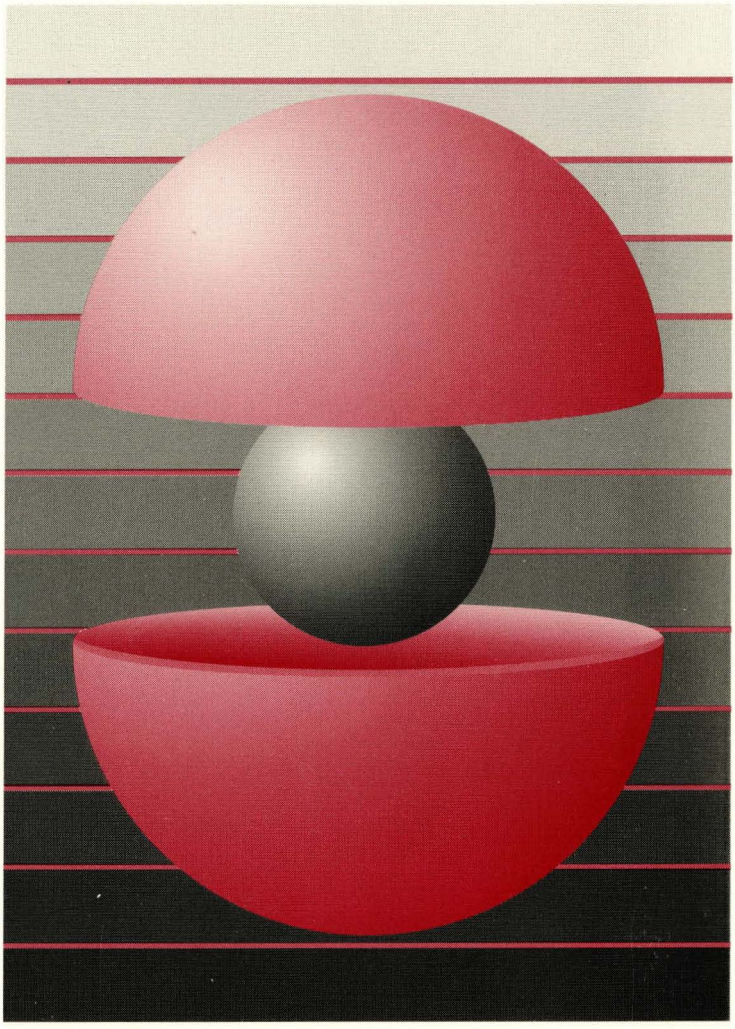
DEC OSF/1

digital

System Tuning and Performance Management



Part Number: AA-Q0R3A-TE

DEC OSF/1

System Tuning and Performance Management

Order Number: AA-Q0R3A-TE

February 1994

Product Version: DEC OSF/1 Version 2.0

This manual explains how you can adjust your applications and various components of the DEC OSF/1 operating system to achieve better performance.

digital equipment corporation
Maynard, Massachusetts

Contents

About This Manual

1 Operating System Overview

Index

Figures

Tables

About This Manual

This manual is useful for system managers who are either experiencing system performance problems or want to get the maximum benefit from their system. Performance problems can be caused by the applications you are running, the virtual memory subsystem, the disk I/O subsystem, file system layout policies, or the network subsystem.

Often it is difficult to determine the cause of the problem or bottleneck. This manual will assist you in determining where the problem resides, what the underlying cause may be, and how to resolve it. Because the DEC OSF/1 operating system can be used in many different types of environments, the resolutions are more suggestions than hard answers.

Audience

This manual is intended for system administrators who are responsible for managing a DEC OSF/1 operating system, and for programmers who are writing applications for the DEC OSF/1 operating system. Administrators and programmers should have in-depth knowledge of operating system concepts, commands, and utilities. Is is also especially important for administrators to understand how their systems are being used. Such understanding can be crucial to the success of an effort to tune a system for better performance.

Organization

This manual consists of three chapters:

Chapter 1	Provides an overview of the system components that can be adjusted to improve performance.
Chapter 2	Describes the tools used to analyze how system resources are being used.
Chapter 3	Provides information about how to tune your system for better performance.

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 DEC OSF/1 operating system.

The printed version of the DEC OSF/1 documentation set is color coded to help specific audiences quickly find the books that meet their needs. (You can order the printed documentation from Digital.) 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	Teal
System Administrators	S	Red
Network Administrators	N	Yellow
Programmers	P	Blue
Reference Page Users	R	Black

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 DEC OSF/1 documentation set.

Reader's Comments

Digital welcomes your comments on this or any other DEC OSF/1 manual. You can send your comments in the following ways:

- Internet electronic mail:
  `readers_comment@ravine.zk3.dec.com`
- Fax: 603-881-0120 Attn: USG Documentation, ZK03-3/Y32
- A completed Reader's Comments form (postage paid, if mailed in the United States). Two Reader's Comments forms are located at the back of each printed DEC OSF/1 manual.

If you have suggestions for improving particular sections or find any errors, please indicate the title, order number, and section numbers. Digital also welcomes general comments.

Conventions

The following conventions are used in this manual:

`%` `$`	A percent sign represents the C shell system prompt. A dollar sign represents the system prompt for the Bourne and Korn shells.
`#`	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.
Ctrl/*x*	This symbol indicates that you hold down the first named key while pressing the key or mouse button that follows the slash. In examples, this key combination is enclosed in a box (for example, Ctrl/C).

Operating System Overview 1

Before you attempt to tune your system to improve performance, you must fully understand your applications, users, and system environment and you must correctly diagnose the source of your performance problem. This chapter provides information on the major elements of the system environment that must be considered in a performance and tuning analysis:

- System architecture (Section 1.1)
- Process management (Section 1.2)
- Memory usage (Section 1.3)
- Interprocess communication (Section 1.4)
- I/O subsystems (Section 1.5)

For more information on all components of the operating system, refer to the manual *Technical Overview.*

1.1 Alpha AXP 64-bit Architecture

The Alpha AXP architecture contains instructions that can operate directly on 64- and 32-bit data items. It does not contain instructions that operate directly on data items that are smaller than 32 bits. As a result, if a program uses a data item that is smaller than 32 bits, the compiler generates a sequence of instructions to extract the data item from a 32-bit quantity. Thus, it consumes more system resources to access a data item that is less than 32 bits than it does to access a 32-bit or 64-bit data item.

This increase in overhead will not cause a problem if a program uses small data only occasionally. However, if a program uses small data regularly (for example, in the body of a critical loop), this overhead can be significant. For information on how to modify data declarations in your program to avoid this problem, see Section 3.2.2.1.

The Alpha AXP architecture also affects disk space and memory usage. While the 64-bit architecture benefits applications that would otherwise exhaust the address space in a 32-bit implementation, the OSF/1 operating system implementation on Alpha AXP systems does result in larger memory and disk space requirements than those associated with operating systems based on a 32-bit architecture. For details on the Alpha AXP architecture, see the *Alpha Architecture Reference Manual.*

1.2 Process Management

Programs that are being executed by the DEC OSF/1 operating system are known as processes. Each process runs within a protected virtual address space. The process abstraction is separated into two low-level abstractions, the task and the thread:

- A task is not executable, but it does have a protected virtual address space. This is the environment in which one or more threads can execute (that is, a thread executes within the framework of a task).
- A thread has access to all the system resources assigned to the task. If the task contains multiple threads, the threads share the task's resources. (See the *Guide to DECthreads* and the *Programmer's Guide* for information about programming using threads.)

The kernel schedules threads. A process priority can be managed by the `nice` interface or by the realtime interface. The `nice` interface allows adjustments of priorities within the range 20 through -20, where 20 is the lowest priority. You can adjust realtime priorities on those systems running the realtime kernel by using the `sched_setscheduler` interface.

Under the DEC OSF/1 operating system, most applications will execute as traditional UNIX processes (a task with a single thread).

1.3 Memory Management

The DEC OSF/1 operating system uses a memory management system that is responsible for distributing the available main memory space among competing processes and buffers. You have some level of control over the following components of the memory management system:

- Virtual memory is used to enlarge the available address space beyond the physical address space. Virtual memory consists of main memory and swap space. The DEC OSF/1 operating system keeps only a set of the recently used pages of all processes in main memory and keeps the other pages on disk in swap space. Virtual memory and the unified buffer cache (UBC) share all physical memory. Virtual memory is discussed in Section 1.3.1.
- Paging and swapping is used to ensure that the active task has the pages in memory that are needed for it to terminate successfully. Paging is controlled by the page reclamation code. Swapping is controlled by the task swapping daemon. Paging and swapping are discussed in Section 1.3.1.1
- The I/O buffer cache is used to minimize the number of accesses to the disk during I/O operations. The I/O buffer cache serves as a layer between the file system on the disk and the operating system. The I/O

buffer cache is divided into the buffer cache region and the unified buffer cache (UBC).

The I/O buffer cache contains file metadata (superblocks, inodes, indirect blocks, directory blocks, and cylinder group summaries). The UBC holds the actual file data for reads and writes and mapped file regions. Virtual memory and the UBC share all physical memory. The UBC is discussed in Section 1.3.2

- Interprocess communication (IPC) is the mechanism that facilitates the exchange of information among processes. The IPC facilities include shared memory, pipes, semaphores, and messages. The IPC facilities are discussed in Section 1.4

The DEC OSF/1 operating system components constantly interact with each other. As a result, a change in one of the components can also affect the other. The following sections discuss each component in more detail.

1.3.1 Virtual Memory

The virtual memory subsystem supplies the initial values of pages and keeps track of the pages that have been paged out. Specifically, the virtual memory subsystem coordinates the allocation of resources for a task among the following hardware components (in the order of fastest to slowest access time):

1. Alpha AXP CPU cache – Internal instruction and data cache that resides in the CPU chip and ranges in size from 8KB to 64KB.
2. Secondary cache – Secondary direct-mapped physical data cache that is external to the CPU, but resides on the processor board. Block sizes for the secondary cache vary from 32 bytes to 256 bytes (depending upon processor type). Secondary cache ranges in size from 128KB to 8MB.
3. Tertiary cache – Same as the secondary cache (not available on all CPUs).
4. System memory – The actual physical memory.
5. Swap space – Block special device. Avoiding the file system saves overhead.

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

Much of the movement of addresses and data among the CPU cache, secondary and tertiary cache, and physical memory is controlled by the hardware logic and the PAL code, which is transparent to the DEC OSF/1 operating system. The virtual memory subsystem becomes involved when the CPU's translation buffer is unable to map a requested virtual address to a physical address and then traps to the PAL's page lookup code, which is

responsible for monitoring and loading addresses from the page table into the CPU's translation buffer.

If the requested address is in the page table, the PAL lookup code loads the address into the translation buffer, which in turn passes the address to the CPU. If the address is not in the page table, the PAL code issues a virtual memory fault, which is the virtual memory subsystem's cue to locate the page with the desired virtual contents and to load its physical address into the page table for use by the PAL lookup code:

- The virtual memory subsystem firsts looks in its internal data structures (for example, hash queue list and page queue list) for a page with the desired virtual contents. If it finds the page, it loads the physical address into the page table. This is known as a short fault.
- If the virtual memory subsystem cannot find the page in the internal data structures and if the requested page is new and has never been referenced, it initializes a physical page (the contents of the page are cleared) and loads the address into the page table. This is known as a zero-filled-on-demand fault.
- If the requested page has been referenced before, the virtual memory subsystem loads the address of the requested page into the translation buffer and performs a disk I/O to copy the contents of the page from swap space into memory. This is known as a page-in page fault.
- Finally, if the requested page is shared by a parent process and one or more child processes (using the `fork` function) and if one of the processes needs to modify the page, the virtual memory system loads a new address into the translation buffer, copies the contents of the requested page into the new address for modification by the process. This is known as a copy-on-write fault.

Page-in page faults and copy-on-write faults are handled by the virtual memory subsystem's paging and swapping mechanism that is described in Section 1.3.1.1.

The virtual memory subsystem attempts keep the movement of pages as fast as possible. To do this, it tracks the utilization and the location of all pages in the memory subsystem.

The virtual memory subsystem maintains five lists to perform its tasks. Each existing page can be found on one of the following lists:

- Free list – Pages that are clean and available for use
- Active list – Pages that are currently in use but can be used for paging
- Inactive list – Virtual memory pages that are allocated but are most likely to be reclaimed when memory is needed

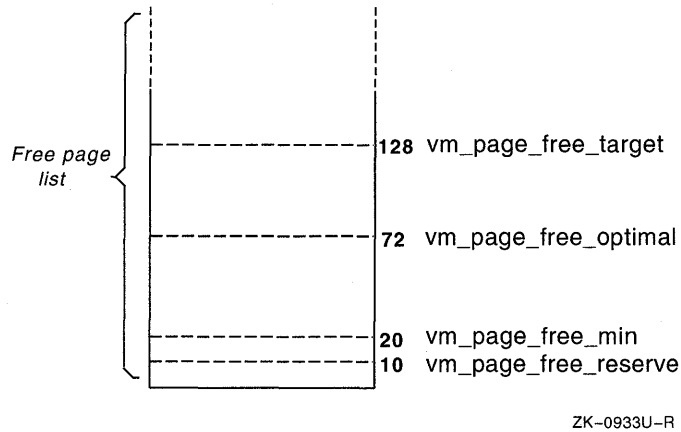
- UBC least recently used list (LRU list) – UBC pages that are allocated but most likely to be used for paging
- Wired list – Pages that are currently in use and cannot be reclaimed (not a real list)

The virtual memory subsystem tries to maintain a reasonable number of pages on the free page list so that pages will be available for use by processes. All pages are shared by virtual memory and the UBC. Four parameters define the size of the free page list and thus control when paging and swapping occur:

- `vm_page_free_min` – Minimum number of pages on the free page list before paging begins
- `vm_page_free_optimal` – Number of pages on the free page list at which task swapping is invoked
- `vm_page_free_target` – Number of pages on the free page list that must exist before paging stops
- `vm_page_free_reserve` – Absolute minimum number of pages on the free page list that can exist before only privileged tasks are able to get memory

Figure 1-1 shows the parameters that control paging and swapping:

Figure 1-1: Paging and Swapping Parameters



If the number of pages on the free page list falls below the value associated with the `vm_page_free_min` parameter, the virtual memory subsystem activates two page-stealer routines that reclaim the least recently used pages from the inactive or the UBC LRU list. This process continues until the

number of pages on the free page list reaches the value associated with the vm_page_free_target parameter. If necessary, the contents of the reclaimed pages are moved to swap space.

