DataGeneral

Data General Corporation, Westboro, Massachusetts 01580

Customer Documentation

Analyzing DG/UX™ System Performance

093–701129–02

AViiON®

PRODUCT LINE

Analyzing DG/UX™ System Performance

093–701129–02

For the latest enhancements, cautions, documentation changes, and other information on this product, please see the Release Notice (085–series) and/or Update Notice (078–series) supplied with the software.

Notice

Analyzing DG/UX™ System Performance
093–701129–02

Revision History:	Effective with:
Original Release – March 1993	DG/UX System 5.4 Release 2.01
First Revision – January 1994	DG/UX System 5.4 Release 3.00
Second Revision – July 1994	DG/UX System 5.4 Release 3.10

A vertical bar in the margin of a page indicates substantive technical change from the previous revision. (The exception is Appendix A, which contains entirely new material.)

About this manual

This manual describes how to analyze your DG/UX system performance and fine-tune your system. It explains how the DG/UX system implements the major abstractions of a computer: a CPU, virtual memory, a file system, and I/O devices.

This manual is for people concerned with system performance, most often system administrators.

How This Manual Is Organized

This manual contains seven chapters, an appendix, and a glossary of terms:

Chapter 1	Introduces DG/UX system performance.
Chapter 2	Describes the CPU and processes.
Chapter 3	Discusses memory.
Chapter 4	Describes the DG/UX operating system's file system and disk I/O.
Chapter 5	Describes terminal I/O.
Chapter 6	Discusses networking.
Chapter 7	Contains some helpful, common sense tips for improving system performance.
Appendix A	Contains a tuning example.
Glossary	Defines some concepts and terms used in this manual.

Related Data General manuals

Within this manual, we refer to the following manuals:

Managing Mass Storage Devices and DG / UX File Systems (093–701136). Explains how to manage disk and tape drives. Also explains DG/UX file systems, virtual disks, mirrors, and caching.

Managing TCP / IP on the DG / UX™ System (093–701051). Explains how to prepare for the installation of Data General's TCP/IP (DG/UX) package on AViiON computer systems. Tells how to tailor the software for your site, and use sysadm to manage the package and troubleshoot system problems.

Reader, please note:

Data General manuals use certain symbols and styles of type to indicate different meanings. The Data General symbol and typeface conventions used in this manual are defined in the following list. You should familiarize yourself with these conventions before reading the manual.

This manual also presumes the following meanings for the terms "command line," "format line," and "syntax line." A command line is an example of a command string that you should type verbatim; it is preceded by a system prompt and is followed by a delimiter such as the curved arrow symbol for the New Line key. A format line shows how to structure a command; it shows the variables that must be supplied and the available options. A syntax line is a fragment of program code that shows how to use a particular routine; some syntax lines contain variables.

Convention	Meaning
boldface	In command lines and format lines: Indicates text (including punctuation) that you type verbatim from your keyboard. All DG/UX commands, pathnames, and names of files, directories, and manual pages also use this typeface.
`Typewriter`	Represents a system response on your screen. Syntax lines also use this font.
italic	In format lines: Represents variables for which you supply values; for example, the names of your directories and files, your username and password, and possible arguments to commands. In text: Indicates a term that is defined or in the manual's glossary.
[ ]	In format lines: These brackets surround an optional argument. Don't type the brackets; they only set off what is optional. The brackets are in regular type and should not be confused with the boldface brackets shown below.
[ ]	In format lines: Indicates literal brackets that you should type. These brackets are in boldface type and should not be confused with the regular type brackets shown above.

...	In format lines and syntax lines: Means you can repeat the preceding argument as many times as desired.
$ and %	In command lines and other examples: Represent the system command prompt symbols used for the Bourne and C shells, respectively. Note that your system might use different symbols for the command prompts.
↲	In command lines and other examples: Represents the New Line key, which is the name of the key used to generate a new line. (Note that on some keyboards this key might be called Enter or Return instead of New Line.) Throughout this manual, a space precedes the New Line symbol; this space is used only to improve readability—you can ignore it.
< >	In command lines and other examples: Angle brackets distinguish a command sequence or a keystroke (such as <**Ctrl-D**>, <**Esc**>, and <**3dw**>) from surrounding text. Note that these angle brackets are in regular type and that you do not type them; there are, however, boldface versions of these symbols (described below) that you do type.
<, >, >>	In text, command lines, and other examples: These boldface symbols are redirection operators, used for redirecting input and output. When they appear in boldface type, they are literal characters that you should type.

Contacting Data General

Data General wants to assist you in any way it can to help you use its products. Please feel free to contact the company as outlined below.

Manuals

If you require additional manuals, please use the enclosed TIPS order form (United States only) or contact your local Data General sales representative.

Telephone assistance

If you are unable to solve a problem using any manual you received with your system, free telephone assistance is available with your hardware warranty and with most Data General software service options. If you are within the United States or Canada, contact the Data General Customer Support Center (CSC) by calling 1–800–DG–HELPS. Lines are open from 8:00 a.m. to 5:00 p.m., your time, Monday through Friday. The center will put you in touch with a member of Data General's telephone assistance staff who can answer your questions.

For telephone assistance outside the United States or Canada, ask your Data General sales representative for the appropriate telephone number.

Joining our users group

Please consider joining the largest independent organization of Data General users, the North American Data General Users Group (NADGUG). In addition to making valuable contacts, members receive *FOCUS* monthly magazine, a conference discount, access to the Software Library and Electronic Bulletin Board, an annual Member Directory, Regional and Special Interest Groups, and much more. For more information about membership in the North American Data General Users Group, call 1–800–253–3902 or 1–508–443–3330.

End of Preface

Contents

Tables

Figures

1 Introduction to DG/UX System Performance

This manual is for people concerned with system performance, most often system administrators. First, you should understand what your system's normal performance is; get a feel for the average system load. Although this manual often gives general guidelines, you should determine what the particular threshold values are for your system. Then, use this manual to analyze your system performance, to see if you can improve performance. You want to maximize efficiency and find system bottlenecks; however, you probably will reach a point at which you achieve maximum system performance—at that point, further work on your part leads to diminishing returns.

Finally, since you'll be familiar with the system's normal performance, you'll no doubt detect when problems occur—use this manual to help find out where the problem lies.

This document explains how the DG/UX system implements the major abstractions of a computer: a CPU, virtual memory, a file system, and I/O devices. Although performance metrics for individual applications would be most useful, statistics for operating systems are the best alternative. This manual explains how you can use system operations performance data to understand application performance.

Chapters 2–6 examine the CPU, memory, file systems and disk I/O, terminal I/O, and networking, respectively. Consult Chapter 7 for common sense performance tips.

Note that you should always use the revision of this manual that is appropriate for your revision of the DG/UX operating system. This manual is appropriate for DG/UX System 5.4 Release 3.10.

This chapter discusses the following topics:

- Sources of performance data
- System resources
- Tuning system parameters

Sources of Performance Data

Most performance data comes in raw form from system calls shipped with the DG/UX operating system such as **dg_sys_info**(), **dg_paging_info**(), **dg_process_info**(), **dg_lwp_info()**, **dg_vm_process_info()**, **dg_cpu_info()**, **getrusage**(), and **ioctl**(). The include files found in **/usr/include/sys** describe the raw numbers these calls generate. For more information about a system call listed above, see its man page.

Performance analysis tools that are bundled with the DG/UX operating system are **sar**, **nsar**, **ps**, **nps**, **timex**, and **prof**.

You can review statistics on CPU performance, disk and terminal I/O, memory usage, process communication and execution, and other activity with the **sar** (system activity reporter) command.

The **nsar** command displays system activity statistics as well. In addition to the information displayed by **sar**, **nsar** displays data such as DG/UX virtual memory statistics, kernel memory allocation activity, page-out activity, and virtual disk I/O statistics.

The System –> System Activity menu of **sysadm** provides operations for starting and stopping system activity monitoring, deleting old data collection files, and reviewing reports. You can also use **nsar** and **sar** at the command line; read the manual pages for details.

The **ps** command prints information about active processes. In addition to the information displayed by **ps**, **nps** prints DG/UX process and LWP information (LWPs are defined in Chapter 2). The System –> Process menu of **sysadm** provides operations for listing and deleting processes as well as changing process priority. You can also use **nps** and **ps** at the command line; read the manual pages for more information.

You can measure the elapsed, user, and system time of a command, report process data, and report system activity with the **timex** command. The given command is executed; the elapsed time, user time and system time spent in execution are reported (in seconds). Optionally, you can choose to list or summarize process accounting data for the command and all its children (see the **acct**(1M) man page for accounting information). You can also receive a report of total system activity during the execution interval. See the **timex** manual page.

The **prof** command interprets a profile file produced by the **monitor**(3C) function. See the manual pages for **prof** and **monitor**.

You can also use optional, value-added tools such as UX/RPM (the DG/UX Real-time Performance Monitor), Mxdb (Data General's Multi–extensible Debugger), and AV SysScope™.

UX/RPM collects and displays system and disk performance information as well as process data. The data is presented through a series of screens, and the performance data can be logged to disk for later playback. UX/RPM can also produce data interchange format (DIF) files from logs for analysis by standard spreadsheet tools.

Another profiling command, **mxprof**, works with Elf executables that use shared objects, and does not require special compilation or linking. **mxprof** is available in the separate product Mxdb.

The **sscope** command is a component of AV SysScope™, which logs and displays performance information about the DG/UX operating system. From the monitor control window, you may display disk data for the current host, start monitoring additional hosts, log performance data to a file, and examine previously created logfiles.

sscope–ps is a component of AV SysScope™ that monitors process performance information. This monitor displays an updating window of performance statistics similar to **ps** statistics.

System Resources

This section describes your system resources and where to find information about them.

CPU

Chapter 2 explains how the DG/UX operating system allocates limited CPU resources to all processes. The chapter covers the concepts of threads, LWPs, LWP groups, the Medium Term Scheduler, context switching, and interrupt handling.

Memory

Chapter 3 explains how the DG/UX operating system uses limited physical memory and a swapping device so that each process has access to a large virtual address space. The chapter describes the concepts of page faults, purging, and swapping.

File Systems and Disk I/O

Chapter 4 explains how the DG/UX operating system supports its file system, file system metadata caching, and raw device I/O. The chapter tells how to organize data on a disk for most efficient access.

Terminal I/O

Chapter 5 explains how the DG/UX operating system supports terminal I/O.

Networking

Chapter 6 explains the most important and common factors affecting network performance.

Tuning System Parameters

Tunable system parameters set various table sizes and system thresholds to handle the expected load on your system. You'll find the default values of tunable parameters are adequate for most configurations and applications. If your application has special performance needs, you may have to experiment with different combinations to find an optimal set.

Chapters 2–6 list the appropriate tunable system parameters at the end of each chapter under the Configuration Variables section. To set tunable parameters described in the following chapters, edit the values in your system file (use the **sysadm** System–>Kernel–>Build menu option) when building a new kernel. General information about DG/UX kernel tuning parameters is available in **/usr/etc/master.d/***.

End of Chapter

2 CPU and Processes

The DG/UX system, like most general-purpose operating systems, provides the abstraction of a virtual CPU to processes running on it. Each process appears to have the processor and a virtual address space to itself.

This chapter describes how your system deals with processes. These topics are covered:

- CPU usage, including system calls, context switches, and signals
- LWPs and LWP groups
- Medium Term Scheduler
- Per-process statistics
- Per–LWP statistics
- System-wide process statistics
- Configuration variables (CPU/process, message, semaphore, and scheduler)

Here are some terms that will be used in the following sections:

A *process* consists of an address space with one or more threads executing within that address space and their required system resources.

A *thread* is a single flow of control within a process. Each thread has its own thread ID number, its own scheduling priority and class, and its own stack, which the thread uses to store local variables. Threads are equal siblings that share the resources of a process. This sharing reduces significantly the thread's overhead and simplifies inter-thread communication.

An *LWP* is a "lightweight process." Each thread has a corresponding LWP in kernel space. These highly optimized LWPs have very low memory overhead (128 bytes). In this manual, "LWP" describes a thread at the kernel level. Because LWPs execute (rather than a process), "LWP" denotes an active executing entity in the kernel.

An *LWP group* is a set of computationally related LWPs sharing the same global priority, global scheduling class, CPU accounting, and scheduling time slice.

A *Job processor (JP)* is one CPU in an AViiON computer system. AViiON computers are available currently with one, two, four, eight, and sixteen JPs.

The *Medium Term Scheduler (MTS)* is a kernel LWP that schedules LWP groups. LWP groups are scheduled globally against other LWP groups in the system based upon their global scheduling priority and class. Each LWP group has one or more LWPs that are scheduled locally within the LWP group, based upon their local scheduling priority and class.

The *dispatcher* runs LWP groups, then LWPs within LWP groups. Multiple processors may execute LWPs from the same LWP group at the same time. The hardware implementation (actual number of physical processors) becomes irrelevant to the higher levels of the kernel.

Figure 2–1 shows the general design.

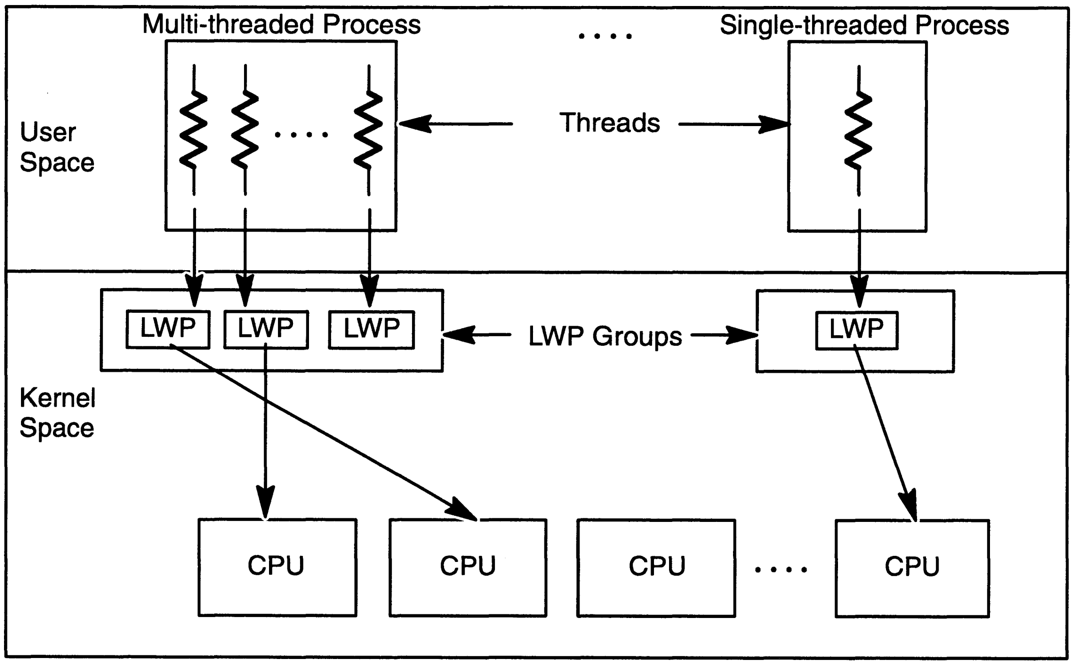


Figure 2–1 Processes, Threads, LWPs, and CPUs

A VP (virtual processor) is the dispatcher-level component of an LWP. The dispatcher maps every LWP to a VP, so multiplexing (where there were more active processes than available VPs) no longer exists.

