentation makes the file more contiguous, which improves performance. Therefore, you can use the `migrate` command to defragment selected files. See `migrate`(8) for more information. Use the `iostat` command to identify which disks are being heavily used. See Section 8.2.1 for information.

9.4 Managing UNIX File System Performance

The UNIX file system (UFS) can provide you with high-performance file system operations, especially for critical applications. For example, UFS file reads from striped disks can be 50 percent faster than if you are using AdvFS, and will consume only 20 percent of the CPU power that AdvFS requires.

However, unlike AdvFS, the UFS file system directory hierarchy is bound tightly to a single disk partition.

The following sections describe how to perform these tasks:

- Use the UFS guidelines to set up a high-performance configuration (Section 9.4.1)
- Obtain information about UFS performance (Section 9.4.2)
- Tune UFS in order to improve performance (Section 9.4.3)

9.4.1 UFS Configuration Guidelines

There are a number of parameters that can improve the UFS performance. You can set all of the parameters when you use the `newfs` command to create a file system. For existing file systems, you can modify some parameters by using the `tunefs` command. See `newfs`(8) and `tunefs`(8) for more information.

Table 9–6 describes UFS configuration guidelines and performance benefits as well as tradeoffs.

Table 9–6: UFS Configuration Guidelines

Action	Performance Benefit	Tradeoff
Make the file system fragment size equal to the block size (Section 9.4.1.1)	Improves performance for large files	Wastes disk space for small files
Use the default file system fragment size of 1 KB (Section 9.4.1.1)	Uses disk space efficiently	Increases the overhead for large files
Reduce the density of inodes on a file system (Section 9.4.1.2)	Frees disk space for file data and improves large file performance	Reduces the number of files that can be created on the file system
Allocate blocks sequentially (Section 9.4.1.3)	Improves performance for disks that do not have a read ahead cache	Reduces the total available disk space
Increase the number of blocks combined for a cluster (Section 9.4.1.4)	May decrease number of disk I/O operations	May require more memory to buffer data

Table 9–6: UFS Configuration Guidelines (cont.)

Action	Performance Benefit	Tradeoff
Use a Memory File System (MFS) (Section 9.4.1.5)	Improves I/O performance	Does not ensure data integrity because of cache volatility
Use disk quotas (Section 9.4.1.6)	Controls disk space utilization	UFS quotas may result in a slight increase in reboot time

The following sections describe the UFS configuration guidelines in detail.

9.4.1.1 Modifying the UFS Fragment Size

The UFS file system block size is 8 KB. The default fragment size is 1 KB. You can use the `newfs` command to modify the fragment size so that it is 25, 50, 75, or 100 percent of the block size.

Although the default fragment size uses disk space efficiently, it increases the overhead for large files. If the average file in a file system is larger than 16 KB but less than 96 KB, you may be able to improve disk access time and decrease system overhead by making the file system fragment size equal to the default block size (8 KB).

See `newfs`(8) for more information.

9.4.1.2 Reducing the Density of inodes

An inode describes an individual file in the file system. The maximum number of files in a file system depends on the number of inodes and the size of the file system. The system creates an inode for each 4 KB (4096 bytes) of data space in a file system.

If a file system will contain many large files and you are sure that you will not create a file for each 4 KB of space, you can reduce the density of inodes on the file system. This will free disk space for file data, but will reduce the number of files that can be created.

To do this, use the `newfs -i` command to specify the amount of data space allocated for each inode. See `newfs`(8) for more information.

9.4.1.3 Allocating Blocks Sequentially

The UFS `rotdelay` parameter specifies the time, in milliseconds, to service a transfer completion interrupt and initiate a new transfer on the same disk. You can set the `rotdelay` parameter to 0 (the default) to allocate blocks sequentially. This is useful for disks that do not have a read-ahead cache. However, it will reduce the total amount of available disk space.

Use either the `tunefs` command or the `newfs` command to modify the `rotdelay` value. See `newfs`(8) and `tunefs`(8) for more information.

9.4.1.4 Increasing the Number of Blocks Combined for a Cluster

The value of the UFS `maxcontig` parameter specifies the number of blocks that can be combined into a single cluster (or file-block group). The default value of `maxcontig` is 8. The file system attempts I/O operations in a size that is determined by the value of `maxcontig` multiplied by the block size (8 KB).

Device drivers that can chain several buffers together in a single transfer should use a `maxcontig` value that is equal to the maximum chain length. This may reduce the number of disk I/O operations. However, more memory will be needed to buffer data.

Use the `tunefs` command or the `newfs` command to change the value of `maxcontig`. See `newfs`(8) and `tunefs`(8) for more information.

9.4.1.5 Using a Memory File System

Memory File System (MFS) is a UFS file system that resides only in memory. No permanent data or file structures are written to disk. An MFS file system can improve read/write performance, but it is a volatile cache. The contents of an MFS file system are lost after a reboot, unmount operation, or power failure.

Because no date is written to disk, an MFS file system is a very fast file system and can be used to store temporary files or read-only files that are loaded into it after it is created. For example, if you are performing a software build that would have to be restarted if it failed, use an MFS file system to cache the temporary files that are created during the build and reduce the build time.

See `mfs`(8) for information.

9.4.1.6 Using UFS Disk Quotas

You can specify UFS file system limits for user accounts and for groups by setting up file system quotas, also known as disk quotas. You can apply quotas to file systems to establish a limit on the number of blocks and inodes (or files) that a user account or a group of users can allocate. You can set a separate quota for each user or group of users on each file system.

You may want to set quotas on file systems that contain home directories, because the sizes of these file systems can increase more significantly than other file systems. Do not set quotas on the `/tmp` file system.

Note that, unlike AdvFS quotas, UFS quotas may cause a slight increase in reboot time. For information about AdvFS quotas, see Section 9.3.2.5.

For information about UFS quotas, see the *System Administration* manual.

9.4.2 Gathering UFS Information

Table 9–7 describes the tools you can use to obtain information about UFS.

Table 9–7: UFS Monitoring Tools

Name	Use	Description
`dumpfs`	Displays UFS information (Section 9.4.2.1)	Displays detailed information about a UFS file system or a special device, including information about the file system fragment size, the percentage of free space, super blocks, and the cylinder groups.
`(dbx) print ufs_clusterstats`	Reports UFS clustering statistics (Section 9.4.2.2)	Reports statistics on how the system is performing cluster read and write transfers.
`(dbx) print bio_stats`	Reports UFS metadata buffer cache statistics (Section 9.4.2.3)	Reports statistics on the metadata buffer cache, including superblocks, inodes, indirect blocks, directory blocks, and cylinder group summaries.
`fsx`	Exercises file systems	Exercises UFS and AdvFS file systems by creating, opening, writing, reading, validating, closing, and unlinking a test file. Errors are written to a log file. See `fsx`(8) for more information.

The following sections describe some of these commands in detail.

9.4.2.1 Displaying UFS Information by Using the dumpfs Command

The `dumpfs` command displays UFS information, including super block and cylinder group information, for a specified file system. Use this command to obtain information about the file system fragment size and the minimum free space percentage.

The following example shows part of the output of the `dumpfs` command:

```
# /usr/sbin/dumpfs /devices/disk/rr3zg | more
magic   11954   format  dynamic time    Tue Sep 14 15:46:52 1998
nbfree  21490   ndir    9       nifree  99541   nffree  60
ncg     65      ncyl    1027    size    409600  blocks  396062
bsize   8192    shift   13      mask    0xffffe000
fsize   1024    shift   10      mask    0xfffffc00
frag    8       shift   3       fsbtodb 1
cpg     16      bpg     798     fpg     6384    ipg     1536
minfree 10%     optim   time    maxcontig 8     maxbpg  2048
rotdelay 0ms    headswitch 0us  trackseek 0us   rps     60
```

The information contained in the first lines are relevant for tuning. Of specific interest are the following fields:

- `bsize`—The block size of the file system in bytes (8 KB).
- `fsize`—The fragment size of the file system (in bytes). For the optimum I/O performance, you can modify the fragment size.
- `minfree`—The percentage of space held back from normal users; the minimum free space threshold.
- `maxcontig`—The maximum number of contiguous blocks that will be laid out before forcing a rotational delay; that is, the number of blocks that are combined into a single read request.
- `maxbpg`—The maximum number of blocks any single file can allocate out of a cylinder group before it is forced to begin allocating blocks from another cylinder group. A large value for `maxbpg` can improve performance for large files.
- `rotdelay`—The expected time (in milliseconds) to service a transfer completion interrupt and initiate a new transfer on the same disk. It is used to decide how much rotational spacing to place between successive blocks in a file. If `rotdelay` is zero, then blocks are allocated contiguously.

See Section 9.4.3 for information about tuning UFS.

9.4.2.2 Monitoring UFS Clustering by Using the dbx Debugger

To determine how efficiently the system is performing cluster read and write transfers, use the `dbx print` command to examine the `ufs_clusterstats` data structure.

The following example shows a system that is not clustering efficiently:

```
# /usr/ucb/dbx -k /vmunix /dev/mem
(dbx) print ufs_clusterstats
struct {
    full_cluster_transfers = 3130
    part_cluster_transfers = 9786
    non_cluster_transfers = 16833
    sum_cluster_transfers = {
        [0] 0
        [1] 24644
        [2] 1128
        [3] 463
        [4] 202
        [5] 55
        [6] 117
        [7] 36
        [8] 123
        [9] 0
    }
}
(dbx)
```

The preceding example shows 24644 single-block transfers and no 9-block transfers. A single block is 8 KB. The trend of the data shown in the example is the reverse of what you want to see. It shows a large number of single-block transfers and a declining number of multiblock (1–9) transfers. However, if the files are all small, this may be the best blocking that you can achieve.

You can examine the cluster reads and writes separately with the `ufs_clusterstats_read` and `ufs_clusterstats_write` data structures.

See Section 9.4.3 for information on tuning UFS.

9.4.2.3 Checking the Metadata Buffer Cache by Using the dbx Debugger

The metadata buffer cache contains UFS file metadata—superblocks, inodes, indirect blocks, directory blocks, and cylinder group summaries. To check the metadata buffer cache, use the `dbx print` command to examine the `bio_stats` data structure.

Consider the following example:

```
# /usr/ucb/dbx -k /vmunix /dev/mem
(dbx) print bio_stats
struct {
    getblk_hits = 4590388
    getblk_misses = 17569
    getblk_research = 0
    getblk_dupbuf = 0
    getnewbuf_calls = 17590
    getnewbuf_buflocked = 0
    vflushbuf_lockskips = 0
    mntflushbuf_misses = 0
    mntinvalbuf_misses = 0
    vinvalbuf_misses = 0
    allocbuf_buflocked = 0
    ufssync_misses = 0
}
(dbx)
```

If the miss rate is high, you may want to raise the value of the `bufcache` attribute. The number of block misses (`getblk_misses`) divided by the sum of block misses and block hits (`getblk_hits`) should not be more than 3 percent.

See Section 9.4.3.1 for information on how to tune the metadata buffer cache.

9.4.3 Tuning UFS

After you configure your UFS file systems, you may be able to improve UFS performance. To successfully improve performance, you must understand how your applications and users perform file system I/O, as described in Section 2.1.

Table 9–8 describes UFS tuning guidelines and performance benefits as well as tradeoffs. In addition, the recommendations described in Table 9–1 apply to UFS configurations.