When the maximum number of pages is reached, the page-stealer daemon becomes dormant again. This procedure enables the virtual memory subsystem to keep the most recently used pages in memory and move the least recently used pages to swap space, where they can be easily accessed if necessary.

The vm_page_free_reserve parameter specifies the absolute minimum number of pages on the free page list. If the free page list falls below the value of the vm_page_free_reserve parameter, only privileged tasks can get memory, thus preventing deadlocks.

The page-stealer daemon maintains a ratio of one active page to two inactive pages. If the inactive list becomes too small, the page-stealer daemon deactivates pages that are the oldest and least recently used pages and moves them to the inactive list.

When the virtual memory subsystem maps an application into memory, it tries to anticipate which pages the task will need next. Using an algorithm that checks which pages were most recently used and the size of the free page list (as well as other factors), it passes some number of pages to the task in addition to the requested page. It tries to anticipate the pages that a task will need, thus accelerating the execution of the application by lowering the chances that a page fault will occur.

The virtual memory subsystem also attempts to optimize the utilization of the secondary cache. To do this, it uses a technique called page coloring. Essentially, it attempts to map the most recently referenced pages of a running task's virtual address space into the secondary cache once and execute the entire task, text, and data out of that cache. If the task is loaded in the secondary cache, the task does not have to fetch from physical memory and the task's execution time is decreased.

The virtual memory subsystem maintains system-wide counters for all the physical pages that it manages. The following counters (viewed with the vmstat command) track the overall use of physical memory:

- act – The total number of pages on the active list, the inactive list, and the UBC LRU list
- free – The physical pages not currently in use
- wired – The physical pages currently in use and not pageable

To determine how much memory an application uses, you can use the ps command. The ps command displays the virtual address size (VSZ), which is the total amount of virtual memory allocated to the process and its resident set size (RSS), which is the total amount of physical memory mapped to

virtual pages at some instance.

1.3.1.1 Paging and Swapping

Paging and swapping is the process of moving pages between memory and disk to ensure that a task has the pages in memory that it needs to run. The virtual memory subsystem controls this activity. It initiates paging and swapping activity under the following circumstances:

- Page in – A page in occurs strictly in the context of a task when it allocates the pages that it needs to execute. When you first execute a task, a page in occurs. If the address of a referenced page is not in the translation buffer or any of the internal data structures, the virtual memory subsystem must go to disk (either to swap space if the page has been referenced before or to the disk resident executable) to obtain the page and bring it into memory.

  To perform a page in, the virtual memory subsystem allocates a physical page off the free page list, which is a linked list of available pages. When it has the address, the virtual memory subsystem fills the physical page with the contents of the page that it obtained from disk, loads the physical address into the page table, and marks the page as active.

- Page out – A page out occurs when the average number of pages in the free page list falls below the value associated with the `vm_page_free_min` parameter. The page reclamation code activates the page-stealer daemon, which performs the following tasks:

  1. Takes the least recently used and lowest priority pages from the inactive and UBC LRU lists.
  2. Activates the code that moves the contents of those selected pages that are dirty to swap space (note that clean pages already have copies in swap space).
  3. Places those pages on the free page list until the average number of pages on the free page list reaches the value associated with the `vm_page_free_target` parameter.

- Swap out – A swap out occurs when the page-stealer daemon cannot keep up with the demand for free pages. This indicates that the system does not have enough memory to execute its processes. The swap out procedure dramatically increases the number of pages on the free page list and reduces the demand for physical memory by suspending swapped out tasks.

  If the average number of pages in the free page list falls below the value of the `vm_page_free_optimal` parameter for more than five seconds, the task swapper (an extension of the page reclamation code) is activated. The task swapper thread suspends processes that have a low

priority and a high resident set size, writes to disk the contents of all the dirtied pages associated with the suspended processes, and places those pages on the free page list. Suspending low-priority, memory-intensive processes decreases the demand for pages. Processes continue to be swapped out until the free page list reaches the value of the `vm_page_free_target` parameter.

- Swap in – A swap in occurs when a swapped out task becomes runnable and the number of pages on the free page list reaches an adequate level (above the value of the `vm_page_free_optimal` parameter for a period of time). This enables a task that was swapped out (suspended) to once again be able to execute. The task will then begin to page in its resident set.

From a performance viewpoint, swapping is worse than paging because swapped out processes can experience a long latency that is unsuitable for interactive processes. In addition, swapping can reduce system throughput. However, swapping does move long-sleeping threads out of memory and thus ''cleans up'' memory.

1.3.1.2 Swap Buffers

To facilitate the movement of data between memory and disk, the virtual memory subsystem uses two types of swap buffers: synchronous and asynchronous.

- Synchronous swap buffers are used by the page-in code for synchronous page ins and by the task swapper for synchronous swap outs. The `syncswapbuffers` parameter specifies the number of synchronous swap I/O requests that the individual process page ins and the task swapper can have outstanding to the I/O subsystem at any one time. This value should be roughly equivalent to the number of simultaneously running processes that the system can easily handle.
- Asynchronous swap buffers are used by the page-stealer daemon for asynchronous page outs. The `asyncswapbuffers` parameter specifies the number asynchronous swap I/O requests that the page-stealer daemon can have outstanding at any one time. This number should be roughly equivalent to the number of I/O transfers the swap devices can handle.

The virtual memory subsystem uses the two swap buffers to satisfy the immediate demands of a page-in request without having to wait for the relatively slow process of a page out, that is, writing a page to disk.

1.3.1.3 Swap Space Allocation Modes

How swap space is allocated is determined by two modes: immediate mode and deferred (or over-commitment) mode. The two strategies differ in the point in time at which swap space is allocated.

- In immediate mode, swap space is allocated when modifiable virtual address space is created.
- In deferred mode, swap space is not allocated until the system needs to write a modified virtual page to swap space.

The DEC OSF/1 operating system's default swap mode is immediate mode. The operating system will reserve swap space for anonymous memory (for example, stack space and memory allocated by the `malloc` routine) when that memory is allocated. This results in more swap space being reserved than is probably required.

You can set the default swap mode to deferred (or over-commitment) mode. This causes the reservation and allocation of swap space used to back up anonymous memory to be postponed until the physical memory actually needs to be reclaimed.

Deferred mode requires less swap space and causes the system to run faster than if you used immediate mode because less swap bookkeeping is required. However, because deferred mode does not reserve swap space in advance, the swap space may not be available when it is needed by a task and the process may be killed asynchronously. You should ensure that you have sufficient swap space if you want to use deferred mode.

Immediate swap mode is used if the `/sbin/swapdefault` file exists. This file is a symbolic link to `/dev/rzxx`, which is the first defined swap device. If this file does not exist, the system uses deferred mode. If you change from one mode to another, you must reboot the system to activate the new mode.

Refer to the manual *System Administration* for more information on swap space allocation modes.

1.3.2 Unified Buffer Cache

The DEC OSF/1 operating system uses a unified buffer cache (UBC) to hold the actual file data, which includes reads and writes from conventional file activity and page faults from mapped file sections. The UBC and the virtual memory subsystem share all memory and utilize the same physical pages. This means that all of physical memory can be used for buffering both I/O and processes address space. A traditional buffer cache area is used for file system metadata (for example, file header information, blocks, directories, and inodes), but the data associated with the file is stored in the UBC.

The UBC uses a buffer to facilitate the movement of data between memory and disk. The `ubcbuffers` configuration file parameter specifies the number of UBC I/O requests that can be outstanding.

The UBC is dynamic, and it can potentially utilize all physical memory; thus the UBC can respond to changing file system demands. You can limit the amount of memory allocated to the UBC. The `ubcmaxpercent` configuration file parameter specifies the maximum percentage of memory that the UBC can utilize. The `ubcminpercent` configuration file parameter specifies the minimum percentage of memory that the UBC will be trimmed down to when page reclamation occurs.

Changes in relative rates of demand can enlarge or shrink the size of the UBC. Heavy virtual memory activity – for example, large increases in the working set caused by large executable files or by large amounts of uninitialized data (BSS) being accessed – will increase the number of pages reserved for virtual memory and decrease the number reserved for the UBC. Heavy file system activity will increase the number of pages reserved for the UBC and decrease the number of pages reserved for virtual memory.

1.4 Interprocess Communications Facilities

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.

In single-process programming, modules within a single process communicate with each other using global variables and function calls, with data passing between the functions and the callers. When programming using separate processes, with images in separate address spaces, you need to use additional communication mechanisms.

The DEC OSF/1 operating system provides the following facilities that are used in interprocess communication:

- System V IPC – System V IPC includes the following IPC facilities: messages, shared memory, and semaphores. See the *System V Compatibility User's Guide* for information about System V IPC.
- 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 *System V Compatibility User'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.

1.5 I/O Subsystems

The I/O subsystems involve the software and hardware that performs all reading and writing operations. The software includes device drivers, file systems, and networks. The hardware portion includes all peripheral equipment (for example, disks, tape drives, printers, and network and communication lines).

The following sections describe the various I/O subsystems.

1.5.1 Disks

The DEC OSF/1 operating system supports two hardware storage architectures: Small Computer System Interface (SCSI) and Digital Storage Architecture (DSA).

All Alpha AXP systems support SCSI devices. This support is provided through the Common Access Method (CAM) architecture. The CAM architecture defines a software model that is layered, providing hardware independence for SCSI device drivers. In the CAM model, a single SCSI/CAM peripheral driver controls SCSI devices of the same type, for example, direct access devices. This driver communicates with a device on the bus through a defined interface. Using this interface makes a SCSI/CAM peripheral device driver independent of the underlying SCSI Host Bus Adapter.

This hardware independence is achieved by using the Transport (XPT) and SCSI Interface Module (SIM) components of CAM. Because the XPT/SIM interface is defined and standardized, users and third parties can write SCSI/CAM peripheral device drivers for a variety of devices and use existing operating system support for SCSI. The drivers do not contain SCSI HBA dependencies; therefore, they can run any hardware platform that has an XPT/SIM interface present.

The Digital Storage Architecture (DSA), supported only on DEC 7000-series systems, conforms to the Mass Storage Control Protocol (MSCP).

1.5.2 File Systems

The DEC OSF/1 operating system file system architecture is based on the OSF/1 Virtual File System (VFS), which is based on Berkeley 4.3 Reno VFS. VFS provides an abstract layer interface into files regardless of the file

systems in which the files reside. Included in VFS is the `namei` cache, which stores 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 to perform pathname translations.

Layered below VFS, the DEC OSF/1 operating system supports the following file systems:

- Berkeley UNIX File System (UFS), which is the operating system's native file system.
- Network File System (NFS), which allows users to mount remote file systems in their own local directories.
- Memory File System (MFS), which is memory-based UFS. Data resides entirely in memory instead of on disk. The contents of an MFS file system are lost after a reboot, unmount operation, or power failure.
- Advanced File System (AdvFS), which is a local file system that uses write-ahead logging to provide rapid crash recovery. AdvFS ensures that file structures are recovered consistently and offers a flexible structure. Also available for AdvFS users is the POLYCENTER Advanced File System Utilities layered product. POLYCENTER provides utilities specifically designed for performance tuning, such as file striping and defragmenting.

File systems use the UBC to avoid disk I/O. Because of this, I/O accesses may appear random. The UBC shares all of memory with the virtual memory subsystem. The UBC adjusts itself dynamically to accommodate varying I/O loads. As the I/O load increases, the UBC increases to the limit defined by `ubcmaxpercent` configuration file parameter. All I/O passes through the UBC and is flushed to disk by the `update` daemon.

Laying out your file system tree across multiple disks can improve performance. The access time, which is the disk latency or seek time (the time the disk takes to access the requested block), tends to be more important than the transfer rate for most workstation, time-share, and server environments.

You can modify file system fragment sizes to optimize either I/O performance or disk space usage. Large fragment sizes optimize for I/O performance, and small fragment sizes optimize for disk space usage.

An important feature in UNIX file system performance is UFS block clustering. Essentially, block clustering causes the file system and the UBC to combine multiple small I/O operations into a larger single I/O operation to disk. This results in a dramatic decrease in read/write requests to disk.

With block clustering, performance is based on hardware speeds (disk and controller). Performance is achieved by taking better advantage of controller

and drive hardware. Clustering reduces the number of trips to the disk, which reduces kernel overhead. With clustering, I/O can nearly attain raw device bandwidth for sequential operations. For example, read/write speed of large files is 95 percent or more of disk subsystem performance (sequential access).

Clusters are groups of file system blocks in a contiguous sequence. For a standard 8KB/1KB (block size/fragment size) UFS file system, the default cluster size is 8 blocks (64KB). This is determined by multiplying the default number of blocks (8) by the block size (8192 bytes). You can modify the the number of blocks that are combined into a single read request by using either the `tunefs` or `newfs` command to establish a new value for `maxcontig`. You can modify the number of blocks that are combined into a single write request by using `dbx` to establish a new value for the `cluster_maxcontig` global parameter. The default value of `cluster_maxcontig` is 8, which tries to combine eight 8KB blocks (or 64KB) into a single write.

UFS tries to group contiguous writes into clusters. Individual contiguous block writes are collected into a cluster. The cluster is written asynchronously as a unit either when its full size is reached or a discontiguous block is encountered. Specifically, contiguous writes are done in 64KB units, which is the file system block size (8) multiplied by the default value of `cluster_maxcontig` (8).