The code for the kernel's dispatch scheduler runs on each JP in a multiprocessor system. A JP may reschedule globally (among LWP groups) or locally (within LWP groups) depending on the event that provoked the rescheduling. When an LWP within an LWP group suspends itself, the JP reschedules locally. Otherwise,when a JP is available, it reschedules globally by looking at the scheduling queues to determine which LWP group to run next.

A *scheduling queue* contains a list of eligible LWP groups. There may be multiple scheduling queues, depending upon the number of CPUs in your system and the complexity of the cache hierarchy. The kernel dispatch code looks at the queues and loads the highest priority LWP group.

Scheduling

The DG/UX system supports symmetric multiprocessing and uses a multi-level scheduler. The Medium Term Scheduler (MTS) determines the policy, while the actual scheduling is done by the dispatcher.

The MTS *binds* an LWP group by giving it one or more wired *transient data* sections. It is called transient data because it is only used by an LWP for one trip inside the kernel. An LWP group must be bound in order to be dispatched. If there are more LWP groups than available transient data, the MTS decides which subset of the LWP groups will be bound. The dispatcher then decides how to distribute transient data among LWPs in an LWP group. An LWP must have transient data in order to run on a processor.

Once an LWP enters the kernel for a normal system call or interrupt, it keeps its transient data until it leaves the kernel or exits. An LWP that suspends itself in user space is free to give up its transient data to other LWPs in its LWP group.

Check the rate of forks (Fork System Calls/Sec using UX/RPM or **sscope**, fork/s using **nsar –c**) versus the rate of binds (Process Binds/sec using UX/RPM, Binds/Second using **sscope**, swpin/s using **nsar –w**). For example, use this **nsar** command to query the system three times at 10-second intervals (see the **nsar** or **sar** man page for more information about command arguments):

```
% nsar -cw 10 3 ↵
...
11:40:12 scall/s sread/s swrit/s  fork/s  exec/s rchar/s wchar/s
         swpin/s bswin/s swpot/s bswot/s pswch/s
11:40:22      38       2       4    0.00    0.00     409     410
            0.10     4.4    0.00     0.0      43
11:40:32      55       3       6    0.00    0.00     428     422
            0.00     0.3    0.00     0.0      39
11:40:42      28       1       4    0.00    0.00     345     357
            0.00     0.0    0.00     0.0      36

Average       41       2       5    0.00    0.00     394     396
Average     0.03     1.6    0.00     0.0      39
```

If the rate of binds is greater than the rate of forks, probably more LWP groups need bound transient data sections than actually have bound transient data sections.

To see if your system is severely loaded, check the unbound runnable LWP groups (Unbound Runnable Processes using UX/RPM or **sscope**, swpq–sz using **nsar –q**). A value other than zero indicates one of two conditions:

- MAXBOUND (the configured number of bound transient data sections) is set too low. If it is consistently the case that more LWP groups need bound transient data sections, it is a good idea to increase the number of LWPs that can be ready to run in the kernel with the MAXBOUND parameter in your system configuration file. MAXBOUND specifies the maximum number of user LWPs that can be ready to run in the kernel, which is the same as the number of bound transient data sections. This is similar to NVPS in previous releases. You will need to rebuild and reboot your system for this change to take effect.
- The MTS has detected thrashing (LWP groups competing simultaneously for inadequate resources) and has temporarily reduced the available number of bound transient data sections in order to ease the system load. The system load due to the current number of bound runnable LWPs is too great to support adequate progress. When thrashing occurs, increasing MAXBOUND will not help. The common correction to reduce thrashing is to increase the amount of memory on the system.

The MTS is tuned to deal with fairness issues in a normal time-sharing environment. Therefore, its algorithms are slanted towards that environment. There are, however, other environments where fairness is not the main goal. In these areas the MTS may be hindering more than helping. There are several tuning parameters that can either adjust some of the MTS algorithms or completely turn them off. These tuning parameters all have default values; if you do not set the parameters, their default values will tune the MTS for a normal time-sharing environment.

If you feel that the MTS is obstructing (rather than helping) your particular environment, you can change configuration parameters to tune the MTS towards that environment. For example, if you want the MTS to unbind fewer transient data sections, you could increase MINBOUND (the minimum number of bound transient data sections). If you want the MTS to bind fewer transient data sections (perhaps you feel that the system is thrashing), you could decrease MAXBOUND.

To get approximate numbers for bound transient data sections, you can use **crash** to read these values:

mts_current_bound_transients: Current number of bound transients.

mts_min_allowed_bound_transients: Minimum; equal to MINBOUND if set, calculated by the MTS if not.

mts_max_allowed_bound_transients: Maximum; equal to MAXBOUND if set, calculated by the MTS if not.

mts_allowed_bound_transients: Current maximum number of allowed bound transients. The MTS will not allow any more bound transients than mts_allowed_bound_transients. This value will range between mts_min_allowed_bound_transients and mts_max_allowed_bound_transients.

If the MTS believes the system is thrashing, it will decrease mts_allowed_bound_transients down to a minumum of mts_min_allowed_bound_transients. If the MTS believes the system is not thrashing, it will increase mts_allowed_bound_transients up to a maximum of mts_max_allowed_bound_transients.

As long as mts_current_bound_transients is less than mts_allowed_bound_transients, the MTS has some available transient data sections. In other words, no processes that are ready to run are being deprived of transient data sections.

For example, after invoking **crash**, type this command:

```
> mr mts_current_bound_transients 1 d ↵
mts_current_bound_transients:      121
```

Medium Term Scheduler (MTS)

One role of the MTS is to maintain interactive response and non-interactive throughput. It accomplishes this by regulating the execution of non-interactive LWP groups to ensure that they do not simultaneously compete for inadequate resources (a condition known as "thrashing").

The MTS tries to detect over-subscription of some of the basic system resources such as memory and the CPU. Once the MTS has identified a resource problem, it reduces the number of available bound transient data sections to prevent thrashing and consequently maintain system throughput. By reducing the number of available bound transient data sections, the MTS effectively reduces the number of dispatchable LWP groups, thereby eliminating some of the system load.

As long as the MTS detects thrashing, it will gradually reduce the number of available bound transient data sections down to the configurable low-water mark, MINBOUND. When the MTS identifies that the system is once again stable, it will gradually increase the number of available bound transients up to configurable high-water mark, MAXBOUND.

Additionally, the MTS tries to be fair to timeshare LWP groups by adjusting their priorities so that

- Users are treated fairly. If LWP groups must be unbound to prevent thrashing, the MTS selects the groups to unbind. It first selects bound LWP groups that have no runnable LWPs, and then selects bound runnable LWP groups with the worst priority.

- Aging and languishing processes are boosted. If an LWP group is runnable but unbound for more than 8 seconds, it is said to be *aging*. When an LWP group is aged, it is given a temporary priority boost. If, after the boost, the LWP group has a better priority than a bound runnable LWP group, the MTS unbinds transient data from the bound runnable LWP group and binds it to the aged unbound runnable LWP group. Once the aged LWP group has made some progress, its aging priority boost is removed. You can turn off the aging check with the NOLANGUISHING configuration variable.

  If an LWP group is bound and runnable, but has not made any progress for 8 seconds, it is said to be *languishing* in the dispatcher. If a runnable timeshare LWP group has not been given any CPU cycles because it is languishing behind higher priority runnable LWP groups, the MTS gives it a temporary priority boost to allow it to make progress. This allows the LWP group to process any pending signals or get its state updated. Once the languishing LWP group has made some progress, its languishing priority boost is removed. You can turn off the languishing check with the NOLANGUISHING configuration variable; however, be aware that since CPU time is required for an LWP group to recognize and process its pending signals, a low priority LWP group may not die for a long time from either a kill command or a console interrupt.

- Interactive response is maintained. The MTS gives priority boosts to interactive LWP groups. A highly interactive LWP group tends to release its processor before its time slice has expired, whereas a CPU-intensive LWP group will use the entire slice. For example, an interactive LWP group might be doing terminal I/O; you want the response time to be minimal. On the other hand, CPU intensive processes are not very interactive and will not get much of a boost, if any. You can turn off the interactivity level adjustments with the NOILEVEL configuration variable.

  Some related statistics that you can view are the process priority (PRI using UX/RPM, **sscope–ps**, or **nps –l**) and the process utilization field (C using **nps –o putil**, **ps –l**, or i using UX/RPM). The process utilization field denotes the priority boost that the MTS gives an LWP group due to aging/languishing and interactivity; higher process utilization values mean higher priority boosts.

CPU Usage

The DG/UX system measures the CPU resources that a process receives by the amount of time its LWPs spend executing instructions (CPU time).

You can measure CPU time consumed by a process or by the system as a whole. To measure CPU time per-process, use the **nps** command. The data under TIME is the CPU time. (You can receive the same data from **sscope–ps** under TOTAL_TIME.) To get a list of the ten processes having the highest cumulative CPU execution times, use the command line

```
% nps –el | grep –v UID | sed –e "s,:,..,"  | sort –rn +9 –10 | head –10 ↵
```

UX/RPM provides a process screen that automatically shows the highest CPU usage per-process over a selected time period, as well as a breakdown of idle, system, and user times.

Note that using a lot of shell scripts on a system can result in forking many processes such as **tput**, **cat**, **stty, awk**, and **grep**. Even though these processes are short-lived (often a second or less), their overhead can be very expensive. If you think that these short-lived processes may be a system problem, turn on accounting (in **sysadm**, System –> Accounting –> Start) and analyze the accounting file; see the **acctcom** manual page. Reducing the size of the configuration variable MAXSLICE can improve the situation, since MAXSLICE specifies the maximum time in milliseconds a user process's LWPs can run before being suspended.

To measure CPU usage system-wide, use the **nsar –u** command. The output will look similar to this:

```
15:47:13    %usr    %sys   %idle
15:47:43      64      24      12
```

The data under %sys is the percent system time, the relative amount of time spent executing instructions in system (not user) code. (You can get the same data from all UX/RPM data screens or from **sscope** under Percent System Time.) A high percentage of system time indicates that applications are requesting CPU-intensive kernel services. Context switching between LWP groups and frequent process synchronization via IPCs are common examples of CPU-intensive kernel operations. Check processes' system time versus user time.

The data under %idle is the time spent by the system waiting for something to do. Ideally, you want the idle time to be rather low—for example, around 20%. However, if the system load is low, the idle time will be high, which is to be expected. If the idle time and the system load are both high, you probably have a memory problem. You could also have disk- or network-related I/O problems. If your system has zero or very low idle time, your system might have a runaway process or it might just be fully loaded.

An application that uses one single-LWP process cannot run any faster than a single processor can execute that process. A quad-processor system that is 75% idle may be executing a process as fast as it can be executed.

The data under %usr is the relative amount of time spent executing instructions in user code. A high percentage of user time indicates that one or more user applications are CPU-intensive. To find out which ones, look for processes with high CPU times per-process. By default, UX/RPM screens sort processes' CPU usage in descending order, highlighting the most active processes.

Short-lived processes can consume the CPU but exit before appearing in more than one **nps** sample. To check for this, look at the difference in the process ID numbers of consecutive **nps** commands or the difference in the fork rate with **nsar –c**. Because process ID numbers are assigned sequentially, the difference tells you the number of processes that have been created between **nps** invocations. See the **acct**(1M) man page for more information about per–process CPU usage.

CPU Time Interruptions

Interruptions of user CPU time can also affect performance. Sequential execution of instructions by an LWP in a user process may be interrupted in any of four ways:

1. System calls
2. Context switches
3. Interrupts
4. Signals

System Calls

Whenever an LWP calls for system services by making a system call, the process accumulates CPU time for the time spent executing instructions in the kernel address space on the process's behalf.

System calls vary widely in what they do and how long they take. However, even the simplest system call requires a fair amount of overhead. This overhead includes saving user registers, loading kernel registers, switching the clock to record system time as opposed to user time, performing the actual operation, switching the clock back, restoring registers, and returning.

In general, system calls such as **read**(), **write**(), **open**(), and **select**() that cause disk I/O, cause network I/O, or wait for a rendezvous are most expensive in terms of resources.

Context Switches

The basic unit of scheduling is the LWP group—each process has at least one LWP group (containing one or more LWPs). LWP groups in the system compete for global processor resources. The DG/UX operating system manages to run many LWPs at once by

- scheduling LWP groups on all the processors available,
- stopping an LWP group after it has run a certain time and letting another LWP group run, and
- running another LWP group that is ready to run whenever all of an LWP group's LWPs are waiting for input or some other event.

An LWP group may not perform well, however, if there are many other LWP groups competing with it for a processor. A lot of context switching (LWP groups being switched onto a physical processor to run) is a waste of time; unfortunately, this is often hard to control.

Look at the number of context switches (pswch/s using **nsar –w**, Process Switches using UX/RPM or **sscope**). When this rate gets into the thousands per second, the system is spending a lot of time switching between LWP groups rather than running LWPs. Hence, LWP groups are not getting much time in any given time slice. Note that this rate is very system dependent and some systems can support a higher rate with no decline in performance.

The measures of how many LWP groups are competing for the processors on a system are

- Number of bound/eligible/bound runnable LWP groups (Bound Processes, Eligible Processes, and Bound Runnable Processes using UX/RPM or **sscope**; runq–sz using **nsar –q**)
- Load average (described in the next section)
- Number of processes, which indicates the total but not how many are competing (Processes using UX/RPM or **sscope**; proc–sz using **nsar –v**; Pids using **sscope–ps**)

- Percent CPU time per–process—the amount of CPU time used divided by the product of the elapsed time and the number of processors (%CPU using UX/RPM or **sscope–ps**)

  If an LWP group has little competition for the CPU and is using the CPU exclusively, but performance is still below what you expect, examine the process to find inefficiencies in its implementation. Consider these things:

1. Can you revise an application to eliminate context switches?
   a. Can you buffer more data so that the system can do more efficient processing?
   b. Can fewer programs do the processing? This can help eliminate the need for such things as IPCs, pipes, reads, and writes.
   c. Can you eliminate or postpone flush or sync operations?
   d. Can you revise the programs to reduce the number of semaphore, message queue, or signal processing calls? Look at the rate of message operations and semaphore operations. Some systems perform well while doing a few thousand message and semaphore operations per second, while other systems have poor performance doing a few hundred per second. This is very application and system dependent.
2. Is the application a good threads candidate?
   a. Are you doing a large amount of I/O?
   b. Is there a large task that could be split up into multiple threads and run on separate CPUs?
   c. Are there large numbers of processes that could be condensed into a single multi-threaded process?
3. Should you consider a CPU upgrade?