Table 9–8: UFS Tuning Guidelines

Action	Performance Benefit	Tradeoff
Increase size of metadata buffer cache to more than 3 percent of main memory (Section 9.4.3.1)	Increases cache hit rate and improves UFS performance	Requires additional memory resources
Increase the size of the metadata hash chain table (Section 9.4.3.2)	Improves UFS lookup speed	Increases wired memory
Defragment the file system (Section 9.4.3.3)	Improves read and write performance	Requires down time
Delay flushing full write buffers to disk (Section 9.4.3.4)	Frees CPU cycles	May degrade real-time workload performance when buffers are flushed

Table 9–8: UFS Tuning Guidelines (cont.)

Action	Performance Benefit	Tradeoff
Increase number of blocks combined for read ahead (Section 9.4.3.5)	May reduce disk I/O operations	May require more memory to buffer data
Increase number of blocks combined for a cluster(Section 9.4.3.6)	May decrease disk I/O operations	Reduces available disk space
Increase the smooth sync caching threshhold for asynchronous UFS I/O requests (Section 9.4.3.7)	Improves performance of AdvFS asynchronous I/O	None
Increase the maximum number of UFS and MFS mounts (Section 9.4.3.8)	Allows more mounted file systems	Requires additional memory resources

The following sections describe how to tune UFS in detail.

9.4.3.1 Modifying the Size of the Metadata Buffer Cache

At boot time, the kernel wires a percentage of physical memory for the metadata buffer cache, which temporarily holds recently accessed UFS and CD-ROM File System (CDFS) metadata. The `vfs` subsystem attribute `bufcache` specifies the size of the metadata buffer cache as a percentage of physical memory. The default is 3 percent.

Usually, you do not have to increase the cache size. However, you may want to increase the size of the metadata buffer cache if you reuse data and have a high cache miss rate (low hit rate).

To determine whether to increase the size of the metadata buffer cache, use the `dbx print` command to examine the `bio_stats` data structure. The miss rate (block misses divided by the sum of the block misses and block hits) should not be more than 3 percent.

If you have a general-purpose timesharing system, do not increase the value of the `bufcache` attribute to more than 10 percent. If you have an NFS server that does not perform timesharing, do not increase the value of the `bufcache` attribute to more than 35 percent.

Allocating additional memory to the metadata buffer cache reduces the amount of memory available to processes and the UBC. See Section 6.1.2.1 for information about how memory is allocated to the metadata buffer cache.

See Section 4.4 for information about modifying kernel subsystem attributes.

9.4.3.2 Increasing the Size of the Metadata Hash Chain Table

The hash chain table for the metadata buffer cache stores the heads of the hashed buffer queues. Increasing the size of the hash chain table distributes the buffers, which makes average chain lengths short. This can improve lookup speeds. However, increasing the size of the hash chain table increases wired memory.

The `vfs` subsystem attribute `buffer-hash-size` specifies the size of the hash chain table, in table entries, for the metadata buffer cache. The minimum size is 16; the maximum size is 524287. The default value is 512.

You can modify the value of the `buffer-hash-size` attribute so that each hash chain has 3 or 4 buffers. To determine a value for the `buffer-hash-size` attribute, use the `dbx print` command to examine the value of the `nbuf` kernel variable, then divide the value by 3 or 4, and finally round the result to a power of 2. For example, if `nbuf` has a value of 360, dividing 360 by 3 gives you a value of 120. Based on this calculation, specify 128 (2 to the power of 7) as the value of the `buffer-hash-size` attribute.

See Section 4.4 for information about modifying kernel subsystem attributes.

9.4.3.3 Defragmenting a File System

When a file consists of noncontiguous file extents, the file is considered fragmented. A very fragmented file decreases UFS read and write performance, because it requires more I/O operations to access the file.

You can determine whether the files in a file system are fragmented by determining how effectively the system is clustering. You can do this by using the `dbx print` command to examine the `ufs_clusterstats`, `ufs_clusterstats_read`, and `ufs_clusterstats_write` data structures. See Section 9.4.2.2 for information.

UFS block clustering is usually efficient. If the numbers from the UFS clustering kernel structures show that clustering is not being particularly effective, the files in the file system may be very fragmented.

To defragment a UFS file system, follow these steps:

1. Back up the file system onto tape or another partition.
2. Create a new file system either on the same partition or a different partition.

3. Restore the file system.

See the *System Administration* manual for information about backing up and restoring data and creating UFS file systems.

9.4.3.4 Delaying Full Write Buffer Flushing

You can free CPU cycles by delaying flushing full write buffers to disk until the next `sync` call (or until the percentage of UBC dirty pages reaches the value of the `delay_wbuffers_percent` kernel variable). However, delaying write buffer flushing may adversely affect real-time workload performance, because the system will experience a heavy I/O load at sync time.

To delay full write buffer flushing, use the `dbx patch` command to set the value of the `delay_wbuffers` kernel variable to 1 (enabled). The default value of `delay_wbuffers` is 0 (disabled).

See Section 4.4.6 for information on using `dbx`.

9.4.3.5 Increasing the Number of Blocks Combined for Read Ahead

You can increase the number of blocks that are combined for a read-ahead operation.

To do this, use the `dbx patch` command to set the value of the `cluster_consec_init` kernel variable equal to the value of the `cluster_max_read_ahead` kernel variable (the default is 8), which specifies the maximum number of read-ahead clusters that the kernel can schedule.

In addition, you must make sure that cluster read operations are enabled on nonread-ahead and read-ahead blocks. To do this, use `dbx` to set the value of the `cluster_read_all` kernel variable to 1, which is the default value.

See Section 4.4.6 for information on using `dbx`.

9.4.3.6 Increasing the Number of Blocks Combined for a Cluster

The `cluster_maxcontig` kernel variable specifies the number of blocks that are combined into a single I/O operation. The default value is 8. Contiguous writes are done in a unit size that is determined by the file system block size (8 KB) multiplied by the value of the `cluster_maxcontig` parameter.

See Section 4.4.6 for information about using `dbx`.

9.4.3.7 Modifying UFS Smooth Sync Caching

Smooth sync functionality improves UFS asynchronous I/O performance by preventing I/O spikes caused by the update daemon and by increasing the UBC hit rate, which decreases the total number of disk operations. Smooth sync also helps to efficiently distribute I/O requests over the sync interval, which decreases the length of the disk queue and reduces the latency that results from waiting for a busy page to be freed. By default, smooth sync is enabled on your system.

UFS caches asynchronous I/O requests in the dirty block queue and in the UBC object dirty page list queue before they are handed to the device driver. With smooth sync enabled (the default), the `update` daemon will not flush the dirty page list and dirty page wired list buffers. Instead, asynchronous I/O requests remain in the queue for the amount of time specified by the value of the `vfs` attribute `smoothsync_age` (the default is 30 seconds). When a buffer ages sufficiently, it is moved to the device queue.

If smooth sync is disabled, every 30 seconds the `update` daemon flushes data from memory to disk, regardless of how long a buffer has been cached.

Smooth sync functionality is controlled by the `smoothsync_age` attribute. However, you do not specify a value for `smoothsync_age` in the `/etc/sysconfigtab` file. Instead, the `/etc/inittab` file is used to enable smooth sync when the system boots to multiuser mode and to disable smooth sync when the system goes from multiuser mode to single-user mode. This procedure is necessary to reflect the behavior of the `update` daemon, which operates only in multiuser mode.

To enable smooth sync, the following lines must be included in the `/etc/inittab` file and the time limit for caching buffers in the smooth sync queue must be specified:

```
smsync:23:wait:/sbin/sysconfig -r vfs smoothsync_age=30 > /dev/null 2>&1
smsyncS:Ss:wait:/sbin/sysconfig -r vfs smoothsync_age=0 > /dev/null 2>&1
```

Thirty seconds is the default smooth sync queue threshhold. If you increase this value, you may improve the chance of a buffer cache hit by retaining buffers on the smooth sync queue for a longer period of time. Consequently, decreasing the value of the `smoothsync_age` attribute will speed the flushing of buffers.

To disable smooth sync, specify a value of 0 (zero) for the `smoothsync_age` attribute.

See Section 4.4 for information about modifying kernel subsystem attributes.

9.4.3.8 Increasing the Number of UFS and MFS Mounts

Mount structures are dynamically allocated when a mount request is made and subsequently deallocated when an unmount request is made. The `vfs` subsystem attribute `max-ufs-mounts` specifies the maximum number of UFS and MFS mounts on the system.

You can increase the value of the `max-ufs-mounts` attribute if your system will have more than the default limit of 1000 mounts. However, increasing the maximum number of UFS and MFS mounts requires memory resources for the additional mounts.

See Section 4.4 for information about modifying kernel subsystem attributes.

9.5 Managing NFS Performance

The Network File System (NFS) shares the UBC with the virtual memory subsystem and local file systems. NFS can put an extreme load on the network. Poor NFS performance is almost always a problem with the network infrastructure. Look for high counts of retransmitted messages on the NFS clients, network I/O errors, and routers that cannot maintain the load.

Lost packets on the network can severely degrade NFS performance. Lost packets can be caused by a congested server, the corruption of packets during transmission (which can be caused by bad electrical connections, noisy environments, or noisy Ethernet interfaces), and routers that abandon forwarding attempts too quickly.

You can monitor NFS by using the `nfsstat` and other commands. When evaluating NFS performance, remember that NFS does not perform well if any file-locking mechanisms are in use on an NFS file. The locks prevent the file from being cached on the client. See `nfsstat`(8) for more information.

The following sections describe how to perform the following tasks:

- Gather NFS performance information (Section 9.5.1)
- Improving NFS performance (Section 9.5.2)

9.5.1 Gathering NFS Information

Table 9–9 describes the commands you can use to obtain information about NFS operations.

Table 9–9: NFS Monitoring Tools

Name	Use	Description
`nfsstat`	Displays network and NFS statistics (Section 9.5.1.1)	Displays NFS and RPC statistics for clients and servers, including the number of packets that had to be retransmitted (`retrans`) and the number of times a reply transaction ID did not match the request transaction ID (`badxid`).
`nfswatch`	Monitors an NFS server	Monitors all incoming network traffic to an NFS server and divides it into several categories, including NFS reads and writes, NIS requests, and RPC authorizations. Your kernel must be configured with the `packetfilter` option to use the command. See `nfswatch`(8) and `packetfilter`(7) for more information.
`ps axlmp`	Displays information about idle threads (Section 9.5.1.2)	Displays information about idle threads on a client system.
`(dbx) print nfs_sv_active_hist`	Displays active NFS server threads (Section 4.4.6)	Displays a histogram of the number of active NFS server threads.
`(dbx) print nchstats`	Displays the hit rate (Section 9.1.2)	Displays the namei cache hit rate.
`(dbx) print bio_stats`	Displays metadata buffer cache information (Section 9.4.2.3)	Reports statistics on the metadata buffer cache hit rate.
`(dbx) print vm_perfsum`	Reports UBC statistics (Section 6.3.5)	Reports the UBC hit rate.

The following sections describe how to use some of these tools in detail.

9.5.1.1 Displaying NFS Information by Using the nfsstat Command

The `nfsstat` command displays statistical information about NFS and Remote Procedure Call (RPC) interfaces in the kernel. You can also use this command to reinitialize the statistics.