UFS uses clusters to make read-ahead more efficient and effective as follows:

- Initial read brings in two blocks: one synchronous, one asynchronous (read-ahead block).
- Subsequent contiguous reads trigger exponential read-ahead in cluster units to a maximum number of eight clusters. (The default is 8, which is an exponent of the `cluster_max_read_ahead` global parameter, which specifies the maximum number of clusters to stay ahead of sequential reader. The `cluster_max_read_ahead` global parameter can be changed by using `dbx`.)
- The first noncontiguous read causes read-ahead to reset its counters.
- The maximum amount of read-ahead is determined by the value of the file system block size multiplied by the value of `maxcontig` multiplied by the value of the `cluster_max_read_ahead` parameter (for example, 8192 * 8 * 8 = 512KB).
- The read-ahead policy is more aggressive than traditional UFS in that it attempts to anticipate the needs of contiguous reads. This policy balances sequential performance with safeguards for cache flushing.

1.5.3 Network Systems

A network provides a means to move data from one device to another. This data may be no more complicated than electronic mail. You can copy files containing printable data (for example, word processor files), or binary data from a local computer to a remote computer, with the same ease as files copied from one directory to another on the local computer. With remote login, users can login to a remote computer on which they have an account and access programs and data as if they were at a terminal connected to their own host computer.

A network consists of two essential component parts: the hardware implementation and the software that runs the network. The hardware consists of controllers and connectors.

1.5.3.1 Network Hardware

The controller sends and receives packets of data over the network. Controllers are specialized and are designed to work with a particular type of computer (bus architecture). For example, controllers designed to work with a Digital workstation will not work with a Sun or Hewlett-Packard workstation, or an IBM-PC, and vice versa.

The cables or wires connecting different computers (or nodes) on a network can be twisted-pair (as with telephone wires), thick or thin Ethernet cable, or optical fiber. The type of controller determines the type of connector.

1.5.3.2 Network Software

Two relevant network software implementations exist for the DEC OSF/1 operating system: TCP/IP and DECnet. Their names refer to the protocol used to send information from one network node to another, as well as to the software written to implement these protocols, which are the rules and formats that conduct communications on a network. Protocols govern the way messages are packaged, addressed, and routed; master/slave relationships among network nodes; polling; the exchange of control information; and the hierarchy. The following paragraphs briefly introduce TCP/IP and DECnet:

- TCP/IP (Transmission Control Protocol/Internet Protocol) was developed by the U.S. Defense Department, Defense Advanced Research Projects Agency (DARPA). Its name is derived from its two main standards. Because TCP/IP is a collection of protocols, rather than a particular software program, the software that provides its services has been implemented for many different hardware platforms and operating systems. The TCP/IP protocols are used as building blocks on which other products or applications are built. As a result, it is a widely accepted standard in the UNIX world.

- DECnet is Digital Equipment Corporation's implementation of a network protocol common to its operating systems. The advantage of DECnet over TCP/IP in a Digital environment is that Digital controls its development and distribution and is able to optimize the implementation for its own hardware and systems software. However, from the user's point of view, DECnet and TCP/IP accomplish the same tasks. Because both can coexist on the same physical network, both can be present and can be used whenever their application is warranted.

Network File System (NFS) is a product that utilizes TCP/IP and was built originally in a Berkeley UNIX environment. It is a proprietary product developed by Sun Microsystems; however, it has now been ported and licensed on many other UNIX implementations, including the DEC OSF/1 operating system.

NFS allows users to mount remote file systems in their own local directories, thereby giving the appearance of an extension of their local file system. The machine that offers file systems for other machines to access is called the server or file server; the machines that access these file systems by remotely mounting them are called clients.

NFS, however, is not a network extension of UNIX and does not adhere to UNIX semantics. It does not support all UNIX file system operations, does not guarantee atomic operations, and cannot obtain access to remote devices. It operates independently of the machine and operating system and can be used on non-UNIX machines as well as those running UNIX.

The User Datagram Protocol (UDP) is commonly used in the Network File System. UDP is the internet standard protocol that allows an application program on one host to send a datagram to an application program on another host. UDP provides an unreliable, connectionless delivery service using IP to transport messages among hosts.

UDP is similar to TCP and it provides a mechanism for user applications to communicate with IP. UDP differs from TCP in that it is a simple protocol that is entirely dependent upon IP's best effort to provide reliability. UDP does not guarantee delivery, occasionally generates duplicate data packets, and may send data in the incorrect order. However, layers above UDP can create reliable services using UDP.

Both the host (client) and remote (server) machines start network daemon processes running when they are booted. Machines that can be reached from the network are listed in a data file with their network addresses. Each local machine knows its own name and network address. As data is sent out over the network, the address and routing information are filled in by the sending network daemon. Network daemons on receiving machines decode the address to determine for whom the message is intended. If the message is intended for the receiving machine, it decodes the message and processes it; otherwise, it does nothing.

Monitoring Your System 2

Before you start to monitor your system to identify a performance problem, you should understand your user environment, the applications you are running and how they use the various subsystems, and what is acceptable performance.

The source of the performance problem may not be obvious. For example, if your disk I/O subsystem is swamped with activity, the problem may be in either the virtual memory subsystem or in the disk I/O subsystem. In general, obtain as much information as possible about the system before you attempt to tune it.

In addition, how you decide to tune your system depends on how your users and applications utilize the system. For example, if you are running a lot of CPU-intensive applications, the virtual memory subsystem may be more important than the file system buffer cache (UBC).

This chapter contains the following information:

- A brief introduction to the tools that you can use to monitor your system (Section 2.1)
- Examples of how to use some of the tools to perform a variety of monitoring tasks (Section 2.2)

2.1 Monitoring Tools

Numerous system monitoring tools are available. You may have to use various tools in combination with each other in order to get an accurate picture of your system. In addition to obtaining information when the system is running poorly, it is also important for you to obtain information about your system when it is running well. By comparing the two sets of data, you will be able to pinpoint the area that is causing the performance problem.

The primary monitoring tools are described in Table 2-1.

Table 2-1: Primary Monitoring Tools

Tool	Description
`iostat`	Reports I/O statistics for terminals, disks, and the system. See Section 2.2.5 for more information on using the `iostat` command to diagnose system performance problems.
`netstat`	Displays network statistics. The `netstat` command symbolically displays the contents of network-related data structures. Depending on the options supplied to `netstat`, the output format will vary. The more common format is to supply the `netstat` command with a time interval to determine the number of incoming and outgoing packets, as well as packet collisions, on a given interface. See Section 2.2.8 for more information on using the `netstat` command to diagnose system performance problems.
`nfsstat`	Displays Network File System (NFS) and Remote Procedure Call (RPC) statistics for clients and servers. The output includes 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`). See Section 2.2.9 for more information on using the `nfsstat` command to diagnose system performance problems.
`ps`	Displays the current status of the system processes. Although `ps` is a fairly accurate snapshot of the system, it cannot begin and finish a snapshot as fast as some processes change state. As a result, the output may contain some inaccuracies. The `ps` command includes information about how the processes use the CPU and virtual memory. See Section 2.2.1 for more information on using the `ps` command to diagnose system performance problems.
`uptime`	Shows how long a system has been running and the system load average. The load average numbers give the number of jobs in the run queue for the last 5 seconds, the last 30 seconds, and the last 60 seconds. See Section 2.2.2 for more information on using the `uptime` command to diagnose system performance problems.
`vmstat`	Shows information about process threads, virtual memory, interrupts, and CPU usage for a specified time interval. See Section 2.2.3 for more information on using the `vmstat` command to diagnose system performance problems.

Other tools can also provide you with important monitoring information. These secondary monitoring tools are described in Table 2-2.

Table 2-2: Secondary Monitoring Tools

Tool	Description
dbx	Analyzes running kernels and dump files. The dbx command invokes a source-level debugger. You can use dbx with code produced by the cc and as compilers and with machine code. After invoking the dbx debugger, you issue dbx commands that allow you to examine source files, control program execution, display the state of the program, and debug at the machine level. To analyze kernels, use the –k option. See Section 2.2.7 for more information on using the dbx command to diagnose system performance problems.
dumpfs	Displays UFS file system information. This command is useful for getting information about the file system block and fragment size and the minimum free space percentage. See Section 2.2.6 for more information on using the dumpfs command to diagnose system performance problems.
ipcs	Reports interprocess communication (IPC) statistics. The ipcs command displays information about currently active messages queues, shared memory segments, semaphores, remote queues, and local queue headers. Of specific interest are the following fields: - QNUM, the number of messages currently outstanding on the associated message queue - CBYTES, the number of bytes in messages currently outstanding on the associated message queue - QBYTES, the maximum number of bytes allowed in messages outstanding on the associated message queue - SEGSZ, the size of the associated shared memory segment - NSEMS, the number of semaphores in the set associated with the semaphore entry See ipcs(1) for details.
kdbx	Analyzes running kernels and dump files. The kdbx debugger is an interactive program that lets you examine either the running kernel or dump files created by the savecore utility. In either case, you will be examining an object file and a core file. For running systems, these files are usually /vmunix and /dev/mem, respectively. Dump files created by savecore are saved in the directory specified by the /sbin/init.d/savecore script which is, by default, /var/adm/crash. All dbx commands are available with kdbx using the dbx option. See the manual *Kernel Debugging* or kdbx(8) for details.

Table 2-2: (continued)

Tool	Description
`nfswatch`	Monitors all NFS network traffic and divides it into several categories. The number and percentage of packets received in each category is displayed on the screen in a continuously updated display. Your kernel must be configured with the `packetfilter` option. See `nfswatch`(8) and `packetfilter`(7) for details.
`prof` and `pixie`	Displays statistics on where time is being spent – at the routine level, basic block level, or instruction level – during the execution of a program. This information will help you to determine where to concentrate your efforts to optimize source code.
`showfdmn`	Displays the attributes of an AdvFS file domain and detailed information about each volume in the file domain.
`showfile`	Displays the full storage allocation map (extent map) for files in an Advanced File System (AdvFS). An extent is a contiguous area of disk space that the file system allocates to a file.
`showfsets`	Displays the filesets (or clone filesets) and their characteristics in a specified domain.
`swapon`	Specifies additional disk space for paging and swapping and displays swap space utilization, including the total amount of allocated swap space, the amount of swap space that is being used, and the amount of free swap space. See Section 2.2.4 for more information on using the `swapon` command to diagnose system performance.
`tcpdump`	Displays network traffic. The `tcpdump` command prints out the headers of packets on a network interface that match the boolean expression. Your kernel must be configured with the `packetfilter` option. See `tcpdump`(8) and `packetfilter`(7) for details.
`w`	Displays a summary of current system activity. The system summary shows the current time, the amount of time since the system was last started, the number of users logged in to the system, and the load averages. The load average numbers give the number of jobs in the run queue for the last 5 seconds, the last 30 seconds, and the last 60 seconds. See `w`(1) for details.
`xload`	System load average display for X. The `xload` command displays a periodically updating histogram of the system load average. See `xload`(1X) for details.

In addition, a number of unsupported monitoring tools are available through the Internet. These tools include the monitoring tools described in Table 2-3.

Table 2-3: Unsupported Monitoring Tools

Tool	Description
`monitor`	Displays CPU, file system, memory, swap I/O, and network statistics. (Available on the Internet at `gatekeeper.dec.com`.)
`xcpustate`	Displays CPU states (idle, nice, system, kernel) statistics. The `xcpustate` utility displays bars showing the percentage of time the CPU spends in different states. (Available on the Internet at `gatekeeper.dec.com`.)
`xnfs`	Displays NFS statistics. The `xnfs` utility displays bars showing the number of active NFS daemons, the number of `mbuf` write, the number of cluster writes, and the number of hits and misses. (Available on the Internet at `gatekeeper.dec.com`.)

2.2 Determining the Problem

The following sections describe how to use the monitoring tools to identify the system component or subsystem that is causing the performance degradation. Once you determine which subsystem or component is causing the problem and you are sure that you understand your system environment and the needs of your users, refer to the appropriate section in Chapter 3 for information on tuning the particular subsystem or component.

2.2.1 Monitoring Processes – ps Command

The `ps` command displays the current status of the system processes. You can use it to determine the current running processes, their state, and how they utilize system memory. The command lists processes in order of decreasing CPU usage, so you can easily determine the processes that are using the most CPU time. Be aware that `ps` is only a snapshot of the system; by the time the command finishes executing, the system state has probably changed. For example, one of the first lines of the command may refer to the `ps` command itself.

An example of the ps command follows:

```
# ps aux
USER  PID  %CPU %MEM   VSZ   RSS  TT  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
sorn  2228  0.0  0.5  1.82M  504K p5  S  + 13:25:03  0:00.02  more
sorn  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]
```

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

[1] Percent CPU usage (%CPU).

[2] Percent real memory usage (%MEM).

[3] Process virtual address size (VSZ) – This is the total amount of virtual memory allocated to the process.

[4] Real memory (resident set) size of the process (RSS) – This is the total amount of physical memory mapped to virtual pages (the amount of memory the application has physically used).

[5] Process status or state (S) – This specifies whether a process is runnable (R), sleeping (S), idle (I), stopped (T), halted (H), or swapped out (W). It also indicates whether the process priority has been reduced (N) or raised (+) with the nice or renice command.

[6] Current CPU time used (TIME).

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

- If a process is using a lot of memory (see the RSS and VSZ fields), the process could have a problem with memory usage.
- Are duplicate processes running? Use the kill command to terminate any unnecessary processes.
- If a process is using a lot of CPU time, it may be in an infinite loop and require changes to its source code.

  If a process using a lot of CPU time is running correctly, you may want to lower its priority with either the nice or renice command. Note that these commands have no effect on a process that is using a lot of memory.
- 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 virtual memory problem.

For information about improving the performance of your applications, see Section 3.2. For information about improving your virtual memory performance, see Section 3.4.

2.2.2 Measuring the System Load – 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 also applications whose priorities have been changed with either the `nice` or `renice` command. The load average numbers give the 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 follows:

```
# uptime
1:48pm  up 7 days,  1:07,  35 users,  load average: 7.12, 10.33, 10.31
```

Note 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 applications during off-peak hours. You can also lower the priority of applications with the `nice` or `renice` command to conserve CPU cycles.