Interrupts

Interrupts are caused by hardware events external to a processor. They include timer expiration and disk device events. Interrupts change the instruction stream suddenly to an interrupt handler that quickly records essential information pertaining to the hardware event. If no other LWP with a higher priority is waiting to execute, the LWP executing at the time of the interrupt then resumes execution. Because their execution speed is important, interrupt handlers do not switch the CPU timer from user to system time. As a consequence, the time spent in interrupt handlers is billed to whatever LWP group (and ultimately its process) that was running at the time of the interrupt.

Signals

Signals provide asynchronous interrupts to a process. They are not intended to be used for process synchronization and have not been optimized to provide the performance expected from common synchronization calls.

IMPORTANT: Note that on many AViiON computers, and UNIX systems in general, the clock ticks only 100 times per second. Thus, the time values returned by various system calls and commands are not exact. This leads to wide variability in reported times, for shorter processes especially.

System Load Average

The system load average is the average number of eligible LWP groups over a time period, normalized for processors. An LWP group is eligible if it has at least one LWP that is

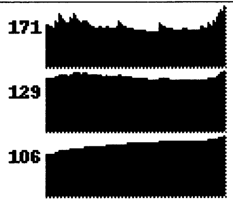
- not waiting for any external event such as keyboard input,
- not waiting of its own accord (including stopped for I/O), and
- scheduled.

In other words, the load average only counts LWP groups that would run if a CPU were available.

If your system has TCP/IP networking and each system on the local network is running the **rwhod** daemon, you can use the **ruptime** command to show the one-, five-, and fifteen-minute load averages for each host on the local network:

```
% ruptime ↵
abc          up 14+23:39,     1 user,   load 1.00, 1.08, 0.99
def          up 35+20:38,     1 user,   load 0.07, 0.15, 0.17
ghi        down 12+20:55
jkl          up 13+21:14,     2 users,  load 1.21, 0.53, 0.37
mno          up 21+02:51,     0 users,  load 0.00, 0.02, 0.16
server       up 19+23:54,     4 users,  load 0.02, 0.09, 0.12
```

You can also get the same information for a system from UX/RPM or **sscope** (use the Load format). Here's sample **sscope** output:



```
171    One Min Load Avg ( x100 )
129    Five Min Load Avg ( x100 )
106    Fifteen Min Load Avg ( x100 )
```

Check the system load average regularly to determine normal system loads for your system and other systems on your network. If your system performance seems to be poor, check the system load average. Note whether the load average is rising or falling—if it is falling, wait for a while to see if performance improves to a normal level for your system.

As stated earlier, the load average is normalized for the number of processors. A lower load average results if the number of LWP groups does not increase with the number of processors.

Run Queues

Another way to get a general idea of system load is to check run queues. You can use **nsar –q** to get this information; the example below queries the system six times at 10-second intervals:

```
% nsar -q 10 6 ↵
...
14:46:58 runq-sz %runocc swpq-sz %swpocc
14:47:08    13.0     100     0.0       0
14:47:18    13.0     100     0.0       0
14:47:28    14.0     100     0.0       0
14:47:38    16.0     100     0.0       0
14:47:48    15.0     100     0.0       0
14:47:58    16.0     100     0.0       0

Average     14.5     100     0.0       0
```

The average length of the run queue for a time interval is shown under **runq–sz**; longer run queues mean a heavier load. Again, determine what is normal for your system by monitoring this value. The **runq–sz** column shows how many LWP groups the scheduler thought were runnable, including LWP groups that were waiting for resources but have not exhausted their await interval. Most of the kernel LWPs (approximately 10 or 11) fit into this category.

The percentage of time that the run queue is occupied is shown under **%runocc**; ideally, this percentage should be high. If your system seems to be performing poorly, but the run queue is empty, look for memory or I/O problems.

UX/RPM and **sscope** report runq–sz as Bound Runnable Processes.

Per-Process Statistics

Descriptions of per-process statistics follow, with CPU, system, user, and elapsed time listed first—the rest are in alphabetical order. Per-process statistics are the sums of statistics from processes' LWP groups. Note that the application that you use or the command that you type is shown to the left of the colon; the field of interest to you is on the right of the colon and should not be typed in:

CPU time

Percent

UX/RPM: %CPU
sscope–ps: %CPU

Amount of CPU time used divided by the product of the elapsed time and the number of processors. Look at which processes are using the most CPU time. See if you can run those processes' LWPs during off-peak hours or adjust their priorities.

Time

UX/RPM: cpu
sscope–ps: TOTAL_TIME
nps: TIME
ps: TIME

Sum of user time and system time.

System time

UX/RPM: system
sscope–ps: SYS_TIME

Amount of time that the kernel was running on behalf of the process.

User time

UX/RPM: user
sscope–ps: USER_TIME

Amount of time the process's LWPs were running in user space.

Elapsed time

UX/RPM: start time
nps –o etime: ETIME

Amount of time elapsed since the process started.

Binds

UX/RPM: binds
nps –o nbind: NBIND

Number of times a process's LWP groups have been bound.

Command name

UX/RPM: command
sscope–ps: COMMAND
nps –o cmd: CMD
ps –l: CMD

Command name of the process.

Context Switches

Involuntary

UX/RPM: iv_switch
nps –o nswtch: NSWTCH

Number of involuntary context switches.

Involuntary (LWP)

nps –o lnswtch: LNSWTCH

Number of involuntary context switches associated with a globally scheduled LWP ("–" for locally scheduled LWPs).

Voluntary

UX/RPM: v_switch
nps –o vswtch: VSWTCH

Number of voluntary context switches.

Voluntary (LWP)

nps –o lvswtch: LVSWTCH

Number of voluntary context switches associated with a globally scheduled LWP ("–" for locally scheduled LWPs).

Emulated instructions

UX/RPM: emul
nps –o emul: EMUL

Emulated instruction count. Applications that were compiled with Gnu C prior to DG/UX System 5.4 or with compilers not fully compatible with the MC88110 platform generate instructions that are not directly supported by the MC88110 processor.

To execute these applications, DG/UX System 5.4 Release 2.10 and greater provide code that emulates the unsupported instruction tasks. The identification and emulation of the obsolete instructions decrease performance substantially. To get full MC88110 capability and performance, you should rebuild applications with the newest compiler revision that is available.

You can find out if you are running an older application with emulated instructions by executing the **nps –o pid,emul** command. The output provides the process ID and the number of instructions being emulated in the process; floating–point instructions are not included.

Floating point exceptions

UX/RPM: fpex
nps –o fpx: FPX

Floating point exception count.

IDs

Process ID

UX/RPM: pid
sscope–ps: PID
nps –f or nps –l: PID
ps –f or ps –l: PID

The process ID.

Parent process ID

UX/RPM: ppid
sscope–ps: PPID
nps –f or nps –l: PPID
ps –f or ps –l: PPID

The parent process ID.

Memory

Process size

UX/RPM: size
sscope–ps: SIZE
nps –l: SZ and RSS
ps –l: SZ

The number of 4096-byte pages that are resident in memory for a process. A process's resident memory requirements are a good indication of how much stress it places on the memory system.

In **nps**, the SZ field is the size of the process's mapped address space in kilobytes. The RSS field is the size of the resident portion of the process's address space in kilobytes. See Chapter 3 for more information about memory.

Use the following command line to get a list of the 10 processes using the most resident memory:

```
% nps –el | grep –v UID | sed –e "s,:,.," | sort –rn +5 –6 | \
head –10 ↵
```

The UX/RPM process screen can sort processes by resident memory size, in descending order.

Swap space

UX/RPM: swap
sscope–ps: SWAP
nps –o swap: SWAP

Amount, in pages, of anonymous memory (swap space) reserved for use by the process, whether actually used or not. This value is a process' contribution to the Reserved Anonymous Pages statistic discussed in Chapter 3.

Process state

UX/RPM: s
sscope–ps: S
nps –l: S
ps –l: S

State of the process's initial LWP group:

–	Non-existent
I	Intermediate
R	Runnable
S	Sleeping
T	Stopped
W	Waiting
Z	Terminated (Zombie)

Sleeping and waiting LWP groups are usually waiting for keyboard input. Waiting LWP groups are candidates for becoming unbound and bound. The process state becomes important to consider if the system begins binding and unbinding LWP groups that are otherwise runnable.

Zombie (also referred to as terminated or defunct) LWP groups are LWP groups that are waiting for their parent process to get a termination IPC (Interprocess Communication). They disappear either when their parent process calls **wait**(2) or when the parent process dies as you reboot your system.

Process utilization

UX/RPM: i
sscope–ps: C
nps –o putil: C
ps –f or ps –l: C

Process utilization represented by an integer from 0 to 7. This number is the sum of the initial LWP group's interactive level and its languishing level and aging level as determined by the MTS. There are 4 levels of interactivity and 4 levels of languishing/aging. Each successive process utilization level results in an additional priority boost being applied to the initial LWP group of the process. Therefore, the higher the process utilization, the more priority the initial thread group of the process has for being loaded onto a processor.

Scheduling

Class

UX/RPM: SC
nps –c: CLS
ps –c: CLS

Scheduling class; based on the process's initial LWP group. The value can be pFF (FIFO), pRR (round robin), dTS (time share), dLF (DG/UX LIFO), or dFF (DG/UX FIFO). However, you'll almost always see dTS. The FIFO scheduling class provides fixed-priority scheduling. A FIFO LWP group has an infinitely long time slice; once its LWPs are executing on CPUs, the LWP group stays there until the LWPs complete, are preempted by a higher priority LWP group, are blocked by actions that they take, or voluntarily give up the CPUs.

The round robin class supports fixed-priority, time-sharing LWP groups. With one exception, the rules for the round robin class are the same as the rules for FIFO class: the round robin class includes the concept of the time slice, which promotes fair scheduling by helping prevent LWP groups of the same priority from monopolizing a CPU.

The DG/UX time share class is very similar to the round robin class, except that round robin class LWP groups have a fixed priority—the MTS does not change their priorities as it can for time share LWP groups. This is the class you will see most often.

The DG/UX FIFO class is provided for developers or users who require absolute control over the way LWP groups are scheduled. LWP groups in this class can be assigned any priority. If you use this class, take care not to inadvertently set the priority of an LWP group higher than that of a critical kernel LWP.

The DG/UX LIFO class is identical to the DG/UX FIFO class, except that unblocked LWPs are placed before blocked LWPs when the process awakens.

Nice value	UX/RPM: nice sscope–ps: NI nps –o nice : NI ps –l: NI

Nice value used in priority computation; based on the process's initial LWP group. The nice values can range from 0 to 39. The higher the nice value, the lower the priority of a process. This reduces the demand that the process makes on the system. The default nice priority value is 20. This means that you can specify a maximum value of 19 or a minimum value of –20 (if you are superuser) when you issue a **nice** command. Only superusers can specify that a process's LWPs should execute at a higher priority.

Also, if you are superuser, you can use **renice** to force a process to a lower priority level. If a process's virtual memory requirements are very large, however, don't force it to a lower priority level; that will only make it linger in your system.

	Process priority	UX/RPM: pri sscope–ps: PRI nps –c: PRI ps –c: PRI

Priority of the process's initial LWP group; higher numbers mean better priority. This number changes dynamically. The system reduces the priority of LWP groups that require a lot of CPU time.

The priority of a process's initial LWP group is the sum of its base scheduling priority and its scheduling boost. The default base scheduling priority for the dTS scheduling class is 0x4FF (1279).

The scheduling boost is derived from this equation:

((MAXNICE – nice_value) * nice_scale) + (process_utilization * 14)

where MAXNICE = 39, nice_value = nice value, nice_scale = the value of mts_nice_scale (you can get this value from **crash**; it is often 11), process_utilization = process utilization, and 14 represents the "fair scheduling level gap."

So, for a process with a dTS initial LWP group, a nice value of 20, and a process utilization of 4, the priority would be

1279 + (((39 – 20) * 11) + (4 * 14)) = 1544

Signals		UX/RPM: sigs nps –o nsig: NSIG

Number of times a signal has been caught.

User	**ID**	sscope–ps: UID nps –l: UID ps –l: UID

User ID of the process.

	Name	UX/RPM: user sscope–ps: NAME nps –f: UID ps –f: UID

User name for the user ID of the process, if available from NIS or the password file. Otherwise, the user ID.

Per–LWP Statistics

Descriptions of per-LWP statistics follow. Note that this information applies directly to a process's corresponding threads (which have the same IDs as the LWPs):

CPU

ID — nps –WL: CPU

The CPU upon which the LWP is running; "–" when the LWP is not on a CPU.

Time — nps –L: LTIME

The total CPU time for globally scheduled LWPs.

LWP

Active LWPs — UX/RPM: lwps
nps –f: NLWP

The number of LWPs active in the process.

ID — nps –L: LWP

The ID of the LWP.

Joining — nps –WL: NJOIN

The number of LWPs waiting for the LWP to terminate; "–" if none.

Join Target ID — nps –WL: JNTARG

The LWP ID of the LWP that this LWP is waiting for to terminate; "'–" if none.

Mutex — nps –WL: MUTEX

The hexadecimal address of the mutex upon which the LWP is waiting or the mutex associated with a conditional variable (if waiting upon a condition); "–" otherwise.

Priority — nps –o lprior: LPRI

The scheduling priority of the LWP. The priority depends upon whether the LWP was locally or globally scheduled; globally scheduled LWPs have dedicated LWP groups.

Scheduling Class — nps –o lsclass: LCLS

Scheduling class of the LWP. The value can be pFF (FIFO), pRR (round robin), dTS (time share), dLF (DG/UX LIFO), or dFF (DG/UX FIFO). However, you'll almost always see dTS. "G" or "L" is appended for globally or locally scheduled LWPs, respectively.

State nps –WL: K U SPECL

The state of the LWP. The K column refers to kernel space, the U column refers to user space and the SPECL column displays special annotations about the LWP state. For most single-threaded processes and kernel LWPs, LWP states reflect kernel space only. For multi–threaded processes, LWPs tend to have both kernel and user states.

In kernel (K) space:

R	runnable
X	exiting
W	waiting

In user (U) space:

R	runnable
X	exiting
W	waiting
S	sleeping

The special (SPECL) annotations:

Y	yielded
I	software interrupted
T	stopped
D	detached
C	canceled

Variable nps –WL: CONDVAR

The hexadecimal address of the conditional variable upon which the LWP is waiting; "–" if the LWP is not waiting upon a condition.