An example of the `nfsstat` command is as follows:

```
# /usr/ucb/nfsstat

Server rpc:
calls     badcalls  nullrecv   badlen   xdrcall
38903     0         0          0        0

Server nfs:
calls     badcalls
38903     0

Server nfs V2:
null      getattr   setattr    root     lookup     readlink   read
5  0%     3345  8%  61  0%     0  0%    5902 15%   250  0%    1497  3%
wrcache   write     create     remove   rename     link       symlink
0  0%     1400  3%  549  1%    1049  2% 352  0%    250  0%    250  0%
mkdir     rmdir     readdir    statfs
171  0%   172  0%   689  1%    1751  4%

Server nfs V3:
null      getattr   setattr    lookup    access    readlink   read
0  0%     1333  3%  1019  2%   5196 13%  238  0%   400  1%    2816  7%
write     create    mkdir      symlink   mknod     remove     rmdir
2560  6%  752  1%   140  0%    400  1%   0  0%     1352  3%   140  0%
rename    link      readdir    readdir+  fsstat    fsinfo     pathconf
200  0%   200  0%   936  2%    0  0%     3504  9%  3  0%      0  0%
commit
21  0%

Client rpc:
calls     badcalls  retrans    badxid    timeout   wait       newcred
27989     1         0          0         1         0          0
badverfs  timers
0         4

Client nfs:
calls     badcalls  nclget     nclsleep
27988     0         27988      0

Client nfs V2:
null      getattr   setattr    root      lookup    readlink   read
0  0%     3414 12%  61  0%     0  0%     5973 21%  257  0%    1503  5%
wrcache   write     create     remove    rename    link       symlink
0  0%     1400  5%  549  1%    1049  3%  352  1%   250  0%    250  0%
mkdir     rmdir     readdir    statfs
171  0%   171  0%   713  2%    1756  6%

Client nfs V3:
null      getattr   setattr    lookup    access    readlink   read
0  0%     666  2%   9  0%      2598  9%  137  0%   200  0%    1408  5%
write     create    mkdir      symlink   mknod     remove     rmdir
1280  4%  376  1%   70  0%     200  0%   0  0%     676  2%    70  0%
rename    link      readdir    readdir+  fsstat    fsinfo     pathconf
100  0%   100  0%   468  1%    0  0%     1750  6%  1  0%      0  0%
commit
```

```
10  0%
#
```

The ratio of timeouts to calls (which should not exceed 1 percent) is the most important thing to look for in the NFS statistics. A timeout-to-call ratio greater than 1 percent can have a significant negative impact on performance. See Chapter 10 for information on how to tune your system to avoid timeouts.

Use the `nfsstat -s -i 10` command to display NFS and RPC information at ten-second intervals.

If you are attempting to monitor an experimental situation with `nfsstat`, reset the NFS counters to 0 before you begin the experiment. Use the `nfsstat -z` command to clear the counters.

See `nfsstat`(8) for more information about command options and output.

9.5.1.2 Displaying Idle Thread Information by Using the ps Command

On a client system, the `nfsiod` daemons spawn several I/O threads to service asynchronous I/O requests to the server. The I/O threads improve the performance of both NFS reads and writes. The optimum number of I/O threads depends on many variables, such as how quickly the client will be writing, how many files will be accessed simultaneously, and the characteristics of the NFS server. For most clients, seven threads are sufficient.

The following example uses the `ps axlmp` command to display idle I/O threads on a client system:

```
#
/usr/ucb/ps axlmp 0 | grep nfs

 0  42   0           nfsiod_  S                0:00.52
 0  42   0           nfsiod_  S                0:01.18
 0  42   0           nfsiod_  S                0:00.36
 0  44   0           nfsiod_  S                0:00.87
 0  42   0           nfsiod_  S                0:00.52
 0  42   0           nfsiod_  S                0:00.45
 0  42   0           nfsiod_  S                0:00.74

#
```

The previous output shows a sufficient number of sleeping threads and 42 server threads that were started by `nfsd`, where `nfsiod_` has been replaced by `nfs_tcp` or `nfs_udp`.

If your output shows that few threads are sleeping, you may be able to improve NFS performance by increasing the number of threads. See Section 9.5.2.2, Section 9.5.2.3, `nfsiod`(8), and `nfsd`(8) for more information.

9.5.2 Improving NFS Performance

Improving performance on a system that is used only for serving NFS differs from tuning a system that is used for general timesharing, because an NFS server runs only a few small user-level programs, which consume few system resources. There is minimal paging and swapping activity, so memory resources should be focused on caching file system data.

File system tuning is important for NFS because processing NFS requests consumes the majority of CPU and wall clock time. Ideally, the UBC hit rate should be high. Increasing the UBC hit rate can require additional memory or a reduction in the size of other file system caches. In general, file system tuning will improve the performance of I/O-intensive user applications.

In addition, a vnode must exist to keep file data in the UBC. If you are using AdvFS, an access structure is also required to keep file data in the UBC.

If you are running NFS over TCP, tuning TCP may improve performance if there are many active clients. However, if you are running NFS over UDP, no network tuning is needed. See Section 10.2 for more information.

Table 9–10 lists NFS tuning and performance-improvement guidelines and the benefits as well as tradeoffs.

Table 9–10: NFS Performance Guidelines

Action	Performance Benefit	Tradeoff
Set the value of the `maxusers` attribute to the number of server NFS operations that are expected to occur each second (Section 5.1)	Provides the appropriate level of system resources	Consumes memory
Increase the size of the namei cache (Section 9.2.1)	Improves file system performance	Consumes memory
Increase the number of AdvFS access structures, if you are using AdvFS (Section 9.3.4.3)	Improves AdvFS performance	Consumes memory
Increase the size of the metadata buffer cache, if you are using UFS (Section 9.4.3.1)	Improves UFS performance	Consumes wired memory
Use Prestoserve (Section 9.5.2.1)	Improves synchronous write performance for NFS servers	Cost
Configure the appropriate number of threads on an NFS server (Section 9.5.2.2)	Enables efficient I/O blocking operations	None

Table 9–10: NFS Performance Guidelines (cont.)

Action	Performance Benefit	Tradeoff
Configure the appropriate number of threads on the client system (Section 9.5.2.3)	Enables efficient I/O blocking operations	None
Modify cache timeout limits on the client system (Section 9.5.2.4)	May improve network performance for read-only file systems and enable clients to quickly detect changes	Increases network traffic to server
Decrease network timeouts on the client system (Section 9.5.2.5)	May improve performance for slow or congested networks	Reduces theoretical performance
Use NFS protocol Version 3 on the client system (Section 9.5.2.6)	Improves network performance	Decreases the performance benefit of Prestoserve

The following sections describe these guidelines in detail.

9.5.2.1 Using Prestoserve to Improve NFS Server Performance

You can improve NFS performance by installing Prestoserve on the server. Prestoserve greatly improves synchronous write performance for servers that are using NFS Version 2. Prestoserve enables an NFS Version 2 server to write client data to a nonvolatile (battery-backed) cache, instead of writing the data to disk.

Prestoserve may improve write performance for NFS Version 3 servers, but not as much as with NFS Version 2, because NFS Version 3 servers can reliably write data to volatile storage without risking loss of data in the event of failure. NFS Version 3 clients can detect server failures and resend any write data that the server may have lost in volatile storage.

See the *Guide to Prestoserve* for more information.

9.5.2.2 Configuring Server Threads

The `nfsd` daemon runs on NFS servers to service NFS requests from client machines. The daemon spawns a number of server threads that process NFS requests from client machines. At least one server thread must be running for a machine to operate as a server. The number of threads determines the number of parallel operations and must be a multiple of 8.

For good performance on frequently used NFS servers, configure either 16 or 32 threads, which provides the most efficient blocking for I/O operations. See `nfsd`(8) for more information.

9.5.2.3 Configuring Client Threads

Client systems use the `nfsiod` daemon to service asynchronous I/O operations such as buffer cache read-ahead and delayed write operations. The `nfsiod` daemon spawns several IO threads to service asynchronous I/O requests to its server. The I/O threads improve performance of both NFS reads and writes.

The optimal number of I/O threads to run depends on many variables, such as how quickly the client is writing data, how many files will be accessed simultaneously, and the behavior of the NFS server. The number of threads must be a multiple of 8 minus 1 (for example, 7 or 15 is optimal).

NFS servers attempt to gather writes into complete UFS clusters before initiating I/O, and the number of threads (plus 1) is the number of writes that a client can have outstanding at any one time. Having exactly 7 or 15 threads produces the most efficient blocking for I/O operations. If write gathering is enabled, and the client does not have any threads, you may experience a performance degradation. To disable write gathering, use the `dbx patch` command to set the `nfs_write_gather` kernel variable to zero. See Section 4.4.6 for information.

Use the `ps axlmp 0 | grep nfs` command to display idle I/O threads on the client. If few threads are sleeping, you may be able to improve NFS performance by increasing the number of threads. See `nfsiod`(8) for more information.

9.5.2.4 Modifying Cache Timeout Limits

For read-only file systems and slow network links, performance may be improved by changing the cache timeout limits on NFS client systems. These timeouts affect how quickly you see updates to a file or directory that has been modified by another host. If you are not sharing files with users on other hosts, including the server system, increasing these values will give you slightly better performance and will reduce the amount of network traffic that you generate.

See `mount`(8) and the descriptions of the `acregmin`, `acregmax`, `acdirmin`, `acdirmax`, `actimeo` options for more information.

9.5.2.5 Decreasing Network Timeouts

NFS does not perform well if it is used over slow network links, congested networks, or wide area networks (WANs). In particular, network timeouts on client systems can severely degrade NFS performance. This condition can be identified by using the `nfsstat` command and determining the ratio of timeouts to calls. If timeouts are more than 1 percent of total calls, NFS performance may be severely degraded. See Section 9.5.1.1 for sample `nfsstat` output of timeout and call statistics and `nfsstat`(8) for more information.

You can also use the `netstat -s` command to verify the existence of a timeout problem. A nonzero value in the `fragments dropped after timeout` field in the `ip` section of the `netstat` output may indicate that the problem exists. See Section 10.1.1 for sample `netstat` command output.

If fragment drops are a problem, on a client system, use the `mount` command with the `-rsize=1024` and `-wsize=1024` options to set the size of the NFS read and write buffers to 1 KB.

9.5.2.6 Using NFS Protocol Version 3

NFS protocol Version 3 provides NFS client-side asynchronous write support, which improves the cache consistency protocol and requires less network load than Version 2. These performance improvements slightly decrease the performance benefit that Prestoserve provided for NFS Version 2. However, with Protocol Version 3, Prestoserve still speeds file creation and deletion.

10

Managing Network Performance

This chapter describes how to manage Tru64 UNIX network subsystem performance. The following sections describe these tasks:

- Monitor the network subsystem (Section 10.1)
- Tune the network subsystem (Section 10.2)

10.1 Gathering Network Information

Table 10–1 describes the commands you can use to obtain information about network operations.

Table 10–1: Network Monitoring Tools

Name	Use	Description
`netstat`	Displays network statistics (Section 10.1.1)	Displays a list of active sockets for each protocol, information about network routes, and cumulative statistics for network interfaces, including the number of incoming and outgoing packets and packet collisions. Also, displays information about memory used for network operations.
`traceroute`	Displays the packet route to a network host	Tracks the route network packets follow from gateway to gateway. See `traceroute`(8) for more information.
`ping`	Determines if a system can be reached on the network	Sends an Internet Control Message Protocol (ICMP) echo request to a host in order to determine if a host is running and reachable, and to determine if an IP router is reachable. Enables you to isolate network problems, such as direct and indirect routing problems. See `ping`(8) for more information.