See Section 3.3 for additional information on how to reduce the load on your system.

2.2.3 Monitoring Virtual Memory and CPU Usage – 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. Because the CPU operates faster than the rest of the system, performance bottlenecks usually exist in the memory or I/O subsystems.

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

```
# 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. For example, the following fields are important:

1 Virtual memory information, including the number of pages that are active (`act`), the number of pages on the free list (`free`), and the number of pages wired down (`wire`). See Section 1.3.1 for more information.

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

3 Interrupt information, 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, including the percentage of used time for normal and priority processes (`us`), the percentage of system time (`sy`), and the percentage of idle time (`id`).

While diagnosing a bottleneck situation, keep the following issues in mind:

- Is the system demand valid? That is, is there something different in your system that is adversely affecting the environment, such as a new process or additional users?
- 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 (`wired`). The sum of the free, active, and wired pages should be close to the amount of physical memory that you have. If the value for `free` is less than 100, you may have a virtual memory problem. Swapping may begin when the free page list is less than 128.
- Examine the `pout` field. If a lot of page outs are occurring, you could have a virtual memory problem; you are using more virtual space than you have physical space. You could also have insufficient swap space or your swap space could be configured inefficiently. Use the `swapon -s` command to display your swap device configuration and the `iostat` command to determine which disk is being used the most.

- Examine the device interrupts (`in`), the number of system calls called per second (`sy`), and the number of task and thread context switches per second (`cs`).

  In general, systems become saturated when they are processing at the following rates:

  - More than 1,000 device interrupts per second
  - More than 10,000 system calls per second
  - More than 1,000 context switches per second

  If the values displayed are hovering around these values, then something in the application load is causing high overhead operations. The source could be high system call frequencies, high interrupt rates, large numbers of small I/O transfers, or large numbers of small IPC or network transfers.

  - If you have a high interrupt rate, the application may require the system to perform certain operations, for example, heavy floating-point emulation or major unaligned accesses. Major unaligned accesses usually result in a message being displayed.

    Note that failing or misconfigured hardware can produce high system time and a high device interrupt rate. Use the `uerf` event report formatter to ensure that the hardware is working properly.

  - If you have a large system call rate, use the `dbx` debugger to examine the `sccount` structure to determine the number of times a system call is issued. This structure is an array that counts each time a system call is made since the last system boot.

  - If you have a high number of context switch interrupts, your application may be performing large numbers of small I/O transfers or large numbers of small IPC or network transfers. Use the `dbx` debugger to examine the `sccount` structure to determine the number of system calls related to IPC messages and semaphores.

    In general, small I/O transfers (for example, I/O chunks less than 8KB for file system and 1KB for network) cause the system to perform a lot of I/O overhead to move each small chunk. If you can buffer the I/O to 8KB multiples for disk I/O or 1KB for IPC or network I/O, you dramatically increase the efficiency of the I/O.

- 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 presents information on how to interpret the values for user, idle, and system time:

- A high user time and a low idle time could indicate that your application code is consuming most of the CPU. You can optimize the application, or you may need a more powerful processor.
- A high system time and low idle time could 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. Use `dbx`'s `sccount` structure to check the rate of system calls.

  Note that a high system time and low idle time could be caused by failing hardware. Use the `uerf` command to check your hardware.

  A high system time could 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 may be doing a lot of I/O, so this could indicate a virtual memory problem.
- In may cases, if the idle time is very low, your system is utilizing its CPU efficiently.

  If you have a high idle time and you are sure that your system has a typical load, one or more of the following problems may exist: the hardware may be saturated (bus bandwidth, arm motion, CPU cycles, cache thrashing), one or more kernel data structures is being exhausted, or you may have a hardware or kernel resource block such as an application, I/O, or network bottleneck.

See Chapter 3 for information on improving CPU usage and I/O operations and for information on tuning virtual memory, disks, and file systems.

2.2.4 Displaying the Swap Space Configuration – swapon Command

Use the `swapon` command with the `–s` option 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 should help you determine how your swap space is being utilized.

For example:

```
# swapon -s
Total swap allocation:
    Allocated space:        54912 pages (429MB)
    Reserved space:          5756 pages ( 10%)
    Available space:        49156 pages ( 89%)

Swap partition /dev/rz1g:
    Allocated space:        27456 pages (214MB)
    In-use space:            1475 pages (  5%)
    Free space:             25981 pages ( 94%)

Swap partition /dev/rz2g:
    Allocated space:        27456 pages (214MB)
    In-use space:            1459 pages (  5%)
    Free space:             25997 pages ( 94%)
```

See Section 3.4.3 for information on how to tune your swap space configuration. Use the `iostat` command to determine which disks are being used the most.

2.2.5 Monitoring Disk I/O – iostat Command

The `iostat` command reports I/O statistics for terminals, disks, and the CPU. The first line of the output is the average since boot time, and each subsequent report is for the last interval. An example of the `iostat` command is as follows:

```
# iostat 1
      tty      rz1       rz2       rz3       cpu
 tin tout bps tps  bps tps  bps tps  us ni sy id
  0    3   3   1    0   0    8   1   11 10 38 40
  0   58   0   0    0   0    0   0   46  4 50  0
  0   58   0   0    0   0    0   0   68  0 32  0
  0   58   0   0    0   0    0   0   55  2 42  0
```

The `iostat` command reports I/O statistics that you can use to diagnose disk I/O problems. For example, the command displays information about the following:

- For each disk, (`rzn`), the number of bytes (in thousands) transferred per second (`bps`) and the number of transfers per second (`tps`). Some disks report the milliseconds per average seek (`msps`).
- For the system, the percentage of time the system has spent in user state running processes either at their default priority or higher priority (`us`), in user mode running processes at a lowered priority (`ni`), in system mode (`sy`), and idle (`id`). This information enables you to determine how disk I/O is affecting the CPU.

Note the following when you use the `iostat` command:

- Determine which disk is being used the most and which is being used the least. The 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 a disk is doing a lot 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 for a small amount of data. You may want to rewrite the application if this behavior is not necessary.

See Section 3.6 for information on how to improve your disk I/O performance.

2.2.6 Displaying UFS File System Information – dumpfs Command

The `dumpfs` command dumps UFS file system information. The command prints out the super block and cylinder group information. The command is useful for getting information about the file system block and fragment sizes and the minimum free space percentage.

The following example shows part of the output of the `dumpfs` command:

```
# dumpfs /dev/rrz3g | more
magic   11954   format  dynamic time    Tue Sep 14 15:46:52 1993
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.
- `fsize` – The fragment size of the file system in bytes.
- `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.
- `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.

Keep the following issues in mind:

- A large block size (`bsize`) benefits large I/O transfers but can waste disk space. A small block size uses the disk efficiently but requires more I/O operations. Note that the UFS file system block size is fixed at 8KB.
- For the optimum I/O performance, the fragment size (`fsize`) can be the same as the block size.
- If `rotdelay` is zero, blocks are allocated contiguously.
- A large value for `maxbpg` can improve performance for large files.

For information about tuning file system parameters to improve your disk I/O performance, see Section 3.6.1.1 (UFS) or Section 3.6.1.2 (AdvFS).

2.2.7 Monitoring AdvFS

There are a number of commands that you can use to display information about AdvFS.

The `showfile` command displays the full storage allocation map (extent map) for files in an Advanced File System (AdvFS). An extent is a contiguous area of disk space that the file system allocates to a file. The following example of the `showfile` command displays the AdvFS-specific attributes for all the files in the current working directory:

```
# showfile *

        Id  Vol  PgSz  Pages  XtntType  Segs  SegSz  Log  Perf  File
    22a.001    1    16      1    simple    **     **  off   50%  Mail
      7.001    1    16      1    simple    **     **  off   20%  bin
    1d8.001    1    16      1    simple    **     **  off   33%  c
   1bff.001    1    16      1    simple    **     **  off   82%  dxMail
    218.001    1    16      1    simple    **     **  off   26%  emacs
    1ed.001    1    16      0    simple    **     **  off  100%  foo
    1ee.001    1    16      1    simple    **     **  off   77%  lib
    1c8.001    1    16      1    simple    **     **  off   94%  obj
    23f.003    1    16      1    simple    **     **  off  100%  sb
   170a.008    1    16      2    simple    **     **  off   35%  t
      6.001    1    16     12    simple    **     **  off   16%  tmp
```

The following example of the `showfile` command shows the attributes and

extent information for the `mail` file, which is a simple file:

```
# showfile -x mail

        Id  Vol   PgSz  Pages  XtntType  Segs  SegSz  Log  Perf  File
 4198.800d    2     16     27    simple    **     **  off   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
```

See `showfile`(8) for information about the output of the 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:

```
% showfdmn usr

               Id              Date Created  LogPgs  Domain Name
2b5361ba.000791be  Tue Jan 12 16:26:34 1993     256  usr

Vol   512-Blks      Free  % Used  Cmode  Rblks  Wblks  Vol Name
 1L     820164    351580     57%     on    256    256  /dev/rz0d
```

See `showfdmn`(8) for information about the output of the command.

The `showfsets` command displays the filesets (or clone filesets) and their characteristics in a specified domain.

The following is an example of the `showfsets` command:

```
# 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 output of the command.

See Chapter 3 for information about tuning AdvFS.

2.2.8 Using dbx to Monitor Subsystems

You can use dbx to examine source files, control program execution, display the state of the program, and debug at the machine code level. You use the dbx print command to examine the values of variables and data structures.

To examine a running system with dbx, use the following command:

```
# dbx -k vmunix /dev/mem
```

The following sections describe how to use dbx to examine various subsystems of the DEC OSF/1 operating system.

2.2.8.1 Checking Virtual Memory with dbx

If the vmstat command shows a large system call rate, use the dbx print command to look at the sccount structure. This is an array that counts each time a system call has been made since the last system boot.

```
(dbx) p sccount
struct {
    [0] 0               -nosys (always yields an error)
    [1] 0               -exit
    [2] 112706          -fork
    [3] 119265107       -read
    [4] 14257390        -write
    [5] 0               -old open, unused
    [6] 5251226         -close
    [7] 147732          -wait4
   .
   .
   .
(dbx)
```

The left column of the display lists the number of the system call as defined in the file /usr/sys/include/sys/syscall.h and the right column is the number of times that system call has been issued. From this output, you can determine the most frequently made system calls. This will help you understand what is really going on in your system and whether it matches your expectations.

If you are surprised by the numbers, examine the source code for the application to determine whether you can make the application more efficient. See Section 3.2 for information on how to tune an application.

You can also check virtual memory by using dbx and examining the vm_perfsum structure. Note the vpf_pagefaults field (number of

hardware page faults) and the vpf_swapspace field (number of pages of swap space not reserved).

```
(dbx) p vm_perfsum
struct {
     vpf_pagefaults = 6732100
  .
  .
  .
     vpf_swapspace = 29230
}
(dbx)
```

See Section 3.4 for information on how to tune the virtual memory subsystem.

2.2.8.2 Checking UFS with dbx

To check UFS using dbx, examine the ufs_clusterstats structure to see how efficiently the system is performing cluster read and write transfers. You can examine the cluster reads and writes separately with the ufs_clusterstats_read and ufs_clusterstats_write structures.

The following example shows a system that is not clustering efficiently:

```
(dbx) p 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)
```

This example shows 24644 single-block transfers and no 9-block transfers.

See Section 3.6.1.1 for information on how to tune the UFS file system.

2.2.8.3 Checking the namei Cache with dbx

The namei cache stores 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 UFS namei cache, use dbx and look at the nchstats data structure. In particular, look at the ncs_goodhits, ncs_neghits, and ncs_misses 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, and ncs_misses.)

For example:

```
(dbx) p nchstats
struct {
    ncs_goodhits = 9748603    -found a pair
    ncs_neghits = 888729      -found a pair that didn't exist
    ncs_badhits = 23470
    ncs_falsehits = 69371
    ncs_miss = 1055430        -did not find a pair
    ncs_long = 4067           -name was too long to fit in the cache
    ncs_pass2 = 127950
    ncs_2passes = 195763
    ncs_dirscan = 47
}
(dbx)
```

For information on how to improve the namei cache hit rate, see Section 3.6.1.

2.2.8.4 Checking the UBC with dbx

To check the UBC, use dbx to examine the vm_perfsum structure. In particular, look at the vpf_piowrites field (number of I/O operations for page outs generated by the page stealing daemon) and vpf_ubcalloc field (number of times the UBC had to allocate a page from the virtual memory free page list to satisfy memory demands). For example:

```
(dbx) p vm_perfsum
struct {
    vpf_pagefaults = 6732100
    vpf_kpagefaults = 119865
    vpf_cowfaults = 926159
    vpf_cowsteals = 192703
    vpf_zfod = 2720195
    vpf_kzfod = 119865
    vpf_pgiowrites = 1882
    vpf_pgwrites = 4747
    vpf_pgioreads = 1874108
    vpf_pgreads = 1412
    vpf_swapreclaims = 4
    vpf_taskswapouts = 0
    vpf_taskswapins = 0
```

```
    vpf_vplmsteal = 1411
    vpf_vplmstealwins = 1365
    vpf_vpseqdrain = 0
    vpf_ubchit = 3851
    vpf_ubcalloc = 103378
    vpf_ubcpushes = 0
    vpf_ubcpagepushes = 0
    vpf_ubcdirtywra = 0
    vpf_ubcreclaim = 0
    vpf_reactivate = 1973
    vpf_allocatedpages = 16177
    vpf_wiredpages = 2805
    vpf_ubcpages = 5494
    vpf_freepages = 3384
    vpf_swapspace = 29230
}
(dbx)
```

You can also monitor the UBC by examining the `ufs_getapage_stats` kernel data structure. You can calculate the hit rate by dividing the value for `read_hits` by the value for `read_looks`. A good hit rate is a rate above 95 percent.