System-Wide Process Statistics

Descriptions of system process statistics follow with idle, system, and user time first—the rest follow in alphabetical order:

Idle time

UX/RPM: Idle
sscope: Percent Idle Time
nsar –u: %idle
sar –u: %idle

Time spent by the system waiting for something to do. If no LWP can run because it is waiting for devices or timeouts, the DG/UX system runs the idle LWP. Idle LWPs may run on each processor; therefore an application with three single-LWP processes on a quad processor system may be completely CPU-bound but nonetheless show 25% idle time.

Ideally, you want the idle time to be rather low—for example, around 20%. However, if the system load is low, the idle time will be high, which is to be expected. If the idle time and the system load are both high, you probably have a memory problem. You could also have disk- or network-related I/O problems.

If your system has zero or very low idle time, your system might have a runaway process (one that uses progressively more system resources over a period of time while you are monitoring it). On the other hand, the system might just be fully loaded.

System time

UX/RPM: Sys
sscope: Percent System Time
nsar –u: %sys
sar –u: %sys

Relative amount of time spent executing instructions in kernel (not user) code. The time spent waiting for devices is not charged to system time.

A high percentage of system time indicates that applications are requesting CPU-intensive kernel services. Context switching caused by forks and frequent process synchronization via IPCs are common examples of CPU-intensive kernel operations. Check processes' system time versus user time.

A system that is spending a large percentage (perhaps 50%) of its time in the system state might be doing a lot of disk I/O, system call processing, scheduling, or handling a saturated resource such as memory. If the system time continues to be high, determine where the time is being spent. Consider using kernel profiling; see the **prfld**(1M) manual page for more information.

See System Calls.

User time

UX/RPM: User
sscope: Percent User Time
nsar –u: %usr
sar –u: %usr

Relative amount of time spent executing instructions in user code. Time spent waiting for devices or executing kernel services is not billed to user time. There are exceptions to this that are normally not significant. For example, user time is billed for the time it takes to handle interrupts that occur while an LWP is running. A high percentage of user time indicates that one or more user applications are CPU-intensive. To find out which ones, look for processes with high percentages of CPU time.

Binds

UX/RPM: Process Binds/sec
sscope: Binds/Sec
nsar –w: swpin/s
sar –w: swpin/s

Rate at which LWP groups are being bound, usually caused by **fork**(2) calls but also caused when the MTS must manage more LWP groups. Binding and unbinding incur overhead and, if they persist, result in lowered system efficiency. An LWP group that has been unbound will be bound only when it is runnable. See Fork System Calls, Bound LWP Groups, and Unbinds.

Bound LWP groups

UX/RPM: Bound Processes
sscope: Bound Processes

Number of LWP groups bound at the time of the sample. LWP groups must be bound before their LWPs can run. Therefore, the LWPs of bound LWP groups are able to run at lower cost than the LWPs of unbound LWP groups, which must first be bound. This number increases with forks and decreases with exits.

The number of bound LWP groups is limited by the number of bound transient data sections, determined by the static system configuration variable MAXBOUND. This is similar to NVPS in previous releases.

Bound runnable LWP groups

UX/RPM: Bound Runnable Processes
sscope: Bound Runnable Processes
nsar –q: runq–sz
sar –q: runq–sz

Number of bound LWP groups that have LWPs ready to run. See Unbound Runnable LWP Groups.

Context switches

UX/RPM: Process Switches/Sec
sscope: Process Switches/Sec
nsar –w: pswch/s
sar –w: pswch/s

Number of times LWP groups are switched onto a CPU to run; previously known as process switches. High numbers (thousands per second) indicate that LWP groups are not getting much time in any given time slice. This number will also go up with the number of CPUs. The number of system calls per second should be higher than this number (perhaps 3–4 times higher), because system calls often complete during one trip into the kernel without being switched out and back.

Eligible LWP groups

UX/RPM: Eligible Processes
sscope: Eligible Processes

Number of LWP groups having LWPs that are either running or that have been chosen to run when a CPU is available. Although its maximum value is limited by the MTS, this number is a measure of system load. See also Bound Runnable LWP Groups and Unbound Runnable LWP Groups.

Exec system calls

UX/RPM: Exec System Calls/Sec
sscope: Exec System Calls/Sec
nsar –c: exec/s
sar –c: exec/s

Number of **exec**(2) system calls. An **exec** call typically follows a fork. Because it typically starts a new executable program, **exec** creates demands on a system to read in the new program's pages.

Fork system calls		UX/RPM: Fork System Calls/Sec sscope: Fork System Calls/Sec nsar –c: fork/s sar –c: fork/s
	Number of **fork**(2) and **vfork**(2) system calls. Forks create new LWP groups and therefore create a demand for additional memory and, potentially, CPU resources.	
Load averages		UX/RPM and sscope: One Minute Load Average, Five Minute Load Average, Fifteen Minute Load Average
	The average number of eligible LWP groups over the last one, five, or fifteen minutes; this average is normalized by the number of CPUs.	
Message operations		UX/RPM: Message Operations sscope: Message Operations/Sec nsar –m: msg/s sar –m: msg/s
	Number of **msgsnd** calls; **msgget** and **msgrcv** calls are not counted in this number.	
Process table	**Overflow**	UX/RPM: Process Table Overflows sscope: Process Table Overflow nsar –v: ov sar –v: ov
	Number of attempts to create more than Process Table Size processes. A non-zero value may indicate an insufficient NPROC value; see Process Table Size.	
	Size	sscope: Process Table Size nsar –v: proc–sz (2nd number) sar –v: proc–sz (2nd number)
	Value of a static system configuration variable, NPROC, that is the upper bound on the number of user processes. Configuring a kernel with a high NPROC value is not without cost because some resources, particularly memory, are allocated for per process whether they are used or not. You can change this value in the system configuration file (follow the **sysadm** path System –> Kernel –> Build option).	
Processes		UX/RPM: Processes sscope: Processes nsar –v: proc–sz (1st number) sar –v: proc–sz (1st number)
	Number of user processes existing at the time of the sample.	

Processors

UX/RPM: CPU Count
sscope: Number of Processors

Number of active CPUs on the monitored system.

Semaphore operations

UX/RPM: Semaphore Operations
sscope: Semaphore Operations
nsar –m: sema/s
sar –m: sema/s

Number of semaphore operations performed by the **semop** system call. If the rate of semaphore operations is high (on the order of 500/CPU/sec) and your system seems underutilized, you may have a bottleneck among processes that are heavily contending for semaphores (and resources that are protected by the semaphores). The solution to this problem is to understand what resources are causing the contention and either reduce dependency on the problem resources, provide more of them, or divide the resource into smaller components that each have their own semaphore.

System calls

UX/RPM: System Calls/Sec
sscope: System Calls/Sec
nsar –c: scalls/s
sar –c: scalls/s

Number of system calls made by LWPs. The rate and type of system calls may determine the amount of CPU time being used by the DG/UX system. This value is very dependent on the application and can range from a few hundred to thirty or forty thousand. If the system is spending a lot of time executing instructions in user code and this value is high (say, greater than 1500 system calls per CPU), applications could be making excessive system calls.

See System Time, User Time, and Context Switches.

Unbinds

UX/RPM: Process Unbinds/sec
sscope: Unbinds/Sec
nsar –w: swpot/s
sar –w: swpot/s

Rate at which LWP groups are being unbound, caused by the MTS. Binding and unbinding incur overhead and, if they persist, result in lowered system efficiency. An LWP group that has been unbound will be bound again only when it is runnable.

Unbound LWP groups

UX/RPM: Unbound Processes
sscope: Unbound Processes

Number of LWP groups that are not bound; see Bound LWP Groups.

Unbound runnable LWP groups	UX/RPM: Unbound Runnable Processes sscope: Unbound Runnable Processes nsar –q: swpq–sz sar –q: swpq–sz

LWP groups having LWPs that could run, but are not bound. When this number is not zero, your system most likely has a performance problem. Either MAXBOUND may be set too low or the MTS has detected system thrashing and has temporarily restricted the number of bound transients.

Configuration Variables

CPU and Process Configuration Variables

The CPU and process configuration variables are also listed in the file **/usr/etc/master.d/dgux**.

NCPUS

Specifies the number of processors to run. If set to **0** (the default), all available CPUs will be used. Any other value specifies that number of CPUs to run. If the value specified is more or less than the number of CPUs present, a message to that effect is printed when the kernel is booted. Notice that on a uniprocessor system, this parameter has no real effect since the one processor will always be run.

NPROC

Specifies the maximum number of user processes the system can have at one time. For various sized systems use the following values: small (such as workstations), **96**; medium (the default, such as AViiON 4xxx and 5xxx computers), **256**; and large (such as AViiON 62xx and 8xxx computers), **2048**. The overall number of processes needed depends on the number of terminal lines available, the number of processes spawned by each user, and the number of system processes and network daemons. If the maximum number of processes is used up, the **fork**(2) or **vfork**(2) system call will result in a process table overflow and will fail.

NLWP

Specifies the maximum number of user LWPs the system can have at any one time. If set to **0** (the default), this value is dynamically calculated based on the amount of available memory in the system.

NLWPGROUPS

Specifies the maximum number of user LWP groups the system can have at any one time. If set to **0** (the default), this value is dynamically calculated based on the amount of available memory in the system. An LWP group is a set of locally scheduled LWPs from the same process that share the same accounting and global scheduling parameters.

MAXUP

Specifies the maximum number of processes that a user (other than root) can have in existence at one time. The default is **50**. This value should not exceed the value of NPROC (NPROC should be at least 10% more than MAXUP). This value is per user identification number, not per terminal. For example, if ten people logged in with the same user ID, the default limit would be reached very quickly.

MAXULWP

Specifies the maximum number of LWPs that a user (other than root) can have in existence at one time. By default, there is no per-user limit.

MAXULWPGROUPS

Specifies the maximum number of LWP groups that a user (other than root) can have in existence at one time. By default, there is no per-user limit.

SDESLIM

Specifies the default (soft) number of file descriptors a process is allowed to have at one time. A non-superuser process may change its soft limit up to the value of the hard limit. The default is **64**. It is a good idea to keep the default value; you can code any applications that require additional file descriptors to use the system call **setrlimit**(), which sets resource limits.

HDESLIM

Specifies the maximum (hard) number of file descriptors a non-superuser process is ever allowed to have at one time. The default is **1024**.

MAXGLOBALSQS

Specifies the maximum number of global synchronization queues that can be used for user process-shared mutexes and condition variables. The default is **32768**.

INITCPUMASK

Specifies the set of CPUs on which the kernel demons and the initial user processes may run. You can use this to reserve some CPUs in the system for later exclusive use by a dedicated application. If this mask is **0** (default), the kernel demons and user processes are allowed to run on all CPUs in the system. In a specified mask, the low-order bit (0x1) specifies CPU0.

STRDEMONSCPUMASK

Specifies the set of CPUs on which the STREAMS demons may run. You can use this to keep STREAMS demons away from certain CPUs in the system. If the CPU mask does not specify any CPU (**0**, the default), then the STREAMS demons are allowed to run on any CPU specified by INITCPUMASK above (default is all). In a specified mask, the low-order bit (0x1) specifies CPU0.

Scheduler Configuration Variables

The following scheduler configuration variables are also listed in **/usr/etc/master.d/dgux**:

MAXSLICE

Specifies the dispatcher round-robin time slice used for the SCHED_OTHER (timesharing) and SCHED_RR scheduling policies. If a user LWP or LWP group runs for this amount of time, it will yield the CPU to other LWPs or LWP groups with the same priority. Note that the round–robin time slice is essentially infinite for SCHED_FIFO and SCHED_DG_LIFO scheduling policies. The default is **100** (1/10 second).

MAXAFFINITYSLICE

Specifies the time slice (in "real" time) that an LWP group may stay transiently joined to a given JP set before its affinity relationship is reset. This affinity time slice is used to implement a load–balancing algorithm for timesharing LWP groups—the smaller the value, the smoother the load balancing, at the expense of better throughput, and vice–versa.

This time slice does not affect realtime LWP groups because they are never allowed (by default) to migrate below the root of the JP set hierarchy—thus, they have no specific affinity. Also, manual affinity assignments are not broken by this time slice. This time slice only breaks migrations within the manual or default constraints imposed on the LWP group.

In summary, the LWP group is brought up to the highest level in the JP set hierarchy that makes sense for that LWP group. The default value for this parameter is **10** realtime seconds.

MAXLATENCY

Specifies the maximum time the current VP will run before being interrupted to check for preemption due to another JP adding a VP to an eligible list. The default is **50** milliseconds.

MINBOUND

Specifies the minimum number of user LWPs that can be ready to run in the kernel, which is the same as the minimum number of bound transient data sections (i.e., wired kernel stacks). If this parameter is set to **0** (the default), the minimum number of bound transients is determined dynamically based on system load. The minimum bound can never be larger than the maximum bound. If MAXBOUND is not 0 and MINBOUND is not 0, the minimum bound is the smaller of MINBOUND and MAXBOUND.

MAXBOUND

Specifies the maximum number of user LWPs that can be ready to run in the kernel, which is the same as the maximum number of bound transient data sections (i.e., wired kernel stacks). By default, this parameter is set to the size of physical memory (in MB) * 16. This configuration parameter corresponds most closely to NVPS in previous releases. The maximum bound can never be less than the minimum bound. If MINBOUND is not 0 and MAXBOUND is not 0, the maximum bound is the larger of MINBOUND and MAXBOUND.

NOLANGUISHING

Specifies to allow or disallow languishing for timeshare LWP groups. The default (**1**) is to detect, and try to correct, languishing for a timeshare LWP group by giving it a temporary priority boost until it makes progress. If the value of this variable is **0**, the scheduler does not do anything about runnable timeshare LWP groups that are not making progress.

NOILEVEL

Specifies to allow or disallow interactive level computations for timeshare LWP groups. The default (**1**) is to automatically adjust the interactive level of timeshare LWP groups. If the value of this variable is **0**, interactive level adjustments for timeshare LWP groups will not be made.

Without interactive level adjustments, the DG/UX operating system does not adjust priority based on LWP groups' CPU utilization. Instead, LWP groups are scheduled in a round-robin nature at the same priority level.

USEFILEPURGES
Specifies whether data file purges should be used in addition to program frame purges to detect thrashing. The default (**0**) is to only use program frames to detect thrashing.

LIFO
Specifies whether to set the dispatcher level scheduling policy for timeshare LWP groups as FIFO or LIFO. The default (**0**) for timeshare LWP groups is to use FIFO. If the value of this variable is **1**, timeshare LWP groups are scheduled LIFO at the same priority within the dispatcher.

Message Configuration Variables

These variables are also listed in **/usr/etc/master.d/dgux**. They are dynamic variables that set the message parameters shown in the following list.

MSGMNI
Specifies the maximum number of message queues that may exist in the system at one time. The default is **1024**.