Table 10–1: Network Monitoring Tools (cont.)

Name	Use	Description
`sobacklog_hiwat` attribute	Reports the maximum number of pending requests to any server socket (Section 10.1.2)	Allows you to display the maximum number of pending requests to any server socket in the system.
`sobacklog_drops` attribute	Reports the number of backlog drops that exceed a socket backlog limit (Section 10.1.2)	Allows you to display the number of times the system dropped a received SYN packet, because the number of queued SYN_RCVD connections for a socket equaled the socket backlog limit.
`somaxconn_drops` attribute	Reports the number of drops that exceed the value of the `somaxconn` attribute (Section 10.1.2)	Allows you to display the number of times the system dropped a received SYN packet because the number of queued SYN_RCVD connections for a socket equaled the upper limit on the backlog length (`somaxconn` attribute).
`tcpdump`	Monitors network interface packets	Monitors and displays packet headers on a network interface. You can specify the interface on which to listen, the direction of the packet transfer, or the type of protocol traffic to display. The `tcpdump` command allows you to monitor the network traffic associated with a particular network service and to identify the source of a packet. It lets you determine whether requests are being received or acknowledged, or to determine the source of network requests, in the case of slow network performance. Your kernel must be configured with the `packetfilter` option to use the command. See `tcpdump`(8) and `packetfilter`(7) for more information.

The following sections describe some of these commands in detail.

10.1.1 Monitoring Network Statistics by Using the netstat Command

To check network statistics, use the `netstat` command. Some problems to look for are as follows:

- If the `netstat -i` command shows excessive amounts of input errors (`Ierrs`), output errors (`Oerrs`), or collisions (`Coll`), this may indicate a network problem; for example, cables are not connected properly or the Ethernet is saturated.
- Use the `netstat -is` command to check for network device driver errors.
- Use the `netstat -m` command to determine if the network is using an excessive amount of memory in proportion to the total amount of memory installed in the system.

  If the `netstat -m` command shows several requests for memory delayed or denied, this means that your system was temporarily short of physical memory.
- Each socket results in a network connection. If the system allocates an excessive number of sockets, use the `netstat -an` command to determine the state of your existing network connections.

  An example of the `netstat -an` command is as follows:

  ```
  # /usr/sbin/netstat -an | grep tcp | awk '{print $6}' | sort | uniq -c
        1 CLOSE_WAIT
       58 ESTABLISHED
        2 FIN_WAIT_1
        3 FIN_WAIT_2
       17 LISTEN
        1 SYN_RCVD
    15749 TIME_WAIT
  ```

  For Internet servers, the majority of connections usually are in a `TIME_WAIT` state. Note that there are almost 16,000 sockets being used, which requires 16 MB of memory.
- Use the `netstat -p ip` command to check for bad checksums, length problems, excessive redirects, and packets lost because of resource problems.
- Use the `netstat -p tcp` command to check for retransmissions, out of order packets, and bad checksums.
- Use the `netstat -p udp` command to look for bad checksums and full sockets.
- Use the `netstat -rs` to obtain routing statistics.

Most of the information provided by `netstat` is used to diagnose network hardware or software failures, not to analyze tuning opportunities. See the *Network Administration* manual for more information on how to diagnose failures.

The following example shows the output produced by the `netstat -i` command:

```
# /usr/sbin/netstat -i
Name  Mtu   Network     Address          Ipkts Ierrs    Opkts Oerrs  Coll
ln0   1500  DLI         none            133194     2    23632     4  4881
ln0   1500  <Link>                      133194     2    23632     4  4881
ln0   1500  red-net     node1           133194     2    23632     4  4881
sl0*  296   <Link>                           0     0        0     0     0
sl1*  296   <Link>                           0     0        0     0     0
lo0   1536  <Link>                         580     0      580     0     0
lo0   1536  loop        localhost          580     0      580     0     0
```

Use the following `netstat` command to determine the causes of the input (`Ierrs`) and output (`Oerrs`) shown in the preceding example:

```
# /usr/sbin/netstat -is

ln0 Ethernet counters at Fri Jan 14 16:57:36 1998

        4112 seconds since last zeroed
    30307093 bytes received
     3722308 bytes sent
      133245 data blocks received
       23643 data blocks sent
    14956647 multicast bytes received
      102675 multicast blocks received
       18066 multicast bytes sent
         309 multicast blocks sent
        3446 blocks sent, initially deferred
        1130 blocks sent, single collision
        1876 blocks sent, multiple collisions
           4 send failures, reasons include:
                Excessive collisions
           0 collision detect check failure
           2 receive failures, reasons include:
                Block check error
                Framing Error
           0 unrecognized frame destination
           0 data overruns
           0 system buffer unavailable
           0 user buffer unavailable
```

The `netstat -s` command displays the following statistics for each protocol:

```
# /usr/sbin/netstat -s
ip:
        67673 total packets received
        0 bad header checksums
        0 with size smaller than minimum
        0 with data size < data length
        0 with header length < data size
        0 with data length < header length
        8616 fragments received
        0 fragments dropped (dup or out of space)
        5 fragments dropped after timeout
        0 packets forwarded
        8 packets not forwardable
        0 redirects sent
icmp:
        27 calls to icmp_error
        0 errors not generated  old message was icmp
        Output histogram:
                echo reply: 8
                destination unreachable: 27
        0 messages with bad code fields
        0 messages < minimum length
        0 bad checksums
        0 messages with bad length
        Input histogram:
                echo reply: 1
                destination unreachable: 4
                echo: 8
        8 message responses generated
igmp:
        365 messages received
        0 messages received with too few bytes
        0 messages received with bad checksum
        365 membership queries received
        0 membership queries received with invalid field(s)
        0 membership reports received
        0 membership reports received with invalid field(s)
        0 membership reports received for groups to which we belong
        0 membership reports sent
tcp:
        11219 packets sent
                7265 data packets (139886 bytes)
                4 data packets (15 bytes) retransmitted
                3353 ack-only packets (2842 delayed)
                0 URG only packets
                14 window probe packets
                526 window update packets
                57 control packets
        12158 packets received
                7206 acks (for 139930 bytes)
                32 duplicate acks
                0 acks for unsent data
                8815 packets (1612505 bytes) received in-sequence
                432 completely duplicate packets (435 bytes)
                0 packets with some dup. data (0 bytes duped)
                14 out-of-order packets (0 bytes)
                1 packet (0 bytes) of data after window
                0 window probes
                1 window update packet
                5 packets received after close
                0 discarded for bad checksums
                0 discarded for bad header offset fields
                0 discarded because packet too short
        19 connection requests
        25 connection accepts
        44 connections established (including accepts)
        47 connections closed (including 0 drops)
        3 embryonic connections dropped
        7217 segments updated rtt (of 7222 attempts)
        4 retransmit timeouts
                0 connections dropped by rexmit timeout
        0 persist timeouts
        0 keepalive timeouts
                0 keepalive probes sent
                0 connections dropped by keepalive
```

```
udp:
        12003 packets sent
        48193 packets received
        0 incomplete headers
        0 bad data length fields
        0 bad checksums
        0 full sockets
        12943 for no port (12916 broadcasts, 0 multicasts)
```

See `netstat`(1) for information about the output produced by the various command options.

10.1.2 Checking Socket Listen Queue Statistics by Using the sysconfig Command

You can determine whether you need to increase the socket listen queue limit by using the `sysconfig -q socket` command to display the values of the following attributes:

- `sobacklog_hiwat`

  Allows you to monitor the maximum number of pending requests to any server socket in the system. The initial value is zero.

- `sobacklog_drops`

  Allows you to monitor the number of times the system dropped a received SYN packet because the number of queued SYN_RCVD connections for a socket equaled the socket backlog limit. The initial value is zero.

- `somaxconn_drops`

  Allows you to monitor the number of times the system dropped a received SYN packet because the number of queued SYN_RCVD connections for the socket equaled the upper limit on the backlog length (`somaxconn` attribute). The initial value is zero.

It is recommended that the value of the `sominconn` attribute equal the value of the `somaxconn` attribute. If so, the value of `somaxconn_drops` will have the same value as `sobacklog_drops`.

However, if the value of the `sominconn` attribute is 0 (the default), and if one or more server applications uses an inadequate value for the backlog argument to its `listen` system call, the value of `sobacklog_drops` may increase at a rate that is faster than the rate at which the `somaxconn_drops` counter increases. If this occurs, you may want to increase the value of the `sominconn` attribute.

See Section 10.2.3 for information on tuning socket listen queue limits.

10.2 Tuning the Network Subsystem

Most resources used by the network subsystem are allocated and adjusted dynamically; however, there are some tuning recommendations that you can use to improve performance, particularly with systems that are Internet servers.

Network performance is affected when the supply of resources is unable to keep up with the demand for resources. The following two conditions can cause this congestion to occur:

- A problem with one or more components of the network (hardware or software)
- A workload (network traffic) that consistently exceeds the capacity of the available resources even though everything is operating correctly

Neither of these problems are network tuning issues. In the case of a problem on the network, you must isolate and eliminate the problem. In the case of high network traffic (for example, the hit rate on a Web server has reached its maximum value while the system is 100 percent busy), you must either redesign the network and redistribute the load, reduce the number of network clients, or increase the number of systems handling the network load. See the *Network Programmer's Guide* and the *Network Administration* manual for information on how to resolve network problems.

Table 10–2 lists network subsystem tuning guidelines and performance benefits as well as tradeoffs.

Table 10–2: Network Tuning Guidelines

Action	Performance Benefit	Tradeoff
Increase the size of the hash table that the kernel uses to look up TCP control blocks (Section 10.2.1)	Improves the TCP control block lookup rate and increases the raw connection rate	Slightly increases the amount of wired memory
Increase the number of TCP hash tables (Section 10.2.2)	Reduces head lock contention for SMP systems	Slightly increases the amount of wired memory
Increase the limits for partial TCP connections on the socket listen queue (Section 10.2.3)	Improves throughput and response time on systems that handle a large number of connections	Consumes memory when pending connections are retained in the queue
Increase the number of outgoing connection ports (Section 10.2.4)	Allows more simultaneous outgoing connections	None

Table 10–2: Network Tuning Guidelines (cont.)

Action	Performance Benefit	Tradeoff
Modify the range of outgoing connection ports (Section 10.2.5)	Allows you to use ports from a specific range	None
Enable TCP keepalive functionality (Section 10.2.6)	Enables inactive socket connections to time out	None
Increase the size of the kernel interface alias table (Section 10.2.7)	Improves the IP address lookup rate for systems that serve many domain names	Slightly increases the amount of wired memory
Make partial TCP connections time out more quickly (Section 10.2.8)	Prevents clients from overfilling the socket listen queue	A short time limit may cause viable connections to break prematurely
Make the TCP connection context time out more quickly at the end of the connection (Section 10.2.9)	Frees connection resources sooner	Reducing the timeout limit increases the potential for data corruption, so guideline should be applied with caution
Reduce the TCP retransmission rate (Section 10.2.10)	Prevents premature retransmissions and decreases congestion	A long retransmit time is not appropriate for all configurations
Enable the immediate acknowledgement of TCP data (Section 10.2.11)	Can improve network performance for some connections	May adversely affect network bandwidth
Increase the TCP maximum segment size (Section 10.2.12)	Allows sending more data per packet	May result in fragmentation at router boundary
Increase the size of the transmit and receive socket buffers (Section 10.2.13)	Buffers more TCP packets per socket	May decrease available memory when the buffer space is being used
Increase the size of the transmit and receive buffers for a UDP socket (Section 10.2.14)	Helps to prevent dropping UDP packets	May decrease available memory when the buffer space is being used
Allocate sufficient memory to the UBC (Section 10.2.15)	Improves disk I/O performance	May decrease the physical memory available to the virtual memory subsystem

Table 10–2: Network Tuning Guidelines (cont.)