```
(dbx) p ufs_getapage_stats
struct {
    read_looks = 2059022
    read_hits = 2022488
    read_miss = 36506
}
(dbx)
```

In addition, you can check the UBC by examining the `vm_tune` structure and the `vt_ubcseqpercent` and `vt_ubcseqstartpercent` fields. These values are used to prevent a large file from completely filling the UBC and using a lot of memory, thus limiting the amount of memory available to the virtual memory subsystem.

For example:

```
(dbx) p vm_tune
struct {
    vt_cowfaults = '^D'
    vt_mapentries = 200
    vt_maxvas = 1073741824
    vt_maxwire = 16777216
    vt_heappercent = '^G'
    vt_anonklshift = 17
    vt_anonklpages = 1
    vt_vpagemax = 16384
    vt_segmentation = '^A'
    vt_ubcpagesteal = 24
    vt_ubcdirtypercent = '\n'
    vt_ubcseqstartpercent = '2'
    vt_ubcseqpercent = '\n'
```

```
    vt_csubmapsize = 1048576
    vt_ubcbuffers = 256
    vt_syncswapbuffers = 128
    vt_clustermap = 1048576
    vt_clustersize = 65536
    vt_zone_size = 67108864
    vt_kentry_zone_size = 16777216
    vt_syswiredpercent = 'P'
    vt_asyncswapbuffers = 4
    vt_inswappedmin = 1
}
```

See Section 3.4.1 for information on how to tune the UBC.

2.2.8.5 Checking the Metadata Cache with dbx

The metadata buffer cache contains file metadata (superblocks, inodes, indirect blocks, directory blocks, and cylinder group summaries). To check the metadata buffer cache, use `dbx` and examine the `bio_stats` structure.

```
(dbx) p 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` configuration file parameter. 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 3.4.1.3 for information on how to tune the metadata cache.

2.2.9 Monitoring the Network – netstat Command

To check network statistics, use the `netstat` command (or `nfsstat` command, see Section 2.2.10). Some problems to look for are as follows:

- If `netstat -i` shows excessive amounts of input errors (`Ierrs`), output errors (`Oerrs`), or collisions (`Coll`), this could indicate a network problem (for example, cables not connected properly or Ethernet saturation).

- If the `netstat -m` command shows several requests for memory delayed or denied, this means that your system had temporarily run short of physical memory.

Most of the information provided by `netstat` is used to diagnose network hardware or software failures, not to analyze tuning opportunities. See the manual *Network Administration and Problem Solving* for additional information on how to diagnose failures.

The following example shows the output produced by the `-i` option of the `netstat` command:

```
# 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
```

The network is reasonably busy in this example, as shown by the collision output packet rate (4881/23632, or about 20 percent). This is a higher than normal value, and might indicate a wiring problem in the local area network. The four output errors are shown in the output produced by the following `netstat` command; all are due to excessive collisions.

```
# netstat -is

ln0 Ethernet counters at Fri Jan 14 16:57:36 1994

        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 –s option for the `netstat` command displays statistics for each protocol:

```
# 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 'cuz 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 options supported by the `netstat` command.