MSGTQL
Specifies the maximum number of outstanding messages that may exist in the system at one time. The default is **1024**.

MSGMNB
Specifies the maximum number of bytes that a message queue may contain. The default is **4096**.

MSGMAX
Specifies the maximum number of bytes that a message may contain. The default **2048**.

Semaphore Configuration Variables

The following semaphore configuration variables are also listed in **/usr/etc/master.d/dgux**; they are dynamic variables:

SEMMNI
Specifies the maximum number of unique semaphore sets that may be active at any one time on the system. The default is **1024**.

SEMMSL
Specifies the maximum number of semaphores that a semaphore set may contain. The default is **256**.

SEMOPM
Specifies the maximum number of semaphore operations that can be executed per **semop**(2) system call. The default is **10**.

SEMVMX
Specifies the maximum value a semaphore may have. The default is the maximum value for this parameter, **32767**.

SEMUME
Specifies the maximum number of undo entries per undo structure. The default is **10**.

SEMAEM
Specifies the maximum value of the adjustment for adjust-on-exit. The value is used whenever a semaphore value becomes greater than or equal to the absolute value of **semop**(2), unless the program has set its own value. The default value is the maximum value for this parameter, **16384**.

SEMAPM
The maximum number of processes that may specify semaphore operation with SEM_UNDO option. The default is **16384**.

End of Chapter

3 Memory

After a general discussion of memory, this chapter lists memory statistics, followed by shared memory configuration variables.

Effective use of memory is critical because the difference between referencing data cached in memory and referencing data on a disk is the difference between perhaps 100 nanoseconds and 50 milliseconds—a hundred-thousand-fold difference! A DG/UX system uses its physical memory to implement a large virtual address space and to cache the computer's file system. The system tries to keep the physical memory filled with processes' address space and file system objects; doing so avoids having to read from a disk. In this discussion, all forms of secondary storage are referred to as "disks" at the risk of some inaccuracy. For instance, file system and swap devices may include remote NFS–mounted file systems, swap areas on a diskless client's server, or even devices that are implemented by NVRAM (non-volatile random access memory).

This chapter describes the many facets of virtual memory and file system buffering that influence a DG/UX system's performance. Some applications do not use file system buffering, so the memory statistics reported by the DG/UX system do not apply to them. Database management systems, for example, may have no choice about whether their data may be cached before writing to a disk; an update transaction may require that data reach a disk before the transaction is considered complete. These database management systems typically bypass file system buffering by using raw disk I/O, described in Chapter 4. Also, memory used for file system metadata (described in Chapter 4) and metadata operations is not reported by the memory statistics.

Typical DG/UX Memory System Behavior

A program begins execution when the DG/UX system overlays the address space of a process with the segments of a program file (called *mapping*). DG/UX memory statistics are reported in terms of *pages*. A page is the smallest unit of memory that supports access control, mapping to secondary storage devices, and modified/unmodified and age attributes; a page is 4096 bytes.

As a program begins execution, it references instructions or data on the pages of its address space. These pages may already be in memory—this occurs when the program is being executed or has recently been executed by another process. If the pages are already in memory, those pages (referred to as "resident pages") do not need to be read from disk; only the page table entry, which maps the program file pages into the process's address, needs to be set up. This operation, called a *soft page fault*, is much less expensive than referencing a page that is not in memory and which must then be read from a disk (called a *hard page fault*).

Pages that are executable and mapped to a file in the file system are called *program file pages*. The DG/UX system reports the number of program file pages that are resident and the numbers of various memory operations performed on these pages. Program file pages are read-only; this is important because these pages stay "clean". A clean page may be replaced in memory without first writing a copy of the page to disk.

Program files in the Executable and Linking Format (ELF) may specify that several portions of a process's address space should come from other files called *shared libraries* . With shared libraries, program file pages may be shared among many executing programs. You can load shared libraries into memory once and then many programs can reference the libraries when using common routines. Shared libraries may dramatically reduce the size of a program file in comparison to static libraries, and correspondingly reduce the pages necessary for many programs to run simultaneously. See **ld**(1) for more information about shared libraries and **cc**(1) for information about ELF.

Memory resident pages from files that are only read from or written to (in other words, files that are not executed) are called *data file pages*. A process can access such pages via the **read**() or **write**() system calls or by using the **mmap**() system call to map pages from those files directly into the process's address space. Data file pages are typically shared amongst all processes that are accessing the file in question, and changes made to the file by any of the processes are immediately seen by all the processes.

A process can map a file so that it can share the pages from the file with other users of the file until the process modifies the page. When it modifies the page, it takes a *copy-on-write (COW)* fault, at which time the process receives a private copy of the page in question, which it may then freely modify. Such a feature is commonly used with file pages to properly handle the initialized data section of a program (the .data section of an executable). As long as the initialized data is unchanged, it may be shared by several processes; once a process modifies its initialized data, it creates a private copy of the page. This is also how the MAP_PRIVATE option of **mmap**() is implemented.

A program's uninitialized data (which includes the ".bss" section of an executable) is not stored in the executable file at all, and therefore does not occupy file pages. The pages available for uninitialized data, along with an initial allotment of pages for a stack, are known as as *private anonymous pages*; these pages are allocated in a system's swap area. When allocating these pages, the system does not actually access a disk. The disk is accessed only when a page has been modified (become *dirty*) and the dirty page must be purged—that is, removed to make room for another page.

Private anonymous pages from the swap area are counted as *reserved anonymous pages*. Any time that you make a request for space (for example, by calling **malloc**() or extending a stack), you decrease the *freeswap* count (the amount of available swap space). The DG/UX system uses the number of reserved anonymous pages to guard against running out of swap space. Swap space consists of the swap area on disk and a portion of physical memory; this is also known as *total anonymous pages*. If you try to allocate a number of anonymous pages that would make the reserved anonymous pages exceed the total anonymous pages, the system reports the error message "out of swap space." Thus, an allocation request may fail even if there is unused swap space if that swap space is reserved by an earlier allocation request. This prevents the situation where an application finds out later that there is not enough swap space for its existing requests.

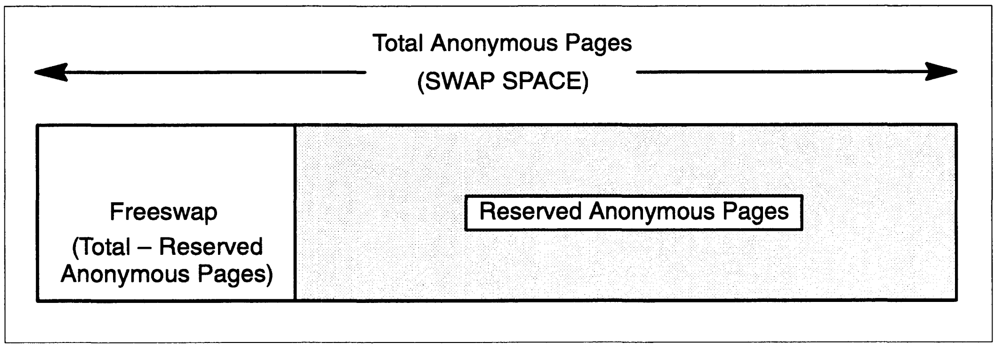


Figure 3–1 Swap Space

Other private anonymous pages are allocated during program execution, most commonly by touching new pages on the stack and by calling **malloc**(), which allocates memory. These calls to **malloc**() increase the number of reserved anonymous pages and may fail if that would cause the reserved anonymous pages to exceed the number of total anonymous pages.

Shared anonymous pages, as the name implies, may be shared by many processes. Interprocess communication through shared memory segments is a common use of this type of page.

Once **exec**() has loaded a program and allocated memory for the executable portion, the initialized data portion, and the uninitialized data portion, the I/O caused by the program depends on the following:

- On what pages are the instructions that the program executes?
- Which data pages does the program touch?
- How does the program handle stack and heap allocation, and how much memory is available for creating new pages?
- Which files does the program read?

  Reading from a file causes the required pages to be brought into memory if they are not resident pages. Memory operations resulting from calls to **read**() and **write**() are counted as *non-fault* operations. The amount of physical memory available for these pages is controlled by the PERCENTBUF configuration parameter.

  It is possible for a program to execute from beginning to end without ever requiring disk I/O. The following must be true:

- The text and data pages for the program are already in memory from a previous execution.
- There are enough free pages in the swap area so that the system can create copy-on-writes and new stack and heap pages without throwing out dirty pages.
- The file pages being read from and written to are already in memory.

  Although pages written to the file system by a program may not reach the disk while the program runs, eventually the data is written to disk. The DG/UX system accomplishes this with a kernel LWP called the page cleaning daemon. This daemon runs periodically to make the file system memory image and the disk consistent. This policy ensures that in case of a system crash, data that programs have written to the file system at the time the page cleaning daemon last ran can be recovered from the disk. The MAXBUFAGE configuration parameter governs the age of unwritten file system data and the frequency at which the page cleaning daemon runs.

You are now familiar with the four classifications of memory that the DG/UX system uses to report memory statistics: program file pages, data file pages, private anonymous pages, and shared anonymous pages. A fifth classification, *kernel anonymous pages*, consists of memory in use by the kernel. This section has considered typical uses of each of the four classifications of user-accessible memory, but other usage patterns may show up in the statistics. For example, the **memctl** and **mmap** system calls allow programs to create portions of their address space with any of the page classifications we've discussed (with the exception of kernel anonymous pages). Some forms of I/O, such as that for metadata, are not reported by the memory statistics.

When the last LWP of a process exits, the process's anonymous pages may be reclaimed, but the file system pages (program file and data file pages) remain in memory for some time, depending on the demand for memory. A second execution of the same program will encounter mostly soft faults, and typically will execute much more quickly.

Here is a summary of the five types of pages:

Program file pages	Pages that are executable and mapped to a file in the file system. These pages are read-only.
Data file pages	Memory resident pages from files that are only read from or written to (in other words, files that are not executed). These pages are typically shared amongst all processes that are accessing the file in question, and changes made to the file by any of the processes are immediately seen by all the processes.
Private anonymous pages	Pages available for uninitialized data, along with an initial allotment of pages for a stack. These pages are allocated in a system's swap area.
Shared anonymous pages	Pages that may be shared by many processes. Interprocess communication through shared memory segments is a common use of this type of page.
Kernel anonymous pages	Memory in use by the kernel.

Swapping

As described above, a program's data is eventually written to disk. However, when the system is extremely short of memory, active LWPs may be "swapped out"; that is, the LWPs are moved from memory to disk so that other LWPs can run.

Adding swap areas on disks where there are not currently any may improve performance; see the **sysadm** File System –> Swap Area menu. Do not skimp on the system's swap area to save disk space. Performance never improves when you must go to disk, since pages are written to the swap area because there is not enough memory to accommodate all LWPs running at a given time. If your system mixes high-speed disks and low-speed disks, use the fastest disks for all your swapping.

The best way to distribute traffic evenly over multiple disks is to use several equal swap areas, one per disk. The reason to keep these areas roughly equal in size is to keep their percentage of free space roughly equal as well. Greater free space percentages imply less fragmentation, which improves efficiency.

You can use the freeswap value reported by **nsar –r** to find out how much swap space is available. (Multiplying freeswap by 512, then dividing by 10^6 converts blocks to megabytes.) As a rule, the freeswap value should be 15–30% of the total physical memory and swap area on the system. For example, if your system has 256 MB of physical memory and 384 MB of swap area space, the sum is 640 MB.

15% of 640 MB = 96 MB, or 196,608 blocks
30% of 640 MB = 192 MB, or 393,216 blocks

For this system, if the freeswap value is usually under 200,000 blocks (or frequently under 100,000 blocks), you should probably increase the swap area space. On the other hand, if the freeswap value is typically over 400,000 blocks (and rarely under 300,000 blocks), you can probably decrease the amount of swap space to recover disk space.

Systems with large bursts of swap usage will need a reserve larger than 15–30%, while systems with more static swap usage will need a smaller reserve. Note that swap space requirements are highly application dependent.

Thrashing

If interactive users notice long pauses in response time, the pauses may be the result of anti-thrashing measures by the kernel. (Thrashing is when non-interactive LWP groups simultaneously compete for inadequate resources.) To verify this, see if the rate of binds (Process Binds/sec using UX/RPM, swpin/s using **nsar –w**, Binds/Sec using **sscope**) is greater than the rate of forks (Fork System Calls/Sec using UX/RPM or **sscope**, fork/s using **nsar –c**). Also, check to see if the number of unbound runnable LWP groups is consistently non-zero (Unbound Runnable Processes using UX/RPM or **sscope**, swpq–sz using **nsar –q**).

If these statistics lead you to believe that slow response time is the result of anti-thrashing measures, consider increasing memory, examining applications that seem to cause high frames purged and page fault rates (for per-process page fault statistics, try **nps –eo cmd,hfault,sfault**), or increasing the value of MINBOUND (the minimum number of user LWPs that can be ready to run in the kernel). Note that by default, the system determines MINBOUND dynamically based upon system load; by specifying this value yourself and guaranteeing that a specific number of LWPs can be ready to run, you are overriding system algorithms. This can either improve performance or increase thrashing. See Chapter 2 for a description of MINBOUND.

Hard Page Faults

As a basic guideline, hard page faults should probably be kept under 15 per second; you can monitor hard page faults with UX/RPM's or **sscope**'s Hard Page Faults or **nsar –p**'s vflt/s statistic. A rate of 15 per second generally indicates increased disk I/O and unnecessary faulting. However, you should determine what your system's threshold value is for hard page faults by regularly monitoring the system; if your system performance is good, 15 per second may be perfectly acceptable.

Per-Process Memory Statistics

Descriptions of per-process memory statistics are given below. Note that the application that you use or the command that you type is shown to the left of the colon; the field of interest to you is on the right of the colon and should not be typed in:

File system operations	**Input**	UX/RPM: fsinop nps –o fsiops: FSIOPS	Number of file system input operations.
	Output	UX/RPM: fsoutop nps –o fsoops: FSOOPS	Number of file system output operations.
Page faults	**Hard**	UX/RPM: hfault nps –o hfault: HFAULT	Number of hard page faults.
	Soft	UX/RPM: sfault nps –o sfault: SFAULT	Number of soft page faults.
Resident memory	**Current size**	UX/RPM: rss nps –o rss: RSS	Size of the resident portion of the process's address space, in kilobytes.
	Maximum size	UX/RPM: maxrss nps –o maxrss: MAXRSS	Maximum resident set size, in kilobytes.
Resident shared memory		nps –o xrss: XRSS	Sum of shared program file, data file, and shared anonymous pages, in kilobytes.
Resident unshared memory	**Size, excluding stack pages**	nps –o drss: DRSS	Private unshared or COW-shared anonymous pages, excluding stack pages, in kilobytes.