Action	Performance Benefit	Tradeoff
Disable the use of a PMTU (Section 10.2.16)	Improves the efficiency of servers that handle remote traffic from many clients	May reduce server efficiency for LAN traffic
Increase the size of the ARP table (Section 10.2.17)	May improve network performance on a system that is simultaneously connected to many nodes on the same LAN	Consumes memory resources
Increase the maximum size of a socket buffer (Section 10.2.18)	Allows large socket buffer sizes	Consumes memory resources
Increase the number of IP input queues (Section 10.2.19)	Reduces IP input queue lock contention for SMP systems	None
Prevent dropped input packets (Section 10.2.20)	Allows high network loads	None
Enable `mbuf` cluster compression (Section 10.2.21)	Improves efficiency of network memory allocation	None
Modify the NetRAIN retry limit (Section 10.2.22)	Controls the time to detect an interface failure	Aggressive monitoring can increase CPU usage
Modify the NetRAIN monitoring timer (Section 10.2.23)	Controls the time to detect an interface failure	Aggressive monitoring can increase CPU usage

The following sections describe these tuning guidelines in detail.

10.2.1 Improving the Lookup Rate for TCP Control Blocks

You can modify the size of the hash table that the kernel uses to look up Transmission Control Protocol (TCP) control blocks. The `inet` subsystem attribute `tcbhashsize` specifies the number of hash buckets in the kernel TCP connection table (the number of buckets in the `inpcb` hash table). The default value is 512.

The kernel must look up the connection block for every TCP packet it receives, so increasing the size of the table can speed the search and and improve performance.

For Internet, Web, proxy, firewall, and gateway servers, set the `tcbhashsize` attribute to 16384.

See Section 4.4 for information about modifying kernel subsystem attributes.

10.2.2 Increasing the Number of TCP Hash Tables

If you have an SMP system, you may be able to reduce head lock contention by increasing the number of hash tables that the kernel uses to look up Transmission Control Protocol (TCP) control blocks.

Because the kernel must look up the connection block for every TCP packet it receives, a bottleneck may occur at the TCP hash table in SMP systems. Increasing the number of tables distributes the load and may improve performance.

The `inet` subsystem attribute `tcbhashnum` specifies the number of TCP hash tables. For busy Internet server SMP systems, you can increase the value of the `tcbhashnum` attribute to 16. The minimum and default values are 1; the maximum value is 64.

It is recommended that you make the value of the `tcbhashnum` attribute the same as the value of the `inet` subsystem attribute `ipqs`. See Section 10.2.19 for information.

See Section 4.4 for information about modifying kernel subsystem attributes.

10.2.3 Tuning the Socket Listen Queue Limits

You may be able to improve performance by increasing the limits for the socket listen queue (only for TCP). The `socket` subsystem attribute `somaxconn` specifies the maximum number of pending TCP connections (the socket listen queue limit) for each server socket. If the listen queue connection limit is too small, incoming connect requests may be dropped. Note that pending TCP connections can be caused by lost packets in the Internet or denial of service attacks. The default value of the `somaxconn` attribute is 1024; the maximum value is 65535.

To improve throughput and response time with fewer drops, you can increase the value of the `somaxconn` attribute. A busy system running applications that generate a large number of connections may have many pending connections. For Internet, Web, proxy, firewall, and gateway servers, set the value of the `somaxconn` attribute to the maximum value of 65535.

The `socket` subsystem attribute `sominconn` specifies the minimum number of pending TCP connections (backlog) for each server socket. The attribute controls how many SYN packets can be handled simultaneously before additional requests are discarded. The default value is zero. The value of the sominconn attribute overrides the application-specific backlog value, which may be set too low for some server software.

To improve performance without recompiling an application and for Internet, Web, proxy, firewall, and gateway servers, set the value of the sominconn attribute to the maximum value of 65535. The value of the sominconn attribute should be the same as the value of the somaxconn attribute.

Network performance can degrade if a client saturates a socket listen queue with erroneous TCP SYN packets, effectively blocking other users from the queue. To eliminate this problem, increase the value of the sominconn attribute to 65535. If the system continues to drop incoming SYN packets, you can decrease the value of the inet subsystem attribute tcp_keepinit to 30 (15 seconds).

See Section 10.1.2 for information about monitoring the sobacklog_hiwat, sobacklog_drops, and somaxconn_drops attributes. If the values show that the queues are overflowing, you may need to increase the socket listen queue limit.

See Section 4.4 for information about modifying kernel subsystem attributes.

10.2.4 Increasing the Number of Outgoing Connection Ports

When a TCP or UDP application creates an outgoing connection, the kernel dynamically allocates a nonreserved port number for each connection. The kernel selects the port number from a range of values between the value of the inet subsystem attribute ipport_userreserved_min and the value of the ipport_userreserved attribute. Using the default attribute values, the number of simultaneous outgoing connections is limited to 3976 (5000 minus 1024).

If your system requires many outgoing ports, you may want to increase the value of the ipport_userreserved attribute. If your system is a proxy server (for example, a Squid Caching Server or a firewall system) with a load of more than 4000 simultaneous connections, increase the value of the ipport_userreserved attribute to the maximum value of 65000.

It is not recommended that you reduce the value of the ipport_userreserved attribute to a value that is less than 5000 or increase it to a value that is higher than 65000.

You can also modify the range of outgoing connection ports. See Section 10.2.5 for information.

See Section 4.4 for information about modifying kernel subsystem attributes.

10.2.5 Modifying the Range of Outgoing Connection Ports

When a TCP or UDP application creates an outgoing connection, the kernel dynamically allocates a nonreserved port number for each connection. The kernel selects the port number from a range of values between the value of the `inet` subsystem attribute `ipport_userreserved_min` and the value of the `ipport_userreserved` attribute. Using the default values for these attributes, the range of outgoing ports starts at 1024 and stops at 5000.

If your system requires outgoing ports from a particular range, you can modify the values of the `ipport_userreserved_min` and `ipport_userreserved` attributes. The maximum value of both attributes is 65000. Do not reduce the `ipport_userreserved` attribute to a value that is less than 5000 or reduce the `ipport_userreserved_min` attribute to a value that is less than 1024.

See Section 4.4 for information about modifying kernel subsystem attributes.

10.2.6 Enabling TCP Keepalive Functionality

Keepalive functionality enables the periodic transmission of messages on a connected socket in order to keep connections active. If you enable keepalive, sockets that do not exit cleanly are cleaned up when the keepalive interval expires. If keepalive is not enabled, those sockets will continue to exist until you reboot the system.

Applications enable keepalive for sockets by setting the `setsockopt` function's `SO_KEEPALIVE` option. To override programs that do not set keepalive on their own or if you do not have access to the application sources, set the `inet` subsystem attribute `tcp_keepalive_default` to 1 in order to enable keepalive for all sockets.

If you enable keepalive, you can also configure the following TCP options for each socket:

- The `inet` subsystem attribute `tcp_keepidle` specifies the amount of idle time before sending a keepalive probe (specified in 0.5 second units). The default interval is 2 hours.
- The `inet` subsystem attribute `tcp_keepintvl` specifies the amount of time between retransmission of keepalive probes in 0.5 second units. The default interval is 75 seconds.

- The `inet` subsystem attribute `tcp_keepcnt` specifies the maximum number of keepalive probes that are sent before the connection is dropped. The default is 8 probes.
- The `inet` subsystem attribute `tcp_keepinit` specifies the maximum amount of time before an initial connection attempt times out in 0.5 second units. The default is 75 seconds.

See Section 4.4 for information about modifying kernel subsystem attributes.

10.2.7 Improving the Lookup Rate for IP Addresses

The `inifaddr_hsize` attribute specifies the number of hash buckets in the kernel interface alias table (`in_ifaddr`). The default value of the `inet` subsystem attribute `inifaddr_hsize` is 32; the maximum value is 512.

If a system is used as a server for many different server domain names, each of which are bound to a unique IP address, the code that matches arriving packets to the right server address uses the hash table to speed lookup operations for the IP addresses. Increasing the number of hash buckets in the table can improve performance on systems that use large numbers of aliases.

For the best performance, the value of the `inifaddr_hsize` attribute is always rounded down to the nearest power of 2. If you are using more than 500 interface IP aliases, specify the maximum value of 512. If you are using less than 250 aliases, use the default value of 32.

See Section 4.4 for information about modifying kernel subsystem attributes.

10.2.8 Decreasing the Partial TCP Connection Timeout Limit

The `inet` subsystem attribute `tcp_keepinit` specifies the amount of time that a partially established TCP connection remains on the socket listen queue before it times out. The value of the attribute is in units of 0.5 seconds. The default value is 150 units (75 seconds).

Partial connections consume listen queue slots and fill the queue with connections in the `SYN_RCVD` state. You can make partial connections time out sooner by decreasing the value of the `tcp_keepinit` attribute. However, do not set the value too low, because you may prematurely break connections associated with clients on network paths that are slow or network paths that lose many packets. Do not set the value to less than 20 units (10 seconds). If you have a 32000 socket queue limit, the default (75 seconds) is usually adequate.

Network performance can degrade if a client overfills a socket listen queue with TCP SYN packets, effectively blocking other users from the queue. To eliminate this problem, increase the value of the `sominconn` attribute to the maximum of 64000. If the system continues to drop SYN packets, decrease the value of the `tcp_keepinit` attribute to 30 (15 seconds).

See Section 4.4 for information about modifying kernel subsystem attributes.

10.2.9 Decreasing the TCP Connection Context Timeout Limit

You can make the TCP connection context time out more quickly at the end of a connection. However, this will increase the chance of data corruption.

The TCP protocol includes a concept known as the Maximum Segment Lifetime (MSL). When a TCP connection enters the `TIME_WAIT` state, it must remain in this state for twice the value of the MSL, or else undetected data errors on future connections can occur. The `inet` subsystem attribute `tcp_msl` determines the maximum lifetime of a TCP segment and the timeout value for the `TIME_WAIT` state.

The value of the attribute is set in units of 0.5 seconds. The default value is 60 units (30 seconds), which means that the TCP connection remains in `TIME_WAIT` state for 60 seconds (or twice the value of the MSL). In some situations, the default timeout value for the `TIME_WAIT` state (60 seconds) is too large, so reducing the value of the `tcp_msl` attribute frees connection resources sooner than the default behavior.

Do not reduce the value of the `tcp_msl` attribute unless you fully understand the design and behavior of your network and the TCP protocol. It is strongly recommended that you use the default value; otherwise, there is the potential for data corruption.

See Section 4.4 for information about modifying kernel subsystem attributes.

10.2.10 Decreasing the TCP Retransmission Rate

The `inet` subsystem attribute `tcp_rexmit_interval_min` specifies the minimum amount of time between the first TCP retransmission. For some wide area networks (WANs), the default value may be too small, causing premature retransmission timeouts. This may lead to duplicate transmission of packets and the erroneous invocation of the TCP congestion-control algorithms.