2.2.10 Displaying NFS Statistics – nfsstat Command

To check NFS statistics, use the `nfsstat` command. For example:

```
# nfsstat

Server rpc:
calls      badcalls   nullrecv   badlen     xdrcall
259        0          0          0          0

Server nfs:
calls      badcalls
259        0
null       getattr    setattr    root       lookup     readlink   read
2  0%      58 22%     0  0%      0  0%      59 22%     0  0%      89 34%
wrcache    write      create     remove     rename     link       symlink
0  0%      36 13%     1  0%      1  0%      0  0%      0  0%      0  0%
mkdir      rmdir      readdir    statfs
```

```
0  0%      0  0%      6  2%      7  2%

Client rpc:
calls      badcalls   retrans    badxid     timeout    wait       newcred    badverfs   timers
180889     315        0          0          315        0          0          0          6

Client nfs:
calls      badcalls   nclget     nclsleep
180574     0          180574     0
null       getattr    setattr    root       lookup     readlink   read
0  0%      82859 45%  0  0%      0  0%      47881 26%  882  0%    46821 25%
wrcache    write      create     remove     rename     link       symlink
0  0%      36  0%     1  0%      1  0%      0  0%      0  0%      0  0%
mkdir      rmdir      readdir    statfs
0  0%      0  0%      2027  1%   66  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 Section 3.6.3 for information on how to tune your system to avoid timeouts.

Tuning Subsystems and Applications 3

After you determine where your performance problem exists, you can then begin to remedy it.

This chapter describes various ways that you can tune your system and applications. Tuning your system can include changing system parameters, increasing resources such as CPU and memory, and changing the system configuration, such as adding disks, spreading out file systems, and adding swap space.

Note

Never attempt to tune your system until you have confirmed that the performance problem on the system is not caused by an application that is either broken or in need of further optimization.

Tuning an application involves modifying the build process, modifying the source code, or both.

As a general rule, performance tuning consists of performing several of the following tasks:

- Reviewing tuning guidelines (Section 3.1)
- Optimizing your applications (Section 3.2)
- Optimizing the use of your CPU (Section 3.3)
- Tuning memory (Section 3.4)
- Tuning interprocess communications (Section 3.5)
- Tuning I/O (Section 3.6)

3.1 Tuning Guidelines

Prior to tuning your system, you need to understand how your system is intended to be used. For example, is your system mostly file serving? Are users running many small applications or are they running mostly large applications? Without this understanding, your attempts at tuning may cause more harm than good. If you understand the system's intended use and you perceive a performance problem, keep the following tuning rules in mind:

- The DEC OSF/1 operating system in many ways tries to tune itself according to the work load. For example, it dynamically adjusts the unified buffer cache (UBC) according to file system I/O.
- All of the components of the DEC OSF/1 operating system interact in some manner. A change you make in one component may have an unexpected effect elsewhere. Try the tuning suggestions that cause the least disruption to the user community.
- Make the easy changes first (for example, add an additional swap space area).
- Make one change at a time. Try to avoid making too many changes at once. By making one change at time, you can track exactly what has helped or hurt the system and, if need be, you can ensure that you can return to a previous state.
- Know when to quit. Tuning has diminishing returns. Squeezing that last half percent performance improvement may not be worth the effort that goes into it.

The following is an example of a procedure you could follow if you are sure that your applications are not causing the performance problem:

1. Is there enough memory? Use the `vmstat` command to display information about virtual memory.

   Check the number of pages on the free list. If the number is less than 128, you may have a virtual memory problem. If so, you could do the following:

   - Ensure that no new application is adversely affecting your system environment.
   - Modify the system load if possible.
   - Check your swap space configuration. Spread out swap partitions across several disks. Configure your swap space at system startup to get the best performance benefit. Use the `swapon -s` command to display your swap space configuration. Use the `iostat` command to determine which disks are getting the most use. Do not swap to the system disk if possible.
   - Decrease the amount of memory available to the UBC.
   - Modify the virtual memory configuration parameters.
   - Add more memory.

2. Is there enough CPU? Use the `vmstat` command to determine how the applications are using the CPU:

   - Check the idle time (`id`). A high idle time may not indicate anything that relates to a problem or it could indicate the following:

- An I/O bottleneck problem
- An application bottleneck problem

– Check the user time (`us`). A high user time and a low idle time could indicate a lack of CPU.

– Check the system time (`sy`). A high system time could indicate nothing, or it could indicate that the system is thrashing; that is, virtual memory is low, and the system is trying to reclaim pages.

3. Is the system paging? Use the `vmstat` command to check for a high page out rate (`pout`). You may have a virtual memory problem if you are paging excessively.
4. Do you have excessive disk I/O? Use the `iostat` command to determine which disks are being used the most. Spread out your swap space across several disks. Spread out file systems across disks.
5. Do you have a lot of network retransmissions or dropped packets? You may have a problem in your network. Use the `netstat` command with the `–i` and `–s` options to produce statistics that will help you to analyze input and output problems. Use the `nfsstat –c` command to obtain information about NFS retransmissions.

The following sections describe the various tuning possibilities.

3.2 Optimizing Applications

In many instances, optimizing an application can result in major improvements in run-time performance. Two preconditions should be met, however, before you begin measuring the run-time performance of an application and analyzing how to improve the performance:

- Check the software on your system to ensure that you are using the latest versions of the compiler and the operating system to build your application. Newer versions of a compiler often perform more advanced optimizations and newer versions of the operating system often operate more efficiently.
- Test your application program to ensure that it runs without errors. Whether you are porting an application from a 32-bit system to DEC OSF/1 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 `lint` with the `–Q` option or compile your program with the `–migrate –check` option to isolate addressing problems that you will need to resolve.)

Once you have verified that these conditions have been met, you can then begin the tuning process.

Application tuning can be divided into two separate, but complementary, activities:

- Tuning your application's build process so that you use, for example, an optimal set of preprocessing and compilation optimizations.
- Analyzing your application's source code to ensure that is uses efficient algorithms and that it does not use programming language constructs that can degrade performance.

The following sections provide details on considerations that relate to these two aspects of the tuning process.

3.2.1 Application-Building Guidelines

Opportunities for improving an application's run-time performance exist in all phases of the build process. The following sections identify some of the major opportunities that exist in the areas of compiling, linking and loading, preprocessing and postprocessing, and library selection.

3.2.1.1 Compilation Considerations

Compile your application with the highest optimization level possible, that is, the level that produces the best performance and the correct results. In general, applications that conform to standards should tolerate the highest optimization levels, and applications that do not conform to standards may have to be built at lower optimization levels. For details, see `cc`(1), `f77`(1), the *Programmer's Guide*, or the DEC Fortran user manual for DEC OSF/1 systems.

If your application will tolerate it, compile all of the source files together in a single compilation. Compiling multiple source files increases the amount of code that the compiler can examine for possible optimizations. This can have the following effects:

- More procedure inlining
- More complete data flow analysis
- A reduction in the number of external references to be resolved during linking

To take advantage of these optimizations, use the `-O3` optimization level for the system C compiler, the `-plus_list_optimize` option for the DEC C compiler, or the `-O4` option (default) for the DEC Fortran compiler. (Note that some routines may not tolerate a high level of optimization and these routines will have to be compiled separately.)

3.2.1.2 Linking and Loading Considerations

If your application does not use many large libraries, consider linking it nonshared. This allows the linker to optimize calls into the library, thus decreasing your application's start-up time. Nonshared applications, however, can use more system resources than call-shared applications. If your application will be used by many users simultaneously, you may increase total system performance by linking call-shared. See the *Programmer's Guide* or the *ULTRIX to DEC OSF/1 Migration Guide* for details.

For applications that use shared libraries, ensure that those libraries can be quickstarted. Quickstarting is a DEC OSF/1 capability that can greatly reduce an application's load time. For many applications, load time is a significant percentage of the total time that it takes to start and run the application. If an object cannot be quickstarted, it still runs, but startup time is slower. See the *Programmer's Guide* for details.

3.2.1.3 Preprocessing and Postprocessing Considerations

Preprocessing and postprocessing (run-time) options include the following:

- Use the KAP (Kuck Associates Preprocessor) tool to gain extra optimizations. The preprocessor uses final source code as input and produces an optimized version of the source code as output. It is especially useful for applications with a large number of floating-point operations. KAP is available for DEC OSF/1 systems as a separately orderable layered product.

  For DEC Fortran, KAP is invoked with the `kapf` command (which invokes separate KAP processing) or `kf77` (which invokes combined KAP processing and DEC Fortran compilation). For C, KAP is invoked with the `kapc` command.

  For information on how to use KAP on a C program, see the *KAP for C for DEC OSF/1 AXP User Guide*. For information on how to use KAP on a DEC Fortran program, see the *KAP for DEC Fortran for DEC OSF/1 AXP User Guide*.

- Use the `cord` utility to improve the cache behavior of C applications. This utility uses data from an actual run of your application to improve your application's use of the instruction cache. To use `cord`, you must first create a feedback file with the `pixie` and `prof` tools. See `pixie`(1), `prof`(1), `cord`(1), and `runcord`(1) for details. The *Programmer's Guide* also describes how to use these tools.

- To improve compiler optimizations, try recompiling your C programs with a feedback file. The C compiler can make use of data from an actual run of the program to fine tune its optimizations. The feedback information is most useful when compiling at optimization level `–O3`.

To create a feedback file, use the `pixie` and `prof` tools. See `pixie`(1), `prof`(1), and `cc`(1) for details.

3.2.1.4 Library Routine Selection

Library routine options include the following:

- Use the Digital Extended Math Library (DXML) for applications with complex floating-point operations. By using DXML, such applications may run significantly faster on DEC OSF/1 systems, especially when used with KAP. DXML routines can be called explicitly from your program or, in certain cases, from KAP (that is, when KAP recognizes opportunities to use the DXML routines). For C or DEC Fortran, you access DXML by specifying `-ldxml` on the compilation command line.
- If your application does not require extended-precision accuracy, you can use math library routines that are faster but slightly less accurate. For DEC C, use the `-FASTMATH` option, and for DEC Fortran, use the `-math_library fast` option. This causes the compiler to use faster floating-point routines at the expense of three bits of floating-point accuracy. See `cc`(1) or `f77`(1) for details.
- If you are using C, consider compiling with the `-D_INTRINSICS` option. This causes the compiler to inline calls to certain standard C library routines.

3.2.2 Application Coding Guidelines

If you are willing to modify your application, use the profiler tools to determine where your application spends most of its time. Many applications spend most of their time in a few routines. Concentrate your efforts on improving the speed of those heavily used routines.

Digital provides several profiling tools that work for C, Fortran, and other languages. See `pixie`(1), and `prof`(1) for more details. The *Programmer's Guide* describes how to use these tools.

After you have identified the heavily used portions of your application, consider the algorithms used by that code. Is it possible to replace a slow algorithm with a more efficient one? Replacing a slow algorithm with a faster one often produces a larger performance gain than tweaking an existing algorithm.

When you are satisfied with the efficiency of your algorithms, consider making code changes to help the compiler optimize your application. (For information on how to assist the compiler in making optimizations, see the manual *High Performance Computing* by Kevin Dowd (O'Reilly & Associates, Inc., ISBN 1-56592-032-5).)

The following sections identify performance opportunities involving data types, cache usage and data alignment, language-specific issues, and a few miscellaneous issues.

3.2.2.1 Data Type Considerations

Data type considerations include the following:

- The smallest unit of efficient access on Alpha AXP systems is 32 bits. Accessing a 8- or 16-bit data type can result in a sequence of machine instructions to access the data. A 32-bit or 64-bit data item can be accessed with a single, efficient machine instruction.

  Avoid using integer and logical data types that are less than 32 bits. In C, consider replacing `char` and `short` declarations with `int` or `long` declarations. In DEC Fortran, declare integer or logical variables using 4- or 8-byte data types.

- Multiplication and division of integer quantities is slower than multiplication and division of floating-point quantities. If possible, consider replacing such integer operations with equivalent floating-point operations.

3.2.2.2 Cache Usage and Data Alignment Considerations

Cache usage patterns can have a critical impact on performance:

- If your application has a few heavily used data structures, attempt to allocate these data structures on cache line boundaries in secondary cache. Doing so can improve your application's cache usage.
- Look for potential data cache collisions between heavily used data structures. Such collisions occur when the distance between two data structures allocated in memory is equal to the size of the data cache. If your data structures are small, you can avoid this by allocating them contiguously in memory.

Data alignment can also affect performance. By default, the DEC Fortran and C compilers align each data item on its **natural boundary**; that is, they position each data item so that its starting address is an even multiple of the size of the data type used to declare it. Data not aligned on natural boundaries is called **misaligned data**. Misaligned data forces necessary adjustments by software at run time, which can slow performance.

Some statements can force misalignments to occur; for example, the following DEC Fortran declaration statements can force data to be misaligned: `COMMON`, `EQUIVALENCE`, `STRUCTURE`, and `RECORD`.

For C, misalignment can occur when you type cast a variable of one data type to a smaller data type; for example, type casting a `char` pointer (1-byte

alignment) to an `int` pointer (4-byte alignment) and then dereferencing the new pointer may cause unaligned access. Also in C, use of the `__unaligned` keyword or packed structures created by `-Zpn` or `#pragma pack` can cause unaligned access.

For DEC Fortran, you can use the `-align` *keyword* option to correct alignment problems within declarations, or you can make necessary modifications to the declarations themselves within the source code. For DEC C, you can use the `-migrate -align` option.

During compilation of DEC Fortran programs, warning messages are issued for cases in which misaligned data can be identified. (Warning messages are not issued during the compilation of C programs.)

During execution of any program, the kernel issues warning messages ("unaligned access") for most instances of misaligned data. The messages include the program counter (pc) value for the address of the instruction that caused the misalignment. You can use the machine code debugging capabilities of the `dbx` debugger to determine the source code locations associated with pc values.

Some misaligned accesses in C programs do not result in a warning message being issued. In these cases, the compiler generates code to avoid the message, but performance is still affected. The code generated by the compiler includes the Alpha AXP machine code instructions `ldx_u` and `stx_u`, which are not as efficient as the codes for handling properly aligned data.

To check for ''unreported'' instances of unaligned access in DEC C applications, you can generate a listing file that contains an assembly language representation of the machine code. Then, you can search the listing file for occurrences of the inefficient instructions. The listing file also provides the relative addresses of the `ldx_u` and `stx_u` instructions, which you can use to locate the source code lines from which they were generated.

For additional information on data alignment in Fortran programs, see the DEC Fortran user manual for DEC OSF/1 systems. See `cc`(1) or `f77`(1) for details on alignment options that you can specify on compilation command lines.

3.2.2.3 C-Specific Coding Considerations

Coding considerations specific to C applications include the following:

- If your application uses large amounts of data for a short period of time, consider allocating the data dynamically with `malloc` instead of declaring it statically. When you have finished using the memory, free it so it can be used for other data structures later in your program. Using this technique to reduce the total memory usage of your application can substantially increase the performance of applications running on systems

with small amounts of physical memory.

- Minimize type casting, especially type conversion from integer to floating point and from a small data type to a larger data type.
- To avoid cache misses, make sure that multidimensional arrays are traversed in natural storage order, that is, in row major order with the rightmost subscript varying fastest and striding by 1. Avoid column major order (which is done by Fortran).

3.2.2.4 Fortran-Specific Coding Considerations

Coding considerations specific to Fortran applications include the following:

- Use arrays efficiently:
  - To avoid cache misses, make sure that multidimensional arrays are traversed in natural ascending storage order, that is, in column major order with the leftmost subscript varying fastest and striding by 1. Avoid row major order (which is done by C).
  - Perform one or few array operations that access all of the array or major parts of an array instead of numerous operations on scattered array elements. The fastest array access occurs when contiguous access to the whole array occurs.
- For applications with numerous floating-point operations, consider using the `-assume noaccuracy_sensitive` option if a small difference in the result is acceptable.
- Avoid mixing integer and floating-point (REAL) data in the same computation; use all floating-point numbers. Expressing all numbers as floating-point values eliminates the need to convert data between fixed and floating-point formats, resulting in improved run-time performance. For example, in the assignment statement `R=A/2*I`, if R and A are both REAL variables, change the constant (2) to a floating-point value (2.).
- Avoid unnecessary I/O associated with the following coding practices:
  - Unnecessary formatting of data
  - Unnecessary transfers of intermediate results
  - Inefficient transfers of small amounts of data

See the DEC Fortran user manual for DEC OSF/1 systems for details.

3.2.2.5 Miscellaneous Programming Considerations

Miscellaneous programming considerations include the following:

- Operations on unsigned integer variables can be faster than on signed variables. If appropriate, consider using unsigned quantities in your

source code instead of signed quantities. Also, for C programs, consider using the `–unsigned` option to treat all `char` declarations as `unsigned char`.

- Avoid using operations that are not native to the Alpha AXP processor. The compiler must emulate these operations in software, so they can be slow. These operations include integer division, transcendental operations (for example, sine and cosine), and square root.

3.3 Optimizing CPU Utilization

When applications are operating correctly but are experiencing high idle time because of a lack of CPU cycles, your options for correcting the situation are limited. If the overload condition is expected to continue indefinitely, the best long-term solution is to add more memory or replace your system with a larger one.

In general, the following adjustments can be made to improve CPU processing on a temporary basis:

- Job scheduling – Spread out the jobs within the time available. This can be done in a variety of ways:
  - Prioritize the jobs so that important jobs get run first (`nice` command for jobs not yet started; `renice` command for jobs that are running).
  - Schedule jobs at distinct times (`at` and `cron` commands) or when the load level permits (`batch` command).
- Job sizing – Extremely large programs may run more efficiently if you increase the following program size limits: `dfldsiz`, `maxdsiz`, `dflssiz`, and `maxssiz`. See the manual *System Administration* for details on how you can adjust these limits. (Note that job scheduling can also be very important for large programs.)
- Reducing the size of the kernel – In certain situations, it may be necessary or helpful to reduce the size of the kernel to free up memory resources. Kernel size reductions can serve as a temporary solution until additional memory can be acquired. Such reductions can be made at installation time or by reconfiguring the kernel:
  - At installation time (during the kernel build phase), you can minimize the number of kernel options in effect for your system. See the *Installation Guide* for details.
  - On a running system, you can remove optional software support using the `setld –d` command. You can also delete support for any unused devices or device types by editing the configuration file (`/usr/sys/conf/`*system_name* file). See the manual *System Administration* for details on these two methods.

If your system is heavily loaded but does not have a shortage of memory, increasing the size of the system vnode table may improve performance. The vnode table limits the number of active files. You can increase its size by giving a higher value to the `maxusers` parameter or by modifying the `nvnode` parameter in the `param.c` file. See the manual *System Administration* for details.

3.4 Tuning Memory

The memory subsystem is one of the first places where a performance problem can occur. Performance can be degraded when the virtual memory subsystem cannot keep up with the demand for pages.

Memory tuning falls into the following two areas of concern:

- Tuning the UBC

  You can limit the size of the UBC that is used for file system buffer cache. This increases the amount of memory available to the virtual memory subsystem, but decreases I/O performance.

- Tuning your virtual memory subsystem

  You can tune several parameters to improve the performance the virtual memory subsystem. Another method of improving its performance is to configure additional swap space or spread out your disk I/O. Adding memory is always an option.

The DEC OSF/1 operating system contains several configuration file parameters that tune memory. Table 3-1 lists some of the parameters that can have a significant impact on virtual memory, including paging and swapping, and the UBC. Reboot the system if you change any system parameters.

Table 3-1: Tunable Memory Parameters

Parameter	Default	Description
`anonklpages`	1	Pages to fetch in a cluster
`bufcache`	3	Percentage of memory dedicated to the metadata buffer cache
`csubmapsize`	1024*1024	Size of kernel copy map
`dfldsiz`	134217728	Default data segment size limit
`dflssiz`	1048576	Default stack size limit
`heappercent`	7	Percent of kernel virtual address space to allocate for use by the `heap`

Table 3-1: (continued)

Parameter	Default	Description
kentry_zone_size	16777216	Amount of kernel virtual address space that is available to create kernel virtual address map entries
mapentries	200	Maximum number of virtual memory map entries
maxusers	32	Number of simultaneous users the system can support easily
maxuprc	64	Maximum number of processes one user can run simultaneously
maxdsiz	1073741824	Maximum data segment size limit
maxvas	1L<<30	Maximum virtual address space for user maps
maxwire	1L<<24	Maximum amount of memory that can be wired
msgmnb	16384	Maximum number of bytes on queue
msgmni	50	Number of message queue identifiers
msgtql	40	Number of system message headers
segmentation	1(on)	Enables shared page tables
swapbuffers	128	Maximum number of swap buffers that are available for swap I/O.
syswiredpercent	80	Maximum percentage of wired memory system-wide
ubcbuffers	256	Minimum number of buffers that the UBC can contain
ubcdirtypercent	10	Percent dirty push value
ubcmaxpercent	100	Percentage of memory that the UBC can consume before page stealing begins
ubcminpercent	10	Percentage of memory at which page stealing is prohibited
ubcpagesteal	24	Steal vnode clean list
ubcseqpercent	10	The size of a file as a percentage of the UBC.
ubcseqstartpercent	50	The size of the UBC as a percentage of total memory.
vpagemax	16384	Maximum vpage for user map, or the maximum number of individually protected pages

Table 3-1: (continued)

Parameter	Default	Description
zone_size	67108864	Amount of kernel virtual address space that is available for many of the system's dynamic data structures.

3.4.1 UBC Subsystem

In some cases, an I/O intensive process may degrade the performance of other processes by using a major portion of the buffer cache. If you need more memory for the virtual memory subsystem, you can reduce the amount of memory that is available to the UBC. Note that reducing the memory available to the UBC can adversely affect file system I/O because the file system buffer cache will not be able to hold as much data.

The buffer cache is flushed with the `update` command. Buffer cache statistics can be viewed by using `dbx` and checking the `vm_perfsums` structure. You can also monitor the UBC by using the `dbx -k` command and examining the `ufs_getapage_stats` kernel data structure.

3.4.1.1 Changing the Size of the UBC

The size of the UBC is determined by the following configuration file parameters:

- `ubcmaxpercent` – Defines the maximum amount of total memory that can be used for the UBC. The default is 100 percent of memory.
- `ubcminpercent` – Defines the minimum amount of total memory allocation for the UBC. The default is 10 percent of memory.

The default size of the UBC is 10 percent to 100 percent of all memory. This means that the UBC will use at least 10 percent of all memory and can use up to 100 percent of all memory. If you wanted to reduce the amount of memory that can be allocated to the UBC, you could set `ubcmaxpercent` to 50 percent of all memory. This ensures that the UBC will not adversely affect the virtual memory subsystem. Note that if an application generates a lot of random I/O, a large UBC will not increase its performance.

If the page out rate is high and you are not using the file system heavily, you could decrease the value of `ubcminpercent` to reduce the rate of paging. Use the `vmstat` command to determine if the system is paging excessively.

Periodically, using `dbx`, examine the `vpf_pgiowrites` and `vpf_ubcalloc` fields of the `vm_perfsum` kernel structure. The page out rate may shrink if page outs greatly exceed UBC allocations.

For I/O servers, you may want to raise `ubcminpercent` to ensure that the UBC has more memory available for the file system buffer cache. If you do this, large programs that run occasionally will not completely fill the buffer cache. To check that you did not raise `ubcminpercent` too high, use the `vmstat` command to examine the page out rate.

If you increase the minimum amount of memory for the UBC with the `ubcminpercent` parameter, you may want to increase the size of the system page tables by increasing the `maxusers` parameter in the system configuration file.

If you change the `ubcmaxpercent` and `ubcminpercent` parameters, do not make the values so close together that you cause the system to page excessively or degrade I/O performance.

3.4.1.2 Preventing Cache Thrashing

The DEC OSF/1 operating system uses some configuration file parameters to prevent a large file from completely filling the UBC and using a lot of memory, thus limiting the amount of memory available to the virtual memory subsystem. The system will reuse the pages in the UBC instead of taking pages from the free page list when both of the following conditions are met:

- The size of the UBC is greater than the value of the `ubcseqstartpercent` parameter (the default is 50 percent of total memory).
- A referenced file is larger than the value of the `ubcseqpercent` parameter (the default is 10 percent of current UBC size).

The `ubcseqstartpercent` and `ubcseqpercent` configuration file parameters are used to ensure that a large file does not take all of the pages on the free page list and cause the system to page excessively.

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.

To determine the values of the `ubcseqstartpercent` and `ubcseqpercent` parameters, examine the `vm_tune` structure using `dbx`.

If you have a lot of memory, you may want to lower the `ubcseqstartpercent` value to 30 percent. Do not specify a lower value unless you decrease the size of the UBC. You probably do not want to change the `ubcseqpercent` parameter.

3.4.1.3 Changing the Size of the Metadata Buffer Cache

Although all memory is shared between the virtual memory subsystem and the UBC, the file system code that deals with the UNIX file system (UFS) metadata (including directories, indirect blocks, and inodes) still uses the traditional BSD buffer cache.

The `bufcache` configuration file keyword defines the size of the kernel's metadata buffer cache. The value for `bufcache` is the percentage of the system's physical memory that is allocated for the metadata buffer cache. The default memory allocation for the metadata buffer cache is 3 percent of physical memory.

Use `dbx` to examine the `bio_stats` 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 high miss rate (low hit rate), you may want to raise the value of `bufcache`. Note that any additional memory that you allocate to the metadata buffer cache is taken away from the rest of the system. This means that the memory is not available to the UBC and the virtual memory subsystem, and system performance may decline.

You can decrease the value of `bufcache` on large memory systems if the hit rate is high and you want to make more memory available to the virtual memory subsystem.

3.4.2 Virtual Memory Subsystem

Excessive paging, which is sometimes called thrashing, decreases performance. This means that the natural working set size has exceeded available memory. Because virtual memory runs at a higher priority, it blocks out other processes and spends all system resources on servicing page faults for the currently running processes.

You can determine if a system has memory problems by examining the output of the `vmstat` command. The `pout` column lists the number of page outs. The `free` column lists the amount of pages on the free page list. Excessive page outs and less than 128 pages on the free page list may indicate that excessive paging and swapping is occurring.

Some general solutions for reducing excessive paging and swapping are as follows:

- Reduce memory demands on the system by running fewer applications simultaneously. Use the `at` or `batch` command to run applications at night.
- Reduce the application's use of memory by using dynamically allocated memory instead of statically allocated memory. Also, use dynamically allocated memory more effectively, if possible.

- Add more physical memory.
- Reduce the amount of memory available for the UBC. Note that this may adversely affect I/O performance. See Section 3.4.1.1 for more information.
- Optimize the use of your swap space. See Section 3.4.3 for more information.

3.4.3 Modifying Your Swap Space Configuration

To optimize the use of your swap space, spread out your swap space across multiple devices and use the fastest disks for swap devices. Use the `swapon -s` command to display your swap space configuration. Use the `iostat` command to determine which disks are being used the most.

To ensure the best performance, place each swap partition on its own disk (instead of placing multiple swap partitions on the same disk). The page reclamation code uses a form of disk striping (known as **swap space interleaving**) so that pages can be written to the multiple disks. In addition, configure all of your swap devices at boot time to optimize swap space. See the manual *System Administration* for details on how to perform these operations.

To increase performance, you can change your swap mode from immediate mode (the default) to deferred mode (over-commitment mode) by removing (or moving) the `/sbin/swapdefault` file. Deferred mode requires less swap space and causes the system to run faster than if you used immediate mode because less swap bookkeeping is required. However, because deferred mode does not reserve swap space in advance, the swap space may not be available when it is needed by a task and the process may be killed asynchronously.

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"
"process limit"
"swap space below 10 percent free"
```