Size, stack pages nps –o srss: SRSS

Private unshared or COW-shared stack pages, in kilobytes.

System-Wide Memory Statistics

Descriptions of system-wide memory statistics are given below:

Anonymous pages

Allocated nsar –W O%swap: %swap

Percentage of total swap space that has been allocated.

Free UX/RPM: Free Anonymous (frames)
nsar –r: freeswp
sar –r: freeswp

Number of 512-byte blocks that are available in swap space.

Reserved UX/RPM: Reserved Anonymous
sscope: Reserved Anonymous Pages

Number of anonymous pages that have been reserved, but not necessarily accessed. Applications reserve anonymous pages for their uninitialized or bss data sections, for heap space when they call **malloc**(3C) or **brk**(2), for stack frames, and when they modify initialized data pages. Attempts to reserve anonymous pages that would make this number equal Total Anonymous Pages will fail and cause the system to report that it is out of swap space.

You can configure more swap space by using **swapon**(1), available from **sysadm**(1M).

Total UX/RPM: Total Anonymous
sscope: Total Anonymous Pages

Number of pages of physical memory and on the swap device that may be used for virtual address space. Anonymous pages are distinct from file pages. Anonymous pages must be written to disk when they are removed from memory.

Bound frames purged

UX/RPM: Bound Frames Purged
sscope: Bound Frames Purged/Sec
nsar –w: bswot/s
sar –w: bswot/s

Number of resident pages that were taken away from bound LWP groups.

Purging bound pages is expensive because bound LWP groups will likely run again soon and require access to the purged pages, resulting in hard page faults. The DG/UX system favors reclaiming pages from unbound LWP groups. A non-zero value of this statistic indicates insufficient memory.

Frames purged

UX/RPM: Frames Purged
sscope: Frames Purged/Sec
nsar –g: pgfree/s
sar –p: rclm/s

The number of pages removed from main memory by the system to make room for other pages. Purging is initiated when the number of free memory frames reaches a minimum level. The frame purger then scans memory frames, looking for the least recently used eligible frames to replace. This number includes all purged pages, such as anonymous, bound, and unbound pages (see Bound Frames Purged). Pages freed voluntarily, such as pages belonging to an exiting LWP, are not counted as purged.

Possible kernel parameters that may assist in decreasing high values (values that indicate poor system performance on your machine) are MAXBUFAGE and PERCENTBUF. If applications on your system are writing randomly to a large number of pages, decreasing MAXBUFAGE from the default of 60 seconds to 30 seconds may improve overall performance by making the frame cleaner run more often, with less to do in each pass.

If a lot of file I/O on your system is causing programs to be pushed out of memory, decreasing PERCENTBUF from its default of 100% to 50% or less ensures that executables will remain longer in memory.

With large memory configurations, the page cleaning daemon can use a substantial amount of time searching for modified data pages. The amount of time is based upon MAXBUFAGE and PERCENTBUF. If you suspect that your system time is high due to page cleaning, increase MAXBUFAGE and PERCENTBUF.

Other kernel parameters that may need fine tuning are MAXPAGEOUTS, MAXSLICE, and HOGFILESIZE.

See Chapter 4 for descriptions of MAXBUFAGE, PERCENTBUF, and HOGFILESIZE; see Chapter 2 for a description of MAXSLICE. MAXPAGEOUTS is described later in this chapter.

Other solutions are to add more physical memory, redesign applications to use memory more efficiently, or reduce the number of users allowed on the system at one time. UX/RPM displays the number of users on its overview screen.

Free memory frames

UX/RPM: Free Memory (frames)
sscope: Free Memory Frames
nsar –r: freemem
sar –r: freemem

Number of memory pages available for immediate allocation, including pages in memory objects that are not mapped by bound LWP groups, and pages of non-open files. This value is an indication of how much cushion there is in meeting processes' demands for memory without purging. Except for a few dedicated memory areas like the free memory pool, the DG/UX system will use all of memory for buffering, unless you've set PERCENTBUF. If you've set PERCENTBUF, using the free memory frames value to evaluate memory usage is difficult. Also, the DG/UX system tries to keep potentially useful information such as recently executed programs and recently read files in memory. Using the rate of frames purged is a better indicator of how busy your virtual memory system is; see Frames Purged.

Kernel Memory Allocation

Non-pageable Allocation nsar –k: (npg mem) alloc

Number of bytes allocated to requests from the kernel for non-pageable memory.

Non-pageable Memory Pool nsar –k: npg mem

Non-pageable memory pool size, in bytes. The memory pool size is dynamic and will always be larger than allocations from it. The maximum size for this memory pool is equal to main memory size.

Non-pageable Request Failure nsar –k: (npg mem) fail

Number of requests for non-pageable memory that have failed.

Pageable Allocation nsar –k: (pg mem) alloc

Number of bytes allocated to requests from the kernel for pageable memory.

Pageable Memory Pool nsar –k: pg mem

Pageable memory pool size, in bytes. The memory pool size is dynamic and will always be larger than allocations from it.

Pageable Request Failure nsar –k: (pg mem) fail

Number of requests for pageable memory that have failed.

Memory Usage

Data File Pages
UX/RPM: Data Pages
nsar –W Odfile–res: dfile

Number of resident pages that are data file pages.

Kernel Anonymous Pages
UX/RPM: Kernel Anon
nsar –W Okanon–res: kanon

Number of resident pages that are kernel anonymous pages. These pages include not only pages allocated from the memory pools, but also all of the kernel wired text and data that is initially loaded from the boot image. A significant amount of kernel data structures are dynamically allocated from the wired memory pool in particular (for example, STREAMS and device driver structures).

Program File Pages
UX/RPM: Program Pages
nsar –W Opfile–res: pfile

Number of resident pages that are program file pages.

User Private Anonymous Pages
UX/RPM: Private Anon
nsar –W Oupanon–res: upanon

Number of resident pages that are user private anonymous pages.

User Shared Anonymous Pages
UX/RPM: Shared Anon
nsar –W Ousanon–res: usanon

Number of resident pages that are user shared anonymous pages. These pages may exist in the address space of more than one process, such as a shared memory segment.

Page Faults

User
UX/RPM: User Faults/Sec
sscope: User Page Faults/Sec
nsar –w: bswin/s
sar –w: bswin/s

Number of page faults accumulated by user processes (the faults are actually taken by LWPs). This number includes hard page faults, faults satisfied by pages already in memory (soft page faults), and copy-on-write faults. It also includes faults on pages of remote mounted file systems. It does not include page faults taken by the DG/UX kernel.

Hard	UX/RPM: Hard Faults/Sec sscope: Hard Page Faults/Sec nsar –p: vflt/s sar –p: vflt/s

Page faults due to referencing pages not in memory and which must then be read from disk, either from local or remote-mounted file systems. These faults are rather expensive: there is the disk read cost and the file system overhead.

See Frames Purged.

Fill from file	UX/RPM: Fill Faults/Sec sscope: Fill From File Page Faults/Sec sar –p: pgfil/s

Number of hard page faults satisfied by reading from a file as opposed to reading from the swap area. This includes hard faults on all mapped files, whether mapped by **exec**(2) or **mmap**(2). The difference between Hard Page Faults and Fill From File Page Faults is the number of swap area page faults.

An increase in **forks** can increase this value since pages that contain code and initialized data must be read in from disk.

Soft	UX/RPM: soft faults nsar –p: atch/s

Page faults satisfied by reclaiming a page already in memory. These faults occur when a page is mapped in the address space of a process, but is not marked resident because it has not been referenced yet by that process (even though the page is resident in memory). Such a fault obviously does not require I/O, but does require that the page table entry for the faulting process be changed to reflect the residency of the page being mapped. You incur the cost of taking the exception and changing the page table entry, but you needn't go to disk to service the fault. These faults are inexpensive.

	Copy on write	UX/RPM: COW/Sec sscope: Copy on Write Page Faults/Sec nsar –p: pflt/s sar –p: pflt/s

User page faults that result in the creation of private copies of shared pages. The shared pages copied by these operations fall into two categories:

1. File pages that have been mapped private by either an **exec**(2) function or **mmap**(2) and then have been modified, such as when a program assigns a new value to an initialized variable for the first time.

2. Privately–mapped anonymous pages that are shared among multiple processes as a consequence of **fork**(2).

Copy-on-write faults may also be hard faults, and always require copying a page. A copy-on-write fault is hard when the page to be copied from is not in memory and must be fetched. This value generally goes up as the **fork** rate goes up.

Page-In	**Pages**	nsar –p: ppgin/s

Pages paged-in per second (system-wide). Each request can bring in more than one page.

	Requests	nsar –p: pgin/s

Number of page-in requests (system-wide). This number includes hard page faults and file faults (such as file page-ins due to read and write system calls).

Page-Out	**Operations**	nsar –g: pgout/s

Number of page-out operations.

	Pages	nsar –g: ppgout/s

Number of pages paged-out per second. This value may be greater than the number of page-out operations because a single operation may page-out several pages.

System Paging Statistics

In addition to the standard statistics listed above, **nsar** reports statistics on system paging activity. The following statistics describe system paging activity, both in terms of the number of pages affected and the number of paging operations. The syntax of the **nsar** command for system paging activity is:

nsar –W O*operation–pagetype–units*

where *operation* is a paging operation type, *pagetype* is a page type, and *units* is a units type. Each of the variables is referred to as a syllable.

These are the paging operation types:

Syllable	Name	Description
hfault	hard faults	Requires physical I/O (read from disk and/or the file system).
sfault	soft faults	Requires page table manipulation but no physical I/O.
hnfault	hard non–faults	Page-in operation that is not caused by a page fault.
snfault	soft non–faults	Explicit request that references a resident or zero-fill-on-demand page.
cfault	copy–on–write page faults	Operations in which a private copy of a page is created. Each of these operations will have already been counted in one the the four previous types.
rdirty	replace dirty page	New pages made available by writing modified or “dirty” pages to backing store.
rclean	replace clean page	New pages made available by making unmodified or “clean” pages non-resident.
fclean	forced page clean	Explicit request to clean a “dirty page.” Modified pages written to backing store by the memcntl(2) MC_SYNC operation or fsync(2).
uclean	unforced file page clean	Modified file pages written to backing store by the file page cleaning daemon.

The syllables for page types are:

Syllable	Name
kanon	Kernel anonymous pages
upanon	User private anonymous pages
usanon	User shared anonymous pages
pfile	Program file pages
dfile	Data file pages

The syllables for specifying the units of these statistics are:

Syllable	Name
op	Number of operations or requests
pp	Number of pages affected. The number of pages affected is always greater than the number of operations/requests because each operation can affect multiple pages.

Here is an example of using **nsar** to report paging statistics (three times at 10-second intervals:

```
% nsar –W Ohfault–pfile–op,hfault–dfile–op,hfault–upanon–op 10 3 ↵
```

This reports the number of hard faults in program file pages, data file pages, and user private anonymous pages. For each sample this will print a line of the form

```
00:00:05   hf-pf-op/s   hf-df-op/s  hp-up-op/s
00:00:15            2            0          12
```

UX/RPM presents these system paging statistics on additional virtual memory windows. Both system paging requests and operations per second are displayed.

Memory Configuration Variables

The following message configuration variables are also listed in **/usr/etc/master.d/dgux**:

PERCENTLOCKABLE

Specifies the percentage of physical memory available for locking by user processes, provided through the **memcntl**(2) system call. The kernel automatically rounds down any reservation requests that would otherwise impinge on memory used for the kernel itself. The default is **10**. Note that reserving and then locking large amounts of memory may deadlock the system. Conversely, not reserving enough may cause programs that need to lock memory to fail.

UPOOL_MIN

Specifies the minimum size, in megabytes, to make the kernel pageable memory pool. A value of **0** (the default) tells the system to choose a reasonable size based on the size of physical memory. Certain small memory systems that use certain classes of devices may need to set this value. Note that unnecessarily setting this value on a small memory system will increase the amount of memory used by the kernel.

CONFMEM

Specifies the configured amount of physical memory, in megabytes, in the system. A value of **0** (the default) tells the system to use all the available physical memory in the system. Specifying a non–zero value allows the system to be configured to use less than the amount of usable physical memory (mostly useful for stress testing).

MAXPAGEOUTS

Specifies the maximum number of concurrent pageout I/O operations that the system can have outstanding at one time. The default is **0**, indicating that the system should pick a reasonable value based on the amount of physical memory present on the system.

This assumes that the system's effective pageout throughput is relatively balanced with main memory size. A small memory system which pages out to a large number of disk spindles may achieve better performance by increasing this value. Conversely, a large memory system which has few spindles effectively used in pageouts may achieve better performance by decreasing this value.

A reasonable guideline is to set this variable to a small multiple of (one to three times) the number of disk spindles which are actively involved in pageouts. This excludes disks which are written to less than others, as well as disks written using only raw or unbuffered I/O (which database management software often uses).

Setting this value too high can cause high latencies for other accesses to the disks used in pageouts. Setting this value too low can artificially limit the system's effective pageout throughput, which can cause delays for memory allocation. Latency is the time it takes for a sector (with the data you want) to revolve under a disk's head.

Shared Memory Configuration Variables

The tunable parameters shown below are associated with interprocess communication shared memory; they are dynamic variables. These parameters are also defined in the **/usr/etc/master.d/dgux** file.

SHMMNI
Specifies the maximum number of shared memory identifiers system wide. Each entry contains 52 bytes. The default is **1024**.

SHMSEG
Specifies the number of attached shared memory segments per process. The default is **256**.

SHMMAX
Specifies the maximum shared memory segment size in bytes. The default is **4*1024*1024** (4 MB). The shared memory maximum should be equal to or greater than the size of the shared memory area defined by an application system.

SHMMIN
Specifies the minimum shared memory segment size in bytes. The default is **1** byte.

End of Chapter

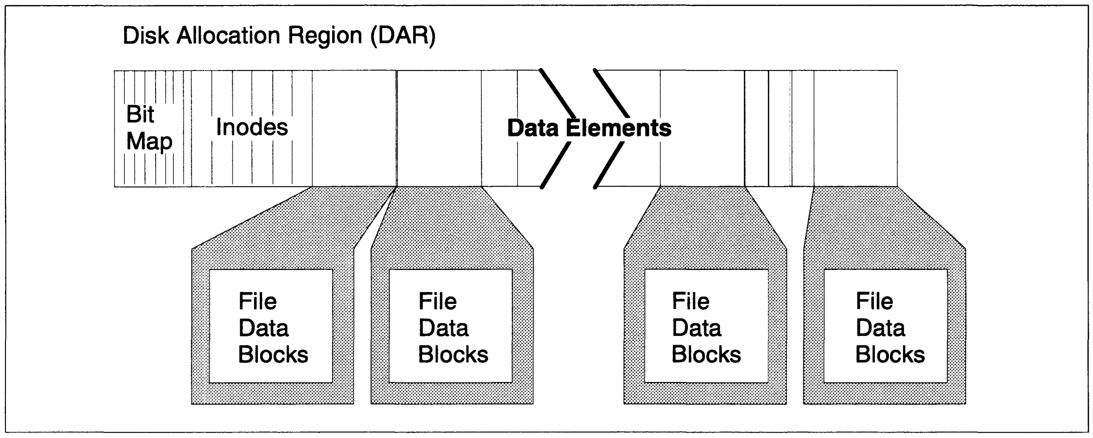
4 File Systems and Disk I/O

This chapter discusses several topics related to I/O and file systems, including:

- Disks
- File systems
- Virtual disks
- File system tools
- Disk I/O statistics
- File system configuration variables

Here are some terms that are used in this chapter:

To increase DG/UX file system performance, the disk storage of a file system is divided into *Disk Allocation Regions* (DARs). To access a file, the DG/UX file system alternately reads a file's inode (to find where the file's blocks are stored) and the blocks themselves. By using DARs, a file system can keep a file's data blocks and inodes physically close together, minimizing seek time.