The `tcp_rexmit_interval_min` attribute is specified in units of 0.5 seconds. The default value is 2 units (1 second).

You can increase the value of the `tcp_rexmit_interval_min` attribute to slow the rate of TCP retransmissions, which decreases congestion and improves performance. However, not every connection needs a long retransmission time. Usually, the default value is adequate. Do not specify a value that is less than 1 unit. Do not change the attribute unless you fully understand TCP algorithms.

See Section 4.4 for information about modifying kernel subsystem attributes.

10.2.11 Disabling Delaying the Acknowledgment of TCP Data

The value of the `inet` subsystem attribute `tcpnodelack` determines whether the system delays acknowledging TCP data. The default is 0, which delays the acknowledgment of TCP data. Usually, the default is adequate. However, for some connections (for example, loopback), the delay can degrade performance.

You may be able to improve network performance by setting the value of the `tcpnodelack` attribute to 1, which disables the acknowledgment delay. However, this may adversely impact network bandwidth. Use the `tcpdump` command to check for excessive delays.

See Section 4.4 for information about modifying kernel subsystem attributes.

10.2.12 Increasing the Maximum TCP Segment Size

The `inet` subsystem attribute `tcp_mssdflt` specifies the TCP maximum segment size (the default value of 536). You can increase the value to 1460. This allows sending more data per socket, but may cause fragmentation at the router boundary.

See Section 4.4 for information about modifying kernel subsystem attributes.

10.2.13 Increasing the Transmit and Receive Socket Buffers

The `inet` subsystem attribute `tcp_sendspace` specifies the default transmit buffer size for a TCP socket. The `tcp_recvspace` attribute specifies the default receive buffer size for a TCP socket. The default value of both attributes is 32 KB. You can increase the value of these attributes to 60 KB. This allows you to buffer more TCP packets per socket. However, increasing the values uses more memory when the buffers are being used by an application (sending or receiving data).

You may want to increase the maximum size of a socket buffer before you increase the transmit and receive buffers. See Section 10.2.18 for information.

See Section 4.4 for information about modifying kernel subsystem attributes.

10.2.14 Increasing the Transmit and Receive Buffers for a UDP Socket

The `inet` subsystem attribute `udp_sendspace` specifies the default transmit buffer size for an Internet User Datagram Protocol (UDP) socket; the default value is 9 KB. The `inet` subsystem attribute `udp_recvspace` specifies the default receive buffer size for a UDP socket; the default value is 40 KB. You can increase the values of these attributes to 64 KB. However, increasing the values uses more memory when the buffers are being used by an application (sending or receiving data).

These attributes do not have an impact on the Network File System (NFS).

See Section 4.4 for information about modifying kernel subsystem attributes.

10.2.15 Allocating Sufficient Memory to the UBC

You must ensure that you have sufficient memory allocated to the Unified Buffer Cache (UBC). Servers that perform lots of file I/O (for example, Web and proxy servers) extensively utilize both the UBC and the virtual memory subsystem. In most cases, use the default value of 100 percent for the `vm` subsystem attribute `ubc-maxpercent`, which specifies the maximum amount of physical memory that can be allocated to the UBC. If necessary, you can decrease the size of the attribute by increments of 10 percent.

See Section 9.2.3 for more information about tuning the UBC.

See Section 4.4 for information about modifying kernel subsystem attributes.

10.2.16 Disabling Use of a PMTU

Packets transmitted between servers are fragmented into units of a specific size in order to ease transmission of the data over routers and small-packet networks, such as Ethernet networks. When the `inet` subsystem attribute `pmtu_enabled` is enabled (set to 1, which is the default behavior), the system determines the largest common path maximum transmission unit (PMTU) value between servers and uses it as the unit size. The system also creates a routing table entry for each client network that attempts to connect to the server.

On a server that handles local traffic and some remote traffic, enabling the use of a PMTU can improve bandwidth. However, if a server handles traffic among many remote clients, enabling the use of a PMTU can cause an excessive increase in the size of the kernel routing tables, which can reduce server efficiency.

If an Internet, Web, proxy, firewall, or gateway server has poor performance and the routing table increases to more than 1000 entries, set the value of the `pmtu_enabled` attribute to 0 to disable the use of PMTU protocol.

See Section 4.4 for information about modifying kernel subsystem attributes.

10.2.17 Increasing the Size of the ARP Table

The `net` subsystem attribute `arptab_nb` specifies the number of hash buckets in the address resolution protocol (ARP) table (that is, the table's width). The default value is 37.

You can modify the value of the `arptab_nb` attribute if the ARP table is thrashing. In addition, you may be able to improve performance by modifying the attribute. You can view the ARP table by using the `arp -a` command or the `kdbx arp` debugger extension. However, changing the attribute values will not affect performance unless the system is simultaneously connected to many nodes on the same LAN. See the *Kernel Debugging* manual and `kdbx`(8) for more information.

You can increase the width of the ARP table by increasing the value of the `inarptab_nb` attribute. In general, wide ARP tables can decrease the chance that a search will be needed to match an address to an ARP entry. Increasing the value of the `arptab_nb` attribute will increase the memory used by the ARP table.

See Section 4.4 for information about modifying kernel subsystem attributes.

10.2.18 Increasing the Maximum Size of a Socket Buffer

If you require a large socket buffer, increase the maximum socket buffer size. To do this, increase the value of the `socket` subsystem attribute `sb_max`, before you increase the size of the transmit and receive socket buffers (see Section 10.2.13).

The default maximum socket buffer size is 128 KB.

See Section 4.4 for information about modifying kernel subsystem attributes.

10.2.19 Increasing the Number of IP Input Queues

For SMP systems, you may be able to reduce lock contention at the IP input queue by increasing the number of queues and distributing the load. The `inet` subsystem attribute `ipqs` specifies the number of IP input queues.

For busy Internet server SMP systems, you may want to increase the value of the `ipqs` attribute to 16. The default value of the `ipqs` attribute is 1. The minimum value is 1; the maximum value is 64.

It is recommended that you make the value of the `ipqs` attribute the same as the value of the `inet` subsystem attribute `tcbhashnum`. See Section 10.2.2 for information.

See Section 4.4 for information about modifying kernel subsystem attributes.

10.2.20 Preventing Dropped Input Packets

If the IP input queue overflows under a heavy network load, input packets may be dropped. To check for dropped packets, examine the `ipintrq` kernel structure by using `dbx`. For example:

```
# dbx -k /vmunix
(dbx)print ipintrq
struct {
    ifq_head = (nil)
    ifq_tail = (nil)
    ifq_len = 0
    ifq_maxlen = 512
    ifq_drops = 0
 .
 .
 .
```

If the `ifq_drops` field is not 0, increase the value of the `inet` subsystem attribute `ipqmaxlen`. For example, you may want to increase the value to 2000. The default and minimum value is 512; the maximum value is 65535.

See Section 4.4 for information about modifying kernel subsystem attributes.

The `ipqmaxlen` attribute cannot be tuned at run time. You can immediately determine the impact of the kernel modification by using `dbx` to increase the value of the `ipintrq.ifq_maxlen` kernel variable. For example:

```
# dbx -k /vmunix
(dbx)patch ipintrq.ifq_maxlen=2000
```

See Section 4.4.6 for information about using `dbx`.

10.2.21 Enabling mbuf Cluster Compression

The `socket` subsystem attribute `sbcompress_threshold` controls whether `mbuf` clusters are compressed.

By default, `mbuf` clusters are not compressed (`sbcompress_threshold` is set to 0), which can cause proxy servers to consume all the available `mbuf` clusters. This situation is more likely to occur if you are using FDDI instead of Ethernet.

To enable `mbuf` cluster compression, modify the default value of the `socket` subsystem attribute `sbcompress_threshold`. Packets will be copied into the existing `mbuf` clusters if the packet size is less than this value. For proxy servers, specify a value of 600.

To determine the memory that is being used for `mbuf` clusters, use the `netstat -m` command. The following example is from a firewall server with 128 MB memory that does not have `mbuf` cluster compression enabled:

```
# netstat -m
  2521 Kbytes for small data mbufs (peak usage 9462 Kbytes)
 78262 Kbytes for mbuf clusters (peak usage 97924 Kbytes)
  8730 Kbytes for sockets (peak usage 14120 Kbytes)
  9202 Kbytes for protocol control blocks (peak usage 14551
     2 Kbytes for routing table (peak usage 2 Kbytes)
     2 Kbytes for socket names (peak usage 4 Kbytes)
     4 Kbytes for packet headers (peak usage 32 Kbytes)
 39773 requests for mbufs denied
     0 calls to protocol drain routines
 98727 Kbytes allocated to network
```

The previous example shows 39773 requests for memory were denied. This indicates a problem because this value should be 0. The example also shows that 78 MB of memory has been assigned to `mbuf` clusters, and that 98 MB of memory is being consumed by the network subsystem.

If you increase the value of the `sbcompress_threshold` attribute to 600, the memory allocated to the network subsystem immediately decreases to 18 MB, because compression at the kernel socket buffer interface results in a more efficient use of memory.

10.2.22 Modifying the NetRAIN Retry Limit

The `netrain` subsystem attribute `nr_max_retries` specifies how many failed tests must occur before a NetRAIN interface is determined to have failed and a backup interface is brought on line. The default value is 4.

Decreasing the default value will cause NetRAIN to be more aggressive about declaring an interface to be failed and forcing an interface failover, but will increase CPU usage. Increasing the default value will cause NetRAIN to be more tolerant of temporary failures, but may result in long failover times.

For ATM LAN Emulation (LANE) interfaces, set the value of the `nr_max_retries` attribute to 5.

10.2.23 Modifying the NetRAIN Monitoring Timer

The `netrain` subsystem attribute `netrain_timeout` specifies the number of clock ticks between runs of the kernel thread that monitors the health of the network interfaces. All other NetRAIN timers are based on this frequency. The default value is 1000 ticks (1 second).

Decreasing the value of the `netrain_timeout` attribute causes NetRAIN to monitor network interfaces more aggressively so that it can quickly detect a failed interface, but it will increase CPU usage. Increasing the value will cause NetRAIN to be more tolerant of temporary failures, but may result in long failover times.

11 Managing Application Performance

You may be able to improve overall Tru64 UNIX performance by improving application performance. This chapter describes how to:

- Profile and debug applications (Section 11.1)
- Improve application performance (Section 11.2)

11.1 Gathering Profiling and Debugging Information

You can use profiling to identify sections of application code that consume large portions of execution time. To improve performance, concentrate on improving the coding efficiency of those time-intensive sections.

Table 11–1 describes the commands you can use to obtain information about applications. Detailed information about these tools is located in the *Programmer's Guide* and the *Kernel Debugging* manual.

Table 11–1: Application Profiling and Debugging Tools

Name	Use	Description
`atom`	Profiles applications	Consists of a set of prepackaged tools (`third`, `hiprof`, or `pixie`) that can be used to instrument applications for profiling or debugging purposes. The `atom` toolkit also consists of a command interface and a collection of instrumentation routines that you can use to create custom tools for instrumenting applications. See the *Programmer's Guide* manual and `atom`(1) for more information.
`third`	Checks memory access and detects memory leaks in applications	Performs memory access checks and memory leak detection of C and C++ programs at run time, by using the `atom` tool to add code to executable and shared objects. The Third Degree tool instruments the entire program, including its referenced libraries. See `third`(5) for more information.