3.5 Tuning Interprocess Communication

You may be able to improve performance by tuning the following message parameters in the kernel configuration file:

- `msgmnb` (maximum number of bytes on queue)

  The process will be unable to send a message to a queue if doing so would make the total number of bytes in that queue greater than the limit specified by `msgmnb`. If the limit is reached, the process sleeps, waiting for this condition to be false.

- `msgtql` (number of system message headers)

  The process will be unable to send a message if doing so would make the total number of message headers currently in the system greater than the limit specified by `msgtql`. If the limit is reached, the process sleeps, waiting for a message header to be freed.

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 increase the values of the `msgmnb` and `msgtql` parameters to diminish waiting.

You might also want to consider tuning a number of other IPC parameters. How you tune the following parameters depends on what you are trying to do in your application:

- Message parameters
  - `msgmax` (maximum message size)
  - `msgmni` (number of message queue identifiers)
- Semaphore parameters
  - `semmni` (number of semaphore identifiers)
  - `semmsl` (number of semaphores per id)
  - `semopm` (maximum number of operations per `semop` call)
  - `semume` (maximum number of undo entries per process)
  - `semvmx` (semaphore maximum value)
  - `semaem` (adjust on exit maximum value)
- Shared memory parameters
  - `shmmax` (maximum shared memory segment size)
  - `shmmin` (minimum shared memory segment size)
  - `shmmni` (number of shared memory identifiers)

– `shmseg` (maximum attached shared memory segments per process)

(Note: As a design consideration, consider whether you would be better off using threads instead of shared memory.)

3.6 Tuning I/O

I/O tuning falls into the following areas of concern:

- Tuning file systems (Section 3.6.1)

  You can improve disk I/O by changing file system fragment sizes and other file system layout parameters.

- Tuning the network (Section 3.6.2)

  You can improve network performance by reducing the number of network applications, redesigning the network, or adding more memory.

- Tuning NFS (Section 3.6.3)

  In addition to improving NFS performance by using the techniques you use to improve the performance of the other file systems, you can improve NFS performance by using Prestoserve and by modifying a number of parameters.

The operating system includes several parameters that can affect the I/O subsystem. As specified in Table 3-2, they are set in either the `param.c` file or the system configuration file or by using `dbx`. Reboot the system if you change any system parameters.

Table 3-2: Tunable I/O Subsystem Parameters

Parameter	Default	Description
Read parameters:		
`cluster_consec_incr`	1	The increment for determining the number of blocks that should be combined on the next read-ahead request after the first read-ahead request. (Set with `dbx`.)
`cluster_consec_init`	2	The number of blocks that should be combined for the first read-ahead request. (Set with `dbx`.)
`cluster_lastr_init`	–1	The number of contiguous reads that need to be detected before read-ahead is requested. The default value will start read-ahead on the very first contiguous read request. (Set with `dbx`.)
`cluster_max_read_ahead`	8	The maximum number of clusters that can be used in read-ahead operations. (Set with `dbx`.)

Table 3-2: (continued)

Parameter	Default	Description
cluster_read_all	1	This variable is either on (!= 0) or off (==0). By default (on), perform cluster read operations on non-read-ahead blocks and read-ahead blocks. If off, perform cluster read operations only on read-ahead blocks. (Set with dbx.)
Write parameters:		
cluster_maxcontig	8	The number of blocks that will combined into a single write request. The default tries to combine eight 8KB blocks into a 64KB cluster. This variable controls all mounted UFS file systems. (Set with dbx.)
cluster_write_one	1	This variable is either on (!=0) or off (==0). By default (on), when a cluster needs to be written (that is, 64KB of data has been dirtied), but non-logically contiguous blocks make up the cluster, just the contiguous data is written, leaving the remaining data. The remaining data may be combined into future cluster requests. If off, 64KB of data will be written regardless of the number of write requests required to do so. (Set with dbx.)
Other parameters that influence I/O:		
delay_wbuffers	0	This variable applies only to UFS. It is either on (!=0) or off (==0). By default (off), write-behind is turned on. If on, flushing full buffers to disk is delayed until a sync call is issued. (Set in the param.c file.)
maxusers	32	The number of simultaneous users that your system can support easily without straining system resources. (Set in the system configuration file.)
open_max_hard	4096	Hard limit for the number of file descriptors that a process may have open. (Set in the param.c file.)
open_max_soft	4096	Soft limit for the number of file descriptors that a process may have open. This value is the default for all processes, and it must be less than or equal to the value of open_max_hard. (Set in the param.c file.)

The parameters listed in Table 3-2 are discussed in more detail in the sections that follow.

3.6.1 Disk Subsystem

Disk throughput is usually the gating performance factor. CPU, memory, and network can perform I/O much faster than disk. Disks are only capable of 2080 transfers per second. Therefore, disk configuration and tuning is critical.

The size of the disk operation is also important. In doing I/O to a disk, most of the time is taken up with the seek followed by the rotational delay. This is called the access time. For small I/O requests, access time is more important than the transfer rate. For large I/O requests, the transfer rate is more critical than the access time. Access time is also important for workstation, time-share, and server environments.

Most performance problems manifest themselves in disk saturation. Before you try to tune UFS and AdvFS file systems and the Common Access Method (CAM) subsystem, try to alleviate the problem with the following measures:

- Use fast disks.
- If possible, use many small disks instead of a few large ones. Small-sized disks usually have a better seek time and less rotational delay.
- Reduce the I/O load on the hardware.
- Reduce or stop paging and swapping by tuning the UBC or virtual memory.
- Run fewer applications simultaneously.
- Compress files to regain disk space.
- Use quotas to limit users' disk space.
- Layout the file systems across multiple disks to spread the I/O load evenly. Group together similar files, projects, and groups. Use as few file systems per disk as possible.
- For disk efficiency, isolate performance critical files. If you have multiple I/O subsystems, spread out disks, giving each drive its own controller. Use VMEbus-based I/O subsystems if possible.
- Spread out swap partitions across multiple disks.
- Optimize some file systems for transfer rate and some for access time.
- The memory file system (MFS) can improve read/write performance, but it is a volatile cache. Data is lost on reboot.
- To make look-up operations faster, you can adjust the size of the `namei` cache, which maps pathnames to inodes. You can monitor the hit rate by using `dbx` and examining the `nchstats` structure. Adjust the cache with the `maxusers` parameter in the configuration file or with the `nchsize` parameter in the `param.c` file.

- A number of system parameters are based on the maximum number of users. By increasing the `maxusers` parameter, you increase several other parameters, such as the maximum number of active processes allocated for each user, the file table (which determines the maximum number of files, sockets, and pipes that can be open simultaneously), the number of vnodes, and the size of the `namei` cache. Increasing the `maxusers` parameter will reduce the incidence of the following error messages:

  ```
  "file table full"
  "out of vnodes"
  ```

  You can also increase the sizes of the file and vnode tables by modifying the `nfile` and `nvnode` parameters in the `param.c` file.

The following sections describe how to tune UFS and AdvFS file systems and CAM.

3.6.1.1 Tuning UFS File Systems

This section describes how to tune your UFS file systems. Use the `dumpfs` command to display file system information.

You can tune file systems as follows:

- Use disk shadowing

  Disk shadowing can improve read performance, but it slows down write performance.

- Use Prestoserve

  Prestoserve can dramatically improve synchronous write performance.

- Check for disk fragmentation

  You can determine whether a disk is fragmented by determining how effectively the system is clustering. You can do this by using `dbx` to examine the `ufs_clusterstats`, `ufs_clusterstats_read`, and `ufs_clusterstats_write` structures. UFS block clustering is usually reasonably efficient. If the numbers from the UFS clustering kernel structures show that clustering is not being particularly effective, the disk may be heavily fragmented.

  Currently, the operating system does not have an on-line disk defragmenter for UFS. However, you can perform a defragmentation procedure to take care of heavily fragmented disk as follows:

  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.

- Adjust the file system fragment size

  You can do this using the `newfs` command. The fragment size is 1KB by default. The UFS file system block size is fixed at 8KB. A block size/fragment size of 8KB/1KB is usually sufficient.

  You can use a larger fragment size if the file system is used for executable files. Note that a large fragment size can waste disk space.

  You can use small fragment size if the file system is used for small files or code development. A small fragment size uses disk space more efficiently than a large fragment size.

  If you want to increase disk speed and most of the files are greater than 2 blocks (16KB), make the file system fragment size equal to the block size (8KB/8KB). This means less overhead to the system, but it requires more space on the disk.

- Reduce the density of inodes

  If the file system has many large files, reduce the density of inodes by using the `newfs -i` command.

- Change the rotational delay

  Use the `tunefs` command or the `newfs` command and set the rotational delay to 0 (zero) to allocate blocks contiguously. A rotational delay of zero will allocate logically contiguous blocks, which will aid UFS block clustering.

  If your applications perform a large number of contiguous writes, you can insert a rotational delay between contiguous block writes by setting the rotational delay to 1.

- Increase the number of blocks that are combined for a read

  Use the `tunefs` command or the `newfs` command to change the value of `maxcontig`, which specifies the maximum number of contiguous blocks that will be laid out before forcing a rotational delay; that is, the number of blocks that can be combined into a cluster. The default is 8. This causes the file system to attempt I/O read requests in a size that is defined by the value of `maxcontig` multiplied by the block size (the default is 64KB).

- Change the value of `maxbpg`

  Use the `tunefs` or the `newfs` command to change the value of `maxbpg`, which is the maximum number of file blocks allocated per cylinder group. Usually, this value is set to about one quarter of the total blocks in a cylinder group. The `maxbpg` parameter is used to prevent a single file from using all the blocks in a single cylinder group, which could degrade access times for all files subsequently allocated in that

cylinder group. By limiting the number of file blocks, large files must perform long seeks more frequently than if they were allowed to allocate all the blocks in a cylinder group before seeking elsewhere.

If your file system contains only large files, you can set the `maxbpg` parameter higher than the default value. To get the performance benefit on an existing file system, you must lay out the files on the disk again.

- Delay write flushing

  When a disk block is scheduled to be written, it is sent asynchronously to disk. The default operating system behavior prevents the block from being read while the write is in progress. An application that reads a block immediately after it is written could improve its performance if the write was delayed so that the block could be read.

  You can use `dbx` to turn on the `delay_wbuffers` kernel parameter to delay flushing full buffers to disk until a `sync` call is issued. If disabled, full buffers are flushed asynchronously when full. If enabled, full buffers are not flushed until the next `sync`. Enabling `delay_wbuffers` is useful when many small files are created or when files are written and immediately reread. The benefit is caused by avoiding I/O and long latencies if buffers are often locked on writes.

  Note

  Note that enabling `delay_wbuffers` causes the buffer cache to be dirtier, and data could be lost if the system is shut down abnormally. In addition, the I/O pattern has more spikes and could negatively affect real workloads.

  You can turn on `delay_wbuffers` in the `param.c` file.