An *inode* contains all the information pertaining to the mode, type, owner, size, and location (of the blocks) of a file. A unique *inode number* identifies each file in a UNIX system's flat file structure. Pointers in an inode tell the file system's Flat File Manager (FFM) where a file's data elements are stored.

A *data block* is a block of data that is stored on a virtual disk. Data blocks are 512 bytes, and are typically equal to the underlying physical disk's sector size.

A *data element* is the logical granularity at which the DG/UX file system transfers a file's data. The default data element size on DG/UX systems is 8 KB, which is sixteen 512 byte disk sectors. As the figure shows, data blocks are stored as data elements in a Disk Allocation Region.

A *virtual disk* is a software abstraction that enables you to construct a file system that appears as if it were a single sequential collection of disk blocks, even though it may span multiple physical disks. All virtual disks are associated with one or more physical devices, usually disks. Virtual disks enable the DG/UX operating system to manage files the same way, regardless of how the files are stored physically. Additionally, you can manipulate virtual disks online (such as renaming, copying, moving, and expanding them).

In order for a virtual disk to be accessible for mounting as a file system, the virtual disk must be a *volume*. Every virtual disk created with a non-null name is made a volume by default. Virtual disks that are volumes have device node entries such as **/dev/dsk/***foo* and **/dev/rdsk/***foo*; it is via these nodes that virtual disks are accessed.

File system *metadata*, in the context of the DG/UX demand-paged file I/O system, is "data about data." This is data, such as inodes, index elements, and directory information, that the file system uses to describe and locate files. File system metadata is cached in the kernel's data cache.

Mirroring is the technique of writing the same data to separate virtual or physical disks at the same time. If one disk fails, the data is still available on the mirror disk. Mirrored disks can also perform read operations faster than single disks because the system can simultaneously read from each of the mirrors. You can set up software disk mirroring (described in this chapter) or hardware disk mirroring (described in disk-array documentation).

Striping is the technique of distributing (or interleaving) data across several disks so that data can be accessed in parallel, increasing disk I/O performance. The DG/UX operating system supports striping at both the software and hardware levels. Software-level striping works within virtual disks. The Data General high availability disk systems support hardware-level striping across disk modules in a disk group. Hardware-level striping is part of the RAID 5 design, which provides uninterrupted access to data if a disk module in the array fails.

A *raw disk* is a disk that is being accessed in character mode. Character I/O can be used with disks to allow unbuffered transfers of an arbitrary number of disk sectors. Applications can bypass virtual memory by performing character I/O operations to disk. For example, a database management program might use character I/O to manage its own disk transfers. Disk transfers are subject to a device's alignment and granularity requirements (usually 512 bytes). Physical disks and virtual disks are accessible in character mode; character nodes for physical disks are in **/dev/rpdsk**, those for virtual disks are in **/dev/rdsk**.

A *block disk* is a disk that is being accessed in block mode—block I/O transfers are of a fixed size and must go through virtual memory. Disks accessed in block mode have no alignment or granularity requirements. For example, you can read a single byte from a block disk. Physical disks and virtual disks are accessible in block mode; block nodes for physical disks are in **/dev/pdsk**, those for virtual disks are in **/dev/dsk**.

Disks

The best way to improve I/O performance is to avoid going to disk. This is accomplished by using a large enough buffer cache, using correct element sizes, and keeping frequently used data in close proximity.

If heavy data file usage is hampering system performance by provoking excessive paging, you may want to reduce the configuration parameter PERCENTBUF (which specifies the maximum percentage of physical memory that can be occupied by data files). The idea is to free memory used by file I/O to allow more memory for code.

Buffering

Buffering is best when it is used to hold shared data. Database products need special consideration. Generally, database data is private to the database, and the database software controls and provides access to all users. Since the database software should have the ability to best understand the use of the data and its buffering needs, database applications often achieve the best performance by allowing the database software to buffer the data internally. Hence, the following tips are important for tuning a system supporting database applications:

- Increase the data buffer size of the database. Since database tuning varies from product to product, please refer to your database administrator's manual.

- Decrease the value of PERCENTBUF to avoid excessive paging as described above and leave memory available for allocation by the database application.
- Set up the database to use raw disk (i.e., **/dev/rdsk/***foo*). I/O to a raw disk is faster than synchronous I/O to a buffered disk, and the additional buffering gained by using the file system does not help if the database is doing a fair job of buffering the data.

  When using block disks, the maximum size of a buffered I/O operation is equal to the data element size assigned to the block special file node that represents the disk. When using raw disks, you can attempt to write up to 2 GB (although data types may be a limiting factor). However, the system will break a large request into smaller ones if it cannot safely allocate enough wired memory to support the request, or if the disks or SCSI interface cannot support the requested transfer size.

Metadata Buffering

Using **nsar –b**, get the average %rcache and %wcache for your system; these file system cache hit ratios measure the effectiveness of file system metadata buffering. On most systems, you should expect an average %rcache ratio of 95 or better. You can also use UX/RPM to check Read Cache Hits. The %wcache is generally much lower (down to around 65%), depending on the size of data files and the type of I/O being done.

If your system's cache hit ratios fall beneath these percentages, it may be possible for you to improve these rates by increasing the kernel parameter PERCENTSYSBUF (which specifies the percentage of physical memory reserved for system buffers) and decreasing MAXSYSBUFAGE (which specifies the maximum time that metadata will remain in system buffer caches before being written to stable storage).

On the other hand, if your system's read cache hit ratio is consistently 99–100%, you might want to decrease PERCENTSYSBUF so that %rcache is 97–98%. To determine the current percentage of physical memory reserved for system buffers, you can use **crash** to read the value of cf_bm_percent_memory_for_system_buffers. For example, after invoking **crash**, enter this command:

```
> mr cf_bm_percent_memory_for_system_buffers 1 b ↵
cf_bm_percent_memory_for_system_buffers:   05
```

Using **nsar –a**, get the rate of inode entry searches (iget/s, or Iget Calls using UX/RPM). If this number seems high (greater than 100 per second), there may be unnecessary file searches or heavy NFS activity (typically, there is one inode entry search per NFS operation). Keep in mind that UNIX systems do not hash filenames within directories. Studies have shown that for directory sizes found in typical UNIX environments, a simple sequential search is quicker than a hashing implementation. If a directory contains thousands of files, then the sequential search can be quite slow and can adversely affect file lookup operations.

In particular, the **/dev** directory may be very large—this is especially a problem when you try to use **ttyname**(3C); if you need to know your TTY device name frequently, set the $TTY environmental variable and use that to retrieve the name.

The login procedure uses **ttyname** to search the **/dev** directory. To set the device name for the TTY that you are logging onto, use **login** with the **–d** device option. You can modify **_pmtab** directly for your ttymon (look in **/etc/saf**) and add this option to each login line.

File System

The following sections describe how files are made on the DG/UX file system.

File Data Element Sizes

The size of the file system's buffered I/O operations to disk is based on file data element sizes. The default data element size for a file system is 8 KB (16 blocks), but can be modified on a per file system basis by using either the **mkfs** or **tunefs** command. The data element size for a file can be specified at file creation by using the **dg_mknod()** call. Data element sizes must be a power of 2, and should be greater than 4 KB for better efficiency. The default data element size and other information about a file system can be dumped by using the **dumpfs** command; this command can be run by the root user with the file system still mounted.

The default element size is based upon the assumption that I/O is often sequential; after a read operation, another read is very likely to occur.

When data access is random and done with read/write operations that are much smaller than 8-KB units, adjusting the data element size becomes important. For example, if a program is doing random, 4-KB reads from a very large database (in relation to the size of the file system buffers) whose data elements are 8 KB,

- The system is reading an extra 4 KB of data for every 4 KB of data that is likely to be used.
- The extra 4 KB of data is wasting buffer cache that could be better used for another I/O request.

By adjusting the data element size to 4 KB, you can effectively increase the buffer cache size (more buffers are available) and improve the speed of the disk (4-KB reads instead of 8-KB reads). 4 KB is the smallest buffer size that the kernel will use for a file; using a data element size smaller than 4 KB will increase disk I/O and adversely affect performance.

In most situations, the default of 8 KB (16 blocks) is fine.

Keeping Data Close Together

The location of data in a file can affect the time it takes to access the data. The first ten data elements are directly accessed. Then, depending on the file's data element size and how deep into the file the data is located, the remaining data elements are accessed through single, double, or triple indirect pointers.

You can keep frequently used data in close proximity on a disk two ways. The first way is to group frequently used file systems close together.

The second way is to adjust anniversary sizes so that large, more permanent type files all have their data in the same Disk Allocation Region (DAR). As a file grows in size, the file system allocates more and more blocks out of the DAR that contains the file, until the file reaches its first anniversary size. The first anniversary size is a limit that, when reached by a file, tells the file system to start allocating a file's data into another DAR. The anniversary size limit protects against having any one DAR too heavily subscribed. The second anniversary size is the size limit that, when reached, causes the file system to stop allocating data blocks from the secondary DAR and starts allocating from another DAR.

Generally, small file systems work well with the default anniversary sizes. However, file systems greater than 300 MB and containing a few files work better when the first and second anniversary sizes equal the DAR size.

Note that disk fragmentation is unavoidable, but DG/UX file systems minimize the effect by limiting fragmentation to a DAR (see “Fragmentation” below). Building a file system with the wrong size DAR can inappropriately spread a file’s data across a disk—distributing a file’s data into multiple DARs. By adjusting the size of the DAR and/or the anniversary size, you can enjoy the benefits of the DAR without the side effect of frequent, long seeks between DARs when reading a single file.

Correctly setting the DAR size and anniversary size is more critical for sequentially accessed data than for data stored in large (or very small) data files that are randomly accessed. The Average Service (avserv using **nsar –d** or aver serv using UX/RPM) time can help you determine how close data is stored. As these times go up, the disk is having to do longer seeks. For example, on some SCSI disks, a 14 millisecond access is theoretically possible. For individual SCSI disks, Average Service times less than 20 milliseconds are excellent. Average Service times in the 25–35 millisecond time range are not bad. High values may indicate on-disk cache misses or, if you are using a striped file system, that the file system’s DARs are misaligned, causing higher seek rates.

The DAR size should be an integer multiple of the data element size. The recommended size is a multiple of 8 KB (16 blocks).

If there is one large file, use a single DAR. If there are a few large files, consider using a single DAR or fewer DARs than the default. However, using only one or two DARs on very large virtual disks may increase the time needed to search bitmaps and increase the CPU time required to process a write request.

Fragmentation

Fragmentation generally worsens over time as files are deleted and created. You should periodically look for and archive (or delete) unnecessary files, such as duplicate file names, log files, accounting system files, and idle files.

Multiple volatile file systems on one disk can cause “ping-pong” fragmentation. Each file system has its own free space, which routines must skip over to reach other file systems on the same disk. Try to balance the load per disk by storing a mix of static and volatile file systems on each.

You can use the **dumpfs** command to display information about DARs; the root user can run this command with a file system still mounted. When the “free data blocks histogram” shows that a DAR does not contain any large blocks (block sizes of 16 or larger), fragmentation may be an issue. Also, mixing files of different data element sizes hurts overall disk subsystem performance. You can eliminate fragmentation by archiving and then restoring the files in the affected file system.

The **dump** (or **dump2**) and **restore** commands are a good way to eliminate fragmentation. Files are restored one at a time, so a file’s data is pretty much contiguous. In addition, **dump** (or **dump2**) and **restore** maintain file “holes”: empty file data elements that increase a file’s extent, created by seeks. These holes use index pointer space but do not use data blocks.

Note that since **tar** and **cpio** see these holes as null data, those commands archive the nulls and restore them; after reloading via **tar** or **cpio**, a file that originally contained holes will actually consume more data blocks and the holes will no longer exist.

File System Size

By default, DG/UX file systems allow only the root user to write to a file system that is more than 90% full. This limit can be overridden with **mkfs**, **tunefs**, or **cpd**. Reducing the minimum free space reclaims some disk storage. However, the more full the system becomes, the longer it takes to find free space; also, data element sizes may become smaller. Searches for files whose data element sizes are smaller than 8 KB take longer than 8-KB searches (searches have been optimized for the default file system data element size). In general, try to keep read-write file systems less than 80% full.

Increasing the minimum free space probably will not increase disk performance; 10% is optimum for most situations.

To display the percentages of blocks and inodes that are in use for each mounted local file system (as well as the total number of blocks and inodes and the number of free blocks and inodes), use this command:

```
# admfsinfo -o diskuse -l ↵
```

When planning a virtual disk’s size, you need to consider file system overhead for the kind of file system (if any) you intend to put on the virtual disk. File system overhead refers to internal data structures, such as data allocation tables, that the operating system requires to manage file access in the file system.

When planning a virtual disk for a DG/UX file system, you need to make it at least 17% larger than the amount of space you intend to use in the file system. This 17% overhead includes the 10% reserved free space buffer and the internal structures that the operating system requires for tracking files and directories. For example, if you need a file system large enough to hold 100 MB of data, you should create a file system 117 MB in size. With no files, the file system will be around 4% full. After adding 100 MB of data, the file system will be around 90% full.

If a file system will contain read-only static data, you can decrease the amount of free space. Also, if a file system is extremely large (1 GB or more), you may want to decrease the amount of free space.

Note that the root file system in particular can fill up and then panic the system.

Virtual Disks

Once you have reduced the disk accesses, the next most important activity is to improve the speed of the disk accesses. This means that you may need to

- Balance the load between disks.
- Balance the load between controllers.
- Use or tune disk caching
- Use data striping
- Use disk mirroring
- Use memory file systems
- Use fast recovery file systems
- Consider other concerns, such as bad blocks, write verification, and file synchronization.