Table 11–1: Application Profiling and Debugging Tools (cont.)

Name	Use	Description
`hiprof`	Produces a profile of procedure execution times in an application	An `atom`-based program profiling tool that produces a flat profile, which shows the execution time spent in any given procedure, and a hierarchical profile, which shows the time spent in a given procedure and all of its descendents. The `hiprof` tool uses code instrumentation instead of program counter (PC) sampling to gather statistics. The `gprof` command is usually used to filter and merge output files and to format profile reports. See `hiprof`(5) for more information.
`pixie`	Profiles basic blocks in an application	Produces a profile showing the number of times each instruction was executed in a program. The information can be reported as tables or can be used to automatically direct later optimizations by using the `-feedback`, `-om`, or `-cord` options in the C compiler (see `cc`(1)). The `pixie` profiler reads an executable program, partitions it into basic blocks, and writes an equivalent program containing additional code that counts the execution of each basic block. The `pixie` utility also generates a file containing the address of each of the basic blocks. When you run this `pixie`-generated program, it generates a file containing the basic block counts. The `prof` and `pixstats` commands can analyze these files. See `pixie`(5) for more information.

Table 11–1: Application Profiling and Debugging Tools (cont.)

Name	Use	Description
`prof`	Analyzes profiling data and displays a profile of statistics for each procedure in an application	Analyzes profiling data and produces statistics showing which portions of code consume the most time and where the time is spent (for example, at the routine level, the basic block level, or the instruction level). The `prof` command uses as input one or more data files generated by the `kprofile`, `uprofile`, or `pixie` profiling tools. The `prof` command also accepts profiling data files generated by programs linked with the `-p` switch of compilers such as `cc`. The information produced by `prof` allows you to determine where to concentrate your efforts to optimize source code. See `prof`(1) for more information.
`gprof`	Analyzes profiling data and displays procedure call information and statistical program counter sampling in an application	Analyzes profiling data and allows you to determine which routines are called most frequently, and the source of the routine call, by gathering procedure call information and performing statistical program counter (PC) sampling. The `gprof` tool produces a flat profile of the routines' CPU usage. To produce a graphical execution profile of a program, the tool uses data from PC sampling profiles, which are produced by programs compiled with the `cc -pg` command, or from instrumented profiles, which are produced by programs modified by the `atom -tool hiprof` command. See `gprof`(1) for more information.
`uprofile`	Profiles user code in an application	Profiles user code using performance counters in the Alpha chip. The `uprofile` tool allows you to profile only the executable part of a program. The `uprofile` tool does not collect information on shared libraries. You process the performance data collected by the tool with the `prof` command. See the *Kernel Debugging* manual or `uprofile`(1) for more information.

Table 11–1: Application Profiling and Debugging Tools (cont.)

Name	Use	Description
`dbx`	Debugs running kernels, programs, and crash dumps, and examines and temporarily modifies kernel variables	Provides source-level debugging for C, Fortran, Pascal, assembly language, and machine code. The `dbx` debugger allows you to analyze crash dumps, trace problems in a program object at the source-code level or at the machine code level, control program execution, trace program logic and flow of control, and monitor memory locations. Use `dbx` to debug kernels, debug stripped images, examine memory contents, debug multiple threads, analyze user code and applications, display the value and format of kernel data structures, and temporarily modify the values of some kernel variables. See `dbx`(8) for more information.
`ladebug`	Debugs kernels and applications	Debugs programs and the kernel and helps locate run-time programming errors. The `ladebug` symbolic debugger is an alternative to the `dbx` debugger and provides both command-line and graphical user interfaces and support for debugging multithreaded programs. See the *Ladebug Debugger Manual* and `ladebug`(1) for more information.
`lsof`	Displays open files	Displays information about files that are currently opened by the running processes. The `lsof` is is available on the Tru64 UNIX Freeware CD-ROM.

11.2 Improving Application Performance

Well-written applications use CPU, memory, and I/O resources efficiently. Table 11–2 describes some guidelines to improve application performance.

Table 11–2: Application Performance Improvement Guidelines

Recommendations	Performance Benefit	Tradeoff
Install the latest operating system patches (Section 11.2.1)	Provides the latest optimizations	None
Use the latest version of the compiler (Section 11.2.2)	Provides the latest optimizations	None
Use parallelism (Section 11.2.3)	Improves SMP performance	None

Table 11–2: Application Performance Improvement Guidelines (cont.)

Recommendations	Performance Benefit	Tradeoff
Optimize applications (Section 11.2.4)	Generates more efficient code	None
Use shared libraries (Section 11.2.5)	Frees memory	May increase execution time
Reduce application memory requirements (Section 11.2.6)	Frees memory	Program may not run optimally
Use memory locking as part of real-time program initialization (Section 11.2.7)	Allows you to lock and unlock memory as needed	Reduces the number of pages that can be used for paging and swapping

The following sections describe how to improve application performance.

11.2.1 Using the Latest Operating System Patches

Always install the latest operating system patches, which often contain performance enhancements.

Check the `/etc/motd` file to determine which patches you are running. See your Compaq service representative or the Compaq Tru64 UNIX Web page for information about installing patches.

11.2.2 Using the Latest Version of the Compiler

Always use the latest version of the compiler to build your application program. In general, new versions of a compiler perform advanced optimizations.

Check the software on your system to ensure that you are using the latest versions of the compiler.

11.2.3 Using Parallelism

To enhance parallelism, application developers working in Fortran or C should consider using the Kuch & Associates Preprocessor (KAP), which can have a significant impact on SMP performance. See the *Programmer's Guide* for details on KAP.

11.2.4 Optimizing Applications

Optimizing an application program can involve modifying the build process or modifying the source code. Various compiler and linker optimization levels can be used to generate more efficient user code. See the *Programmer's Guide* for more information on optimization.

Whether you are porting an application from a 32-bit system to Tru64 UNIX or developing a new application, never attempt to optimize an application until it has been thoroughly debugged and tested. If you are porting an application written in C, use the `lint` command with the `-Q` option or compile your program using the C compiler's `-check` option to identify possible portability problems that you may need to resolve.

11.2.5 Using Shared Libraries

Using shared libraries reduces the need for memory and disk space. When multiple programs are linked to a single shared library, the amount of physical memory used by each process can be significantly reduced. However, shared libraries initially result in an execution time that is slower than if you had used static libraries.

11.2.6 Reducing Application Memory Requirements

You may be able to reduce an application's use of memory, which provides more memory resources for other processes or for file system caching. Follow these coding considerations to reduce your application's use of memory:

- Configure and tune applications according to the guidelines provided by the application's installation procedure. For example, you may be able to reduce an application's anonymous memory requirements, set parallel/concurrent processing attributes, size shared global areas and private caches, and set the maximum number of open/mapped files.
- You may want to use the `mmap` function instead of the `read` or `write` function in your applications. The `read` and `write` system calls require a page of buffer memory and a page of UBC memory, but `mmap` requires only one page of memory.
- Look for data cache collisions between heavily used data structures, which occur when the distance between two data structures allocated in memory is equal to the size of the primary (internal) data cache. If your data structures are small, you can avoid collisions by allocating them contiguously in memory. To do this, use a single `malloc` call instead of multiple calls.
- If an application uses large amounts of data for a short time, allocate the data dynamically with the `malloc` function instead of declaring it statically. When you have finished using dynamically allocated memory, it is freed for use by other data structures that occur later in the

program. If you have limited memory resources, dynamically allocating data reduces an application’s memory usage and can substantially improve performance.

- If an application uses the `malloc` function extensively, you may be able to improve its processing speed or decrease its memory utilization by using the function’s control variables to tune memory allocation. See `malloc`(3) for details on tuning memory allocation.
- If your application fits in a 32-bit address space and allocates large amounts of dynamic memory by using structures that contain many pointers, you may be able to reduce memory usage by using the `-xtaso` option. The `-xtaso` option is supported by all versions of the C compiler (`-newc`, `-migrate`, and `-oldc` versions). To use the `-xtaso` option, modify your source code with a C-language pragma that controls pointer size allocations. See `cc`(1) for details.

See the *Programmer’s Guide* for detailed information on process memory allocation.

11.2.7 Controlling Memory Locking

Real-time application developers should consider memory locking as a required part of program initialization. Many real-time applications remain locked for the duration of execution, but some may want to lock and unlock memory as the application runs. Memory-locking functions allow you to lock the entire process at the time of the function call and throughout the life of the application. Locked pages of memory cannot be used for paging or swapping.

Memory locking applies to a process’s address space. Only the pages mapped into a process’s address space can be locked into memory. When the process exits, pages are removed from the address space and the locks are removed.

Use the `mlockall` function to lock all of a process’ address space. Locked memory remains locked until either the process exits or the application calls the `munlockall` function.

Memory locks are not inherited across a fork and all memory locks associated with a process are unlocked on a call to the `exec` function or when the process terminates. See the *Guide to Realtime Programming* manual and `mlockall`(3) for more information.

Glossary

This glossary lists the terms that are used to describe Tru64 UNIX performance and availability.

active list

Pages that are being used by the virtual memory subsystem or the UBC.

adaptive RAID 3/5

See **dynamic parity RAID**.

anonymous memory

Modifiable memory that is used for stack, heap, or `malloc`.

attributes

Dynamically configurable kernel variables, whose values you can modify to improve system performance. You can utilize new attribute values without rebuilding the kernel.

bandwidth

The rate at which an I/O subsystem or component can transfer bytes of data. Bandwidth is especially important for applications that perform large sequential transfers. Bandwidth is also called the transfer rate.

bottleneck

A system resource that is being pushed near to its capacity and is causing a performance degradation.

cache

A temporary location for holding data that is used to improve performance by reducing latency. CPU caches and secondary caches hold physical addresses. Disk track caches and write-back caches hold disk data. Caches can be volatile (that is, not backed by disk data or a battery) or nonvolatile.

capacity

The maximum theoretical throughput of a system resource, or the maximum amount of data, in bytes, that a disk can contain. A resource that has reached its capacity may become a bottleneck and degrade performance.

cluster

A loosely coupled group of servers (cluster member systems) that share data for the purposes of high availability. Some cluster products utilize a high-performance interconnect for fast and dependable communication.

copy-on-write page fault

A page fault that occurs when a process needs to modify a read-only virtual page.

configuration

The assemblage of hardware and software that comprises a system or a cluster. For example, CPUs, memory boards, the operating system, and mirrored disks are parts of a configuration.

configure

To set up or modify a hardware or software configuration. For example, configuring the I/O subsystem can include connecting SCSI buses and setting up mirrored disks.

deferred mode

A swap space allocation mode by which swap space is not reserved until the system needs to write a modified virtual page to swap space. Deferred mode is sometimes referred to as lazy mode.

disk access time

A combination of the seek time and the rotational latency, measured in milliseconds. A low access time is especially important for applications that perform many small I/O operations.

dynamic parity RAID

Also called adaptive RAID 3/5, dynamic parity RAID combines the features of RAID 3 and RAID 5 to improve disk I/O performance and availability for a wide variety of applications. Adaptive RAID 3/5 dynamically adjusts, according to workload needs, between data transfer-intensive algorithms and I/O operation-intensive algorithms.

eager mode

See **immediate mode**.