- Increase the number of read-ahead clusters

  The UFS read-ahead algorithm is exponential in nature, but starts gradually. The kernel parameter `cluster_max_read_ahead` defines the maximum number of read-ahead clusters that the kernel will schedule. The default for `cluster_max_read_ahead` is 8. You can make the open algorithm faster by setting `cluster_read_all` to 1 and `cluster_consec_init` to the value of `cluster_max_read_ahead`. These global parameters can be changed with `dbx`.

- Increase the number of blocks that are combined for a write

  The `cluster_maxcontig` parameter is the number of blocks that will be combined into a single write. This variable controls all UFS file systems. The default value for `cluster_maxcontig` is 8. The value can be changed with `dbx`.

- Reduce the number of open file descriptors

  The `open_max_soft` parameter specifies the soft limit maximum number of open file descriptors. You could reduce the value to save memory. The parameter is set in the `param.c` file.

3.6.1.2 Tuning the Advanced File System

The Advance File System (AdvFS) is a file system option available on the DEC OSF/1 operating system. It provides rapid crash recovery, high performance, and a flexible structure that enables you to manage your file system while the system is on line. Associated with AdvFS are the optional POLYCENTER utilities that further enhance the capabilities of AdvFS. In particular, the `defragment`, `stripe`, and `migrate` utilities provide on-line performance tuning. The POLYCENTER utilities are available as a separately licensed layered product.

Methods for improving AdvFS performance include the following:

- Set up disks

  Enhance AdvFS performance by dedicating an entire disk (usually partition C) to one file domain. This avoids I/O scheduling contention.

- Back up and restore data

  If you do not have the optional POLYCENTER utilities, you can defragment your disks by backing up and restoring the filesets. Use the following procedure:

  1. Back up the filesets to tape or another partition using the `vdump` command.
  2. Recreate the domain using the `mkfdmn` command.
  3. Recreate the filesets using the `mkfsets` command.
  4. Restore the files using the `vrestore` command.

- Use fileset quotas

  Fileset quotas apply to the fileset, not to individual users or groups. By establishing quotas you can limit the amount of disk storage and number of files consumed by a fileset. This is useful when a file domain contains several filesets. Without fileset quotas, all filesets have access to all disk space in a file domain, allowing one fileset to use all the disk space in a file domain.

- Use the `defragment` utility

  If you have the optional POLYCENTER utilities, you can defragment your file system frequently without reducing system availability.

  File fragmentation can reduce the read/write performance of the file

because it results in more I/O operations to access the file. The `defragment` utility reduces the amount of file fragmentation in a file domain by attempting to move files and parts of files together so that the number of file extents is reduced.

You do not need to dismount the filesets in a file domain or otherwise take the domain off-line in order to run the `defragment` utility. You can perform all normal I/O operations while the `defragment` utility is running.

- Migrate files

  You can use the `migrate` utility in conjunction with the `showfile` command to improve file performance by monitoring and altering the way large files are mapped on the disk. This method of defragmenting files is useful for defragmenting specific files. (Use the `defragment` utility to defragment all files in a domain.) Use the following procedure as a guideline for this method of improving file performance:

  1. Show the status of all files in your working directory by using the `showfile` command with the `*` wildcard character.
  2. Check the performance percentage of each file. A low percentage (under 80 percent. indicates that the file is fragmented on the disk.
  3. Show the extent map of a fragmented file by using the `showfile` command with the `-x` option and the file name. The extent map shows whether the entire file or a portion of the file is fragmented.
  4. Based on the information provided by the extent map, migrate some or all of the file to the same volume or another volume in the file domain. In a single-volume file domain, the volume must have enough free, contiguous space to migrate the file.

  If several files in the file system are fragmented, you can add a new volume to the file domain and remove the volume containing the fragmented files. This action prompts AdvFS to automatically migrate all the files to the new volume and defragment each file during the process.

- Use file striping

  A file striping utility is provided with the optional POLYCENTER utilities. File striping provides load balancing and a higher transfer rate. File striping is a way to increase contiguous read/write performance by allocating storage in segments across more than one disk or volume without preconfiguring the disks.

  Using the `stripe` utility, you can distribute segments of a file across specific disks (or volumes) within a file domain. The Advanced File System determines the number of pages per stripe segment; the segments alternate among the disks in a sequential pattern. For instance, the file system allocates the first segment of a three-disk striped file on the first

disk; the next segment on the second disk; and the next segment on the third disk. This completes one sequence, or stripe. The next stripe starts on the first disk, and so on.

3.6.1.3 Tuning CAM

The operating system uses the Common Access Method (CAM) as the operating system interface to the hardware. CAM maintains pools of buffers (another type of cache) that are used to perform I/O. Each buffer takes approximately 1KB of physical memory. These pools should be monitored and can be tuned, if necessary.

The following parameters can be checked with the `dbx` debugger and modified in the `/usr/sys/data/cam_data.c` file:

- `cam_ccb_pool size` – The initial size of the buffer pool free list at boot time. The default is 200.
- `cam_ccb_high_water` – The number of buffers in the pool free list at which buffers are released to the kernel from the free list. The default is 1000 and should not be changed.
- `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 some memory to shutdown 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.
- `xpt_qhead` – This data structure contains information regarding the current size of the buffer pool free list (`xpt_nfree`), the current number of processes waiting for buffer (`xpt_wait_cnt`), and the total number of times processes had to wait for free buffers (`xpt_times_wait`).
- `ccmn_bp_head` – This data structure provides statistics on the buffer structure pool. This pool is used for raw I/O to disk. Some spreadsheet and database applications with their own file system use the raw device instead of UFS. The information provided is the current size of the buffer structure pool (`num_bp`) and the wait count for buffers (`bp_wait_cnt`).

Any modifications to the `/usr/sys/data/cam_data.c` file require rebuilding the kernel. See the manual *System Administration* for information on building a new kernel.

3.6.2 Network Subsystems

All resources used by the network subsystems are allocated and adjusted dynamically, so tuning is not really an issue with the network itself. NFS is the heaviest user of the network, and NFS tuning can be critically important (see Section 3.6.3).

Network performance is affected only when the supply of resources is unable to keep up with the demand for resources. Two types of 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 the problem and fix it (which may involve tuning some other components of the system). In the case of an overloaded network (for example, when the kernel issues a "`can't get mbuf`" message on a regular basis), you must either redesign the network, reduce the number of network applications, or increase the physical memory (RAM). See the *Network Programmer's Guide* or the manual *Network Administration and Problem Solving* for information on how to resolve network problems.

3.6.3 Network File System

The Network File System (NFS) shares the unified buffer cache with the virtual memory subsystem and local file systems. Much of what is described in Section 3.6.1.1 also applies to NFS. For example, adding more disks on a server and spreading the I/O across spindles can greatly enhance NFS performance.

Most performance problems with NFS can be attributed to bottlenecks in the file system, network, or disk subsystems. For example, NFS performance is severely degraded by lost packets on the network. Packets can be lost as a result of a variety of network problems. Such problems include congestion in the server, corruption of packets during transmission (which can be caused by bad electrical connections, noisy environments, babbling Ethernet interfaces, and other problems), and routers that abandon forwarding attempts too readily.

Apart from adjustments to the file system, network, and disk subsystems, NFS performance can be directly enhanced in the following ways:

- Install Prestoserve on the server. Prestoserve greatly improves NFS write performance. An NFS server must write a client's write data to stable storage before responding to the client. With Prestoserve, these

synchronous write operations are stored in the nonvolatile cache area (NVRAM). Storing them is this way is much faster than writing them to disk. See the *Guide to Prestoserve* for details.

- Adjust the number of `nfsiod` and `nfsd` daemons on client and server systems. These daemons perform the following functions:
  - The `nfsiod` daemons are used on the client to service asynchronous I/O requests. NFS servers attempt to gather writes into complete UFS clusters before initiating I/O, and the number of `nfsiod` daemons (plus 1) is the number of writes that a client will have outstanding at any one time. For good performance, a client should have 7 or 15 `nfsiod` daemons. (Having exactly 7 or 15 `nfsiod` daemons produces the most efficient blocking of I/O requests.)
  - The `nfsd` daemons are used on the server to service NFS requests from client machines. For good performance on heavily used NFS servers, a network should be configured with either 16 or 32 `nfsd` daemons. (Having exactly 16 or 32 `nfsd` daemons produces the most efficient blocking of I/O requests.)

  To determine whether performance is being degraded by an insufficient number of `nfsiod` and `nsfd` daemons, issue the following command:

  ```
  % ps alxww | grep nfs
  ```

  This command displays the `nfsiod` and `nfsd` daemons that have been established to service client and server requests. If only one or two `nfsiod` or `nfsd` daemons are idle, increasing their numbers may improve NFS performance. See the `nfsiod`(8) and `nsfd`(8) reference pages for details.
- For read-only file systems and slow network links, performance may be improved by changing the cache timeout limits. Note that 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 the `mount`(8) reference page (`acregmin`, `acregmax`, `acdirmin`, `acdirmax`, `actimeo` parameters) for details.
- NFS does not perform well when it is used over slow network links, congested networks, or wide area networks. In particular, network timeouts 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 will be severely degraded. See Section 2.2.10 for sample `nfsstat` output containing timeout and call statistics. (You can also use the `netstat -s` command to verify the existence of this problem; a non-zero count for ''fragments dropped after timeout'' in the ''ip''

section of the `netstat` output is a reliable indicator that the problem exists. See Section 2.2.9 for sample `netstat` output.)

To reduce the number of timeouts, increase the amount of time between NFS request retries. See the `mount`(8) reference page (`timeo` parameter) for details.

Also, when evaluating NFS performance, be aware that NFS also does not perform well if any file locking mechanisms are in use on an NFS file because the file cannot be cached on the client.

Index

D

F

G

I

K

L

M

N

O

P

R

S

T

U

V

W

X

How to Order Additional Documentation

Technical Support

If you need help deciding which documentation best meets your needs, call 800-DIGITAL (800-344-4825) before placing your electronic, telephone, or direct mail order.

Electronic Orders

To place an order at the Electronic Store, dial 800-234-1998 using a 1200- or 2400-bps modem from anywhere in the USA, Canada, or Puerto Rico. If you need assistance using the Electronic Store, call 800-DIGITAL (800-344-4825).

Telephone and Direct Mail Orders

Your Location	Call	Contact
Continental USA, Alaska, or Hawaii	800-DIGITAL	Digital Equipment Corporation P.O. Box CS2008 Nashua, New Hampshire 03061
Puerto Rico	809-754-7575	Local Digital subsidiary
Canada	800-267-6215	Digital Equipment of Canada Attn: DECdirect Operations KAO2/2 P.O. Box 13000 100 Herzberg Road Kanata, Ontario, Canada K2K 2A6
International	————	Local Digital subsidiary or approved distributor
Internal[a]	————	SSB Order Processing – NQO/V19 *or* U. S. Software Supply Business Digital Equipment Corporation 10 Cotton Road Nashua, NH 03063-1260

[a] For internal orders, you must submit an Internal Software Order Form (EN-01740-07).

Reader's Comments

DEC OSF/1
System Tuning and
Performance Management
AA-Q0R3A-TE

Please use this postage-paid form to comment on this manual. If you require a written reply to a software problem and are eligible to receive one under Software Performance Report (SPR) service, submit your comments on an SPR form.

Thank you for your assistance.

Please rate this manual:	Excellent	Good	Fair	Poor
Accuracy (software works as manual says)	☐	☐	☐	☐
Completeness (enough information)	☐	☐	☐	☐
Clarity (easy to understand)	☐	☐	☐	☐
Organization (structure of subject matter)	☐	☐	☐	☐
Figures (useful)	☐	☐	☐	☐
Examples (useful)	☐	☐	☐	☐
Index (ability to find topic)	☐	☐	☐	☐
Page layout (easy to find information)	☐	☐	☐	☐

What would you like to see more/less of? _____

_____

What do you like best about this manual? _____

_____

What do you like least about this manual? _____

_____

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 _____ Dept. _____

Company _____ Date _____

Mailing Address _____

_____ Email _____ Phone _____

---- Do Not Tear - Fold Here and Tape ----

digital™

No Postage
Necessary
if Mailed in the
United States

BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO.33 MAYNARD MASS.

POSTAGE WILL BE PAID BY ADDRESSEE

DIGITAL EQUIPMENT CORPORATION
OPEN SOFTWARE PUBLICATIONS MANAGER
ZKO3-3/Y32
110 SPIT BROOK ROAD
NASHUA NH 03062-9987

--- Do Not Tear - Fold Here ----

Cut
Alon
Dott
Line

Reader's Comments

DEC OSF/1
System Tuning and
Performance Management
AA-Q0R3A-TE

Please use this postage-paid form to comment on this manual. If you require a written reply to a software problem and are eligible to receive one under Software Performance Report (SPR) service, submit your comments on an SPR form.

Thank you for your assistance.

Please rate this manual:	Excellent	Good	Fair	Poor
Accuracy (software works as manual says)	☐	☐	☐	☐
Completeness (enough information)	☐	☐	☐	☐
Clarity (easy to understand)	☐	☐	☐	☐
Organization (structure of subject matter)	☐	☐	☐	☐
Figures (useful)	☐	☐	☐	☐
Examples (useful)	☐	☐	☐	☐
Index (ability to find topic)	☐	☐	☐	☐
Page layout (easy to find information)	☐	☐	☐	☐

What would you like to see more/less of? _____

_____

What do you like best about this manual? _____

_____

What do you like least about this manual? _____

_____

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 _____ Dept. _____

Company _____ Date _____

Mailing Address _____

_____ Email _____ Phone _____

---- Do Not Tear - Fold Here and Tape ----------

digital™

No Postage
Necessary
if Mailed in the
United States

BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO.33 MAYNARD MASS.

POSTAGE WILL BE PAID BY ADDRESSEE

DIGITAL EQUIPMENT CORPORATION
OPEN SOFTWARE PUBLICATIONS MANAGER
ZKO3-3/Y32
110 SPIT BROOK ROAD
NASHUA NH 03062-9987

--- Do Not Tear - Fold Here ----------

Cut
Alon
Dott
Line