fail over

To automatically utilize a redundant resource after a hardware or software failure, so that the resource remains available. For example, if a cluster member system fails, the applications running on that system automatically fail over to another member system.

file-backed memory

Memory that is used for program text or shared libraries.

free list

Pages that are clean and are not being used (the size of this list controls when page reclamation occurs).

hardware RAID

A storage subsystem that provides RAID functionality by using intelligent controllers, caches, and software.

high availability

The ability of a resource to withstand a hardware or software failure. High availability is achieved by using some form of resource duplication that removes single points of failure. Availability also is measured by a resource's reliability. No resource can be protected against an infinite number of failures.

immediate mode

A swap space allocation mode by which swap space is reserved when modifiable virtual address space is created. Immediate mode is often referred to as eager mode and is the default swap space allocation mode

kernel variables

Variables that determine kernel and subsystem behavior and performance. System attributes and parameters are used to access kernel variables.

lazy mode

See **deferred mode**.

latency

The amount of time to complete a specific operation. Latency is also called delay. High performance requires a low latency time. I/O latency can be measured in milliseconds, while memory latency is measured in microseconds. Memory latency depends on the memory bank configuration and the system's memory requirements.

mirroring

Maintaining identical copies of data on different disks, which provides high data availability and improves disk read performance. Mirroring is also known as RAID 1.

multiprocessor

A system with two or more processors (CPUs) that share common physical memory.

page

The smallest portion of physical memory that the system can allocate (8 KB of memory).

page coloring

The attempt to map a process’ entire resident set into the secondary cache.

page fault

An instruction to the virtual memory subsystem to locate a requested page and make the virtual-to-physical address translation in the page table.

page in

To move a page from a disk location to physical memory.

page-in page fault

A page fault that occurs when a requested address is found in swap space.

page out

To write the contents of a modified (dirty) page from physical memory to swap space.

page table

An array that contains an entry for each current virtual-to-physical address translation.

paging

The process by which pages that are allocated to processes and the UBC are reclaimed for reuse.

parameters

Statically configurable kernel variables, whose values can be modified to improve system performance. You must rebuild the kernel to utilize new parameter values. Many parameters have corresponding attributes.

parity RAID

A type of RAID functionality that provides high data availability by storing on a separate disk or multiple disks redundant information that is used to regenerate data.

RAID

RAID (redundant array of independent disks) technology provides high disk I/O performance and data availability. The Tru64 UNIX operating system provides RAID functionality by using disks and software (LSM). Hardware-based RAID functionality is provided by intelligent controllers, caches, disks, and software.

RAID 0

Also known as disk striping, RAID 0 functionality divides data into blocks and distributes the blocks across multiple disks in a array. Distributing the disk I/O load across disks and controllers improves disk I/O performance.

However, striping decreases availability because one disk failure makes the entire disk array unavailable.

RAID 1

Also known as data mirroring, RAID 1 functionality maintains identical copies of data on different disks in an array. Duplicating data provides high data availability. In addition, RAID 1 improves the disk read performance, because data can be read from two locations. However, RAID 1 decreases disk write performance, because data must be written twice. Mirroring *n* disks requires 2*n* disks.

RAID 3

RAID 3 functionality divides data blocks and distributes (stripes) the data across a disk array, providing parallel access to data. RAID 3 provides data availability; a separate disk stores redundant parity information that is used to regenerate data if a disk fails. It requires an extra disk for the parity information. RAID 3 increases bandwidth, but it provides no improvement in the throughput. RAID 3 can improve the I/O performance for applications that transfer large amounts of sequential data.

RAID 5

RAID 5 functionality distributes data blocks across disks in an array. Redundant parity information is distributed across the disks, so each array member contains the information that is used to regenerate data if a disk fails. RAID 5 allows independent access to data and can handle simultaneous I/O operations. RAID 5 provides data availability and improves performance for large file I/O operations, multiple small data transfers, and I/O read operations. It is not suited to applications that are write-intensive.

random access pattern

Refers to an access pattern in which data is read from or written to blocks in various locations on a disk.

raw I/O

I/O to a device that does not use a file system. Raw I/O bypasses buffers and caches, and can provide better performance than file system I/O.

redundancy

The duplication of a resource for purposes of high availability. For example, you can obtain data redundancy by mirroring data across different disks or by using parity RAID. You can obtain system redundancy by setting up a cluster, and network redundancy by using multiple network connections. The more levels of resource redundancy you have, the greater the resource availability. For example, a cluster with four member systems has more levels of redundancy and thus higher availability than a two-system cluster.

reliability

The average amount of time that a component will perform before a failure that causes a loss of data. Often expressed as the mean time to data loss (MTDL) or the mean time to first failure (MTTF).

resident set

The complete set of all the virtual addresses that have been mapped to physical addresses (that is, all the pages that have been accessed during process execution).

resource

A hardware or software component (such as the CPU, memory, network, or disk data) that is available to users or applications.

physical memory

The total capacity of the memory boards installed in your system. Physical memory is either wired by the kernel or it is shared by virtual memory and the UBC.

rotational latency

The amount of time, in milliseconds, for a disk to rotate to a specific disk sector.

scalability

The ability of a system to utilize additional resources with a predictable increase in performance, or the ability of a system to absorb an increase in workload without a significant performance degradation.

seek time

The amount of time, in milliseconds, for a disk head to move to a specific disk track.

sequential access pattern

Refers to an access pattern in which data is read from or written to contiguous blocks on a disk.

short page fault

A page fault that occurs when a requested address is found in the virtual memory subsystem’s internal data structures.

SMP

Symmetrical multiprocessing (SMP) is the ability of a multiprocessor system to execute the same version of the operating system, access common memory, and execute instructions simultaneously.

software RAID

Storage subsystem that provides RAID functionality by using software (for example, LSM).

striping

Distributing data across multiple disks in a disk array, which improves I/O performance by allowing parallel access. Striping is also known as RAID 0. Striping can improve the performance of sequential data transfers and I/O operations that require high bandwidth.

swap in

To move a swapped-out process' pages from disk swap space to physical memory in order for the process to execute. Swapins occur only if the number of pages on the free page list is higher than a specific amount for a period of time.

swap out

To move all the modified pages associated with a low-priority process or a process with a large resident set size from physical memory to swap space. A swapout occurs when number of pages on the free page list falls below a specific amount for a period of time. Swapouts will continue until the number of pages on the free page list reaches a specific amount.

swapping

Writing a suspended process' modified (dirty) pages to swap space, and putting the clean pages on the free list. Swapping occurs when the number of pages on the free list falls below a specific threshold.

throughput

The rate at which an I/O subsystem or component can perform I/O operations. Throughput is especially important for applications that perform many small I/O operations.

tune

To modify the kernel by changing the values of kernel variables, thus improving system performance.

UBC

See **Unified Buffer Cache**.

Unified Buffer Cache

A portion of physical memory that is used to cache most-recently accessed file system data.

UltraSCSI

Refers to a storage configuration of devices (adapters or controllers) and disks that doubles the performance of SCSI-2 configurations. UltraSCSI (also called Fast-20) supports increased bandwidth and throughput, and can support extended cable distances.

virtual address space

The array of pages that an application can map into physical memory. Virtual address space is used for anonymous memory (memory used for stack, heap, or `malloc`) and for file-backed memory (memory used for program text or shared libraries).

virtual memory

A subsystem that uses a portion of physical memory, disk swap space, and daemons and algorithms in order to control the allocation of memory to processes and to the UBC.

VLDB

Refers to very-large database (VLDB) systems, which are VLM systems that use a large and complex storage configuration. The following is a typical VLM/VLDB system configuration:

- An SMP system with two or more high-speed CPUs
- More than 4 GB of physical memory
- Multiple high-performance host bus adapters
- RAID storage configuration for high performance and high availability

VLM

Refers to very-large memory (VLM) systems, which utilize 64-bit architecture, multiprocessing, and at least 2 GB of memory.

wired list

Pages that are wired by the kernel and cannot be reclaimed.

working set

The set of virtual addresses that are currently mapped to physical addresses. The working set is a subset of the resident set and represents a snapshot of the process' resident set.

workload

The total number of applications running on a system and the users utilizing a system at any one time under normal conditions.

zero-filled-on-demand page fault

A page fault that occurs when a requested address is accessed for the first time.

Index

How to Order Tru64 UNIX Documentation

You can order documentation for the Tru64 UNIX operating system and related products at the following Web site:

`http://www.businesslink.digital.com/`

If you need help deciding which documentation best meets your needs, see the Tru64 UNIX *Documentation Overview* or call **800-344-4825** in the United States and Canada. In Puerto Rico, call **787-781-0505**. In other countries, contact your local Compaq subsidiary.

If you have access to Compaq's intranet, you can place an order at the following Web site:

`http://asmorder.nqo.dec.com/`

The following table provides the order numbers for the Tru64 UNIX operating system documentation kits. For additional information about ordering this and related documentation, see the *Documentation Overview* or contact Compaq.

Name	Order Number
Tru64 UNIX Documentation CD-ROM	QA-MT4AA-G8
Tru64 UNIX Documentation Kit	QA-MT4AA-GZ
End User Documentation Kit	QA-MT4AB-GZ
Startup Documentation Kit	QA-MT4AC-GZ
General User Documentation Kit	QA-MT4AD-GZ
System and Network Management Documentation Kit	QA-MT4AE-GZ
Developer's Documentation Kit	QA-MT5AA-GZ
General Programming Documentation Kit	QA-MT5AB-GZ
Windows Programming Documentation Kit	QA-MT5AC-GZ
Reference Pages Documentation Kit	QA-MT4AG-GZ
Device Driver Kit	QA-MT4AV-G8

Reader's Comments

Tru64 UNIX
System Configuration and Tuning
AA-Q0R3G-TE

Compaq welcomes your comments and suggestions on this manual. Your input will help us to write documentation that meets your needs. Please send your suggestions using one of the following methods:

- This postage-paid form
- Internet electronic mail: `readers_comment@zk3.dec.com`
- Fax: (603) 884-0120, Attn: UBPG Publications, ZKO3-3/Y32

If you are not using this form, please be sure you include the name of the document, the page number, and the product name and version.

Please rate this manual:

	Excellent	Good	Fair	Poor
Accuracy (software works as manual says)	☐	☐	☐	☐
Clarity (easy to understand)	☐	☐	☐	☐
Organization (structure of subject matter)	☐	☐	☐	☐
Figures (useful)	☐	☐	☐	☐
Examples (useful)	☐	☐	☐	☐
Index (ability to find topic)	☐	☐	☐	☐
Usability (ability to access information quickly)	☐	☐	☐	☐

Please list errors you have found in this manual:

Page	Description

Additional comments or suggestions to improve this manual:

____
____
____
____

What version of the software described by this manual are you using? ____________________

Name, title, department ____
Mailing address ____
Electronic mail ____
Telephone ____
Date ____

------- Do not cut or tear - fold here and tape -------

COMPAQ

NO POSTAGE
NECESSARY IF
MAILED IN THE
UNITED STATES

BUSINESS REPLY MAIL
FIRST CLASS MAIL PERMIT NO. 33 MAYNARD MA

POSTAGE WILL BE PAID BY ADDRESSEE

COMPAQ COMPUTER CORPORATION
UBPG PUBLICATIONS MANAGER
ZKO3 3/Y32
110 SPIT BROOK RD
NASHUA NH 03062 9987

------- Do not cut or tear - fold here and tape -------

Cut on this line