Balancing the Disk Load

You determine the disk load by looking at disk data such as percent busy (%busy using **nsar –d** or UX/RPM, Percent Busy using **sscope**), average wait (avwait using **nsar –d**, Average Wait using **sscope**), and average response (aver resp using UX/RPM, Average Response using **sscope**). The busy data gives you an idea of how much time the disk is spending servicing requests; generally, this statistic should be under 30%. Faster machines and servers are able to push %busy to larger values. The average wait time gives you an idea of how long requests are waiting in a queue before getting sent to the disk subsystem. For good system response, this value should be less than 15–20 milliseconds on individual disks.

For response time, 40 milliseconds is considered good for an individual disk. For a system disk (the disk that contains root) or a RAID 5 unit, 50 milliseconds is generally good.

Also look at the average queue (avque using **nsar –d**, aver queue using UX/RPM, and Average Queue using **sscope**) disk data: for good system response, it is important for the disk subsystem to be able to keep this number in the low single digits. If this number seems high, your system may need more disk modules, controllers, disk striping, and adjustment of file system parameters like DAR size and data element sizes. As this queue length increases, the average wait time will also increase.

With a multi-disk configuration of similar disks, the number of requests should be balanced across the disk units; writes are particularly expensive. Check the read and write requests (**r+w/s** using **nsar –d**, % tot using UX/RPM, and Read+Writes Reqs/Sec using **sscope**). Overall system performance could be bottlenecked if one or a few disk units in a multi-disk configuration handle most of the I/O load. To balance the load, check which file systems are mounted on the most heavily used disks and attempt to better distribute those file systems. You may need disk striping (discussed later in this chapter) to better distribute the load across disks. To see which file systems reside on each physical disk, use **sysadm** (Device–>Disk–>Physical–>List, answering "partitions" to the Listing Style query).

You can determine which virtual disks are bottlenecks by listing the I/O statistics per virtual disk with the command **nsar –d –WD=v** or with the UX/RPM disk screen. For **nsar** to list a virtual disk, it must be a volume. As described earlier in this chapter, every virtual disk created with a non-null name is made a volume by default. However, if a virtual disk is not a volume, you can use the **admvdisk** command with the **–o modify –vy** option to make it a volume.

It is quite possible for the system physical disk (the disk that contains the root virtual disk) to become a bottleneck. Normally, the system disk contains the first piece of swap, **/tmp**, and **/var**; many administrative log files are also written to the system disk. An application that does a lot of sorting can overtax **/var/tmp**. Also, print spooling uses the **/var** directory. Here are some possible solutions:

- Move **/tmp** and **/var** to less active physical disks. Those two virtual disks might be good candidates for software striping if they are heavily used.

- Create other swap virtual disk areas on less frequently used physical disks (one area per disk). Using multiple swap areas of equal size allows DG/UX to optimize swapping operations by using a round-robin system between areas (however, sequential memory pages are not mapped sequentially in the swap area).
- Do not put application executables on the same physical disk as root.

Balancing the Load Between Controllers

Balancing the load between controllers means that you want to keep all controllers as busy as possible, without overtaxing them. It is best to distribute data over multiple controllers. Refer to the specification guides for the type of disks and controllers you are using to determine limits. Also look at the disk I/O statistics from **nsar**, UX/RPM, and **sscope**.

On a CLARiiON, the dual-SP configuration may provide better performance and higher availability than a single-SP configuration. Using dual-SP, one host and one SCSI-2 adapter with one channel is connected by the SCSI-2 bus to two SPs.

To improve performance on the dual-SP, you can bind some disks on one SP and the other disks on the other SP. The SP that binds a physical disk determines the primary storage-system route to that disk; the route through the other SP will be available if a component in the primary route fails.

Disk Caching

Disk caching increases access rates to disk media by combining small fast storage devices (nonvolatile RAM or disks) with large slow disks. An application uses a fast device for read and write operations, while the operating system duplicates these operations on a larger device. The purpose of the configuration is to accelerate file system access for I/O-intensive applications without risking data integrity.

The primary caching configuration consists of a nonvolatile RAM (NVRAM) or battery backed-up random access memory (BBURAM) board functioning as the fast device (the *frontend*) while a physical disk functions as the slow device (the *backend*). Although the RAM board has a relatively small storage capacity, its superior I/O performance can boost the performance of I/O-intensive applications such as database management systems. DG/UX System 5.4 Release 3.00 also supports the use of a disk drive, preferably a fast one, as a frontend device.

RAM-based caches introduce the risk that a failure could lose the data in the cache before the system has a chance to write it to the more stable backend device. The ideal frontend device is nonvolatile or battery backed-up RAM or a fast disk, which provides the required speed as well as stability.

The disk functioning as the backend, meanwhile, provides greater storage capacity than a RAM device or fast disk device and has the added stability normally attributed to disk drives. The backend device must be a local disk.

The DG/UX system optimizes disk caching for accessing DG/UX file systems rather than for other data structures (such as databases built directly on virtual or physical disks). You may, nevertheless, use cached disks for any purpose that benefits from the accelerated I/O performance.

Note that caching a virtual disk may be redundant with file system buffering, which will cause no performance improvement. You may consider running your applications both with and without disk caching, then comparing results to see which configuration offers the best performance. With virtual disks, you can add and remove a cache from a file system while it is active and in use. You can experiment with disk caching without having to shut down your application or file system.

The following section tells how to get the most out of a cached disk configuration.

How Caching Works

As I/O requests arrive for the cached disk, the system allocates buffers in the frontend device to hold data for the backend device. These allocated buffers are considered either *clean* or *dirty.* A clean buffer is one whose data matches the corresponding buffer on the backend. For example, a buffer that was copied from the backend to the frontend for a read operation is considered clean because it contains the same data that is on the backend. A dirty buffer contains data that is inconsistent with the backend. For example, a buffer that was written by an application but has not yet been flushed to the backend is considered dirty.

As I/O access to the cached disk continues, the system allocates more buffers until the frontend device becomes full. Then, the cache flushes a dirty buffer from the frontend to the backend to make room for additional buffers. There is an asynchronous flushing LWP running in the background that does much of this flushing so that buffers are usually immediately available on the frontend.

The process of freeing buffers involves seeking out which buffers are the least frequently accessed and flushing their contents to disk (if dirty) and then flagging them as unallocated. The system is then free to allocate them for more I/O requests.

To determine which buffers are the least frequently accessed, the system maintains an *access weight number* for each buffer. Each time an I/O request accesses the buffer, the system increments its access weight number. When the time comes to reclaim buffers, the system can then compare the access weight numbers of the buffers to see which are most frequently accessed (and should stay in the cache) and which are least frequently accessed (and may be freed).

Tuning a Cached Disk

The DG/UX system's disk caching feature provides several parameters you can tune to optimize cache efficiency. These parameters are:

Cache reads
: Specify if reading from the cache is enabled. Answering "yes" causes data to be shipped from the backend device to the frontend for reading. Otherwise, data is read directly from the slow backend device.

Cache writes
: Specify if writing to the cache is enabled (when possible). Answering "yes" causes data to be written directly to the frontend, which subsequently sends the data to the backend. Otherwise, data is written directly to the slow backend device.

Asynchronous write policy
: This parameter is meaningful only if writes are cached. You can request direct writes from the frontend to the backend device without caching it under certain conditions:

  `Never:` The system will never write data directly to the backend. The advantage is that no redundant disk activity takes place, but the disadvantage is that cache performance may degrade if the frontend gets full.

  `All writes:` Any time a buffer is written to the cache, the buffer is flushed to the backend device. This may improve performance in caches doing more reads than writes. A disadvantage of this operation is that substantial redundant disk activity may take place.

`First write`: The first time a new buffer is written to the cache, it is flushed to the backend device. This helps to keep the cache clean when there are a lot of buffers that are written only once, such as for large sequential writes. However, it minimizes disk activity for those blocks that are written repeatedly. In general, "first write" is the best policy.

Cache only file system metadata

Metadata includes important file system data such as file system inodes, size, the date stamp, and owner. There are several instances in which caching file system *data* may not be useful, but caching the *metadata* is useful. For example,

1) A database manager application is responsible for regulating its own data transfers. Allowing the DG/UX system to cache file system data would be superfluous and probably ineffective.

2) An NFS server using a cache over a LAN also regulates its own buffering needs. Data received over a LAN would likely flood a cache's buffers.

Caching metadata only is useful in instances such as these. File system data is read from and written to the slow backend device directly, bypassing the frontend entirely.

Caching both file system data and its metadata is desired for a local file system.

If the cache is to contain a non–DG/UX file system, such as a database, do not elect to cache file system metadata only. Such a selection would prevent any data caching.

Read weight

Use the read weight parameter to assign relative priority to read operations that occur in a frontend device; a larger weight means higher priority. When a cache is searched for available space, buffers with lower weights are reused first. Thus, buffers with a high read rates can be retained in the cache longer.

If multiple cached virtual disks share a frontend, you can assign a higher priority to one cache by increasing its read and write weight values. Conversely, you may assign a lower priority to one or more remaining caches by decreasing their read and write weight values. The maximum read weight is 100; the default is **1**.

Write weight

Use the write weight parameter to assign relative priority to write operations that occur in a frontend device; a larger weight means higher priority. When a cache is searched for available space, buffers with lower weights are reused first. Thus, buffers with high write rates can be retained in the cache longer.

If multiple cached virtual disks share a frontend, you can assign a higher priority to one cache by increasing its write and read weight values. Conversely, you may assign a lower priority to one or more remaining caches by decreasing their write and read weight values. The maximum write weight is 100; the default is **1**.

Search Percentage

This is the percentage of the cache's frontend that the system searches when it is looking for a clean buffer or data block with the lowest read or write weight. If the system does not find a clean buffer, it flushes and reuses a dirty buffer with the lowest read or write weight. On the next search for a clean buffer, the search will begin where the last search ended, so eventually the entire cache will be searched. Setting the percentage to zero causes the first available buffer to be used. By default, when creating a cache, the search percentage is 10 percent.

Ideally, a cached disk provides a performance improvement by satisfying disk accesses using much faster memory accesses. There are two obstacles, however, that prevent disk caching from reaching this ideal level of performance:

- The frontend device is not large enough to contain all the data that applications will require of it; therefore, some I/O requests will require accessing the backend device for data not currently in the frontend. The ratio of I/O requests satisfied by the frontend device (*cache hits*) to the total number of I/O requests is called the *cache hit rate*. You want the cache hit rate, which is expressed as a percentage, to be as high as possible.

- In a cache used for writing as well as reading, the cache must at some point write, or *flush*, the data in its buffers to disk. If your application accesses the cache during a flush, or if your application causes a flush, it will have to wait, or *stall*, until the flush completes. You want stalls to occur as seldom as possible. Increasing the search percentage will bring down the occurrence of stalls.

By experimenting with the various parameters, you can find ways to maximize the cache hit rate and minimize the frequency of stalls for your cached disk. Once you have created the cached disk, use **sysadm**'s Device–> Disk–> Virtual–> Cache–> List to review performance statistics, and use **sysadm**'s Device–> Disk–> Virtual–> Cache–> Modify to adjust the operating parameters.

For more information on caching, see *Managing Mass Storage Devices and DG/UX File Systems*.

If you have a CLARiiON system configured as RAID 3 or RAID 5, there is a cache option available that can help performance significantly. If you have a lot of output to the disks, you may want to configure all the cache as write cache instead of configuring it as read/write cache.

Software Data Striping

Depending on the nature of your applications, you may find that data striping improves disk I/O performance. You can implement data striping through the hardware (if you have a disk array) or through the software.

To implement software data striping, you need to create a virtual disk with this purpose in mind. Once you have created a striped virtual disk and its file system, striping is transparent to your applications. You use and manage the striped file system just like any other file system. The only difference is that you cannot change the size of a striped virtual disk; you cannot expand or shrink it.

Applications that perform a lot of random I/O (reads as well as writes) and applications that perform a lot of sequential reads can benefit from data striping. Also, striping is often used to balance disk load. Data striping may not help applications that perform a lot of sequential writes.

If your application does not appear suited to striping, do not attempt to implement striping: striping can have a negative impact on performance for inappropriate applications.

Data striping involves placing consecutive file data elements in the file system so that they alternate from one partition of the virtual disk to the next. For example, in the case of a striped aggregation of partitions, each partition must be the same size and each must reside on a different physical disk. The system will place the first data element in the first partition, the second data element in the second partition, and the third data element in the third partition, and so on. The data elements alternate this way so that consecutive data elements are stored on alternating disks.

The performance advantage results not only because you have distributed the I/O load across three disks, but also because you are using the hardware's read-ahead implementation to get the next sectors on that disk and the operating system's read-ahead implementation to get the next data element, even before you have explicitly requested it.

The partitions must all be the same size, and the size must be a multiple of the stripe size (16 blocks by default). The stripe size must be no smaller than the data element size, and it must be a multiple of the data element size (16 blocks by default).

IMPORTANT: Rotate the starting address of the first piece of each virtual disk across multiple physical disks, using a number that is a multiple of the stripe size.

IMPORTANT: The root (/) and **/usr** file systems must not be on striped virtual disks if you intend to boot images from them.

For information on software data striping, see *Managing Mass Storage Devices and DG / UX File Systems*. For information on hardware data striping, see your disk array documentation.

Software Disk Mirroring

You can improve the reliability and availability of your Data General AViiON system by using disk mirrors. You can set up disk mirroring through the hardware (if you have a disk array), through the software, or both. Software disk mirroring involves maintaining redundant virtual disks where all are "mirror images" of each other; they all contain the same data. The system manages access to the disks in a manner that is transparent to users.

Disk mirrors provide higher data availability by allowing your system to continue service to users even when disk errors occur. Disk mirrors also protect data integrity by maintaining redundant images of the same data.

Disk mirrors whose images lie on different physical disks offer increased throughput (and thus improve overall system performance) in environments where multiple concurrently running applications perform intensive reads of the mirror. This benefit arises because the system can use the images of the mirror as individual virtual disks during concurrent read operations, using one image to satisfy one read request while using another image to satisfy a different read request. Thus, the mirror distributes the I/O load across multiple disk drives.

Mirroring can slow performance of applications that are write intensive.

For information on how to create software disk mirrors, see *Managing Mass Storage Devices and DG/UX File Systems*. For information on hardware disk mirroring, see your disk-array documentation.

Memory File Systems

For applications that would benefit from very fast access to relatively small databases, you can use memory file systems. A memory file system is a portion of your computer's physical memory that is mounted. A memory file system behaves like a disk that has no rotational, seek, or controller overhead. You can access it the same as any file system. Also, memory file systems can speed access to files on diskless workstations. However, memory file systems are volatile—if the system loses power, the data in the memory file system is lost. For this reason, you should be able to recreate these file systems. For more information on how to create memory file systems, see *Managing Mass Storage Devices and DG/UX File Systems*.

Fast Recovery File Systems

To reduce the amount of time that the system requires to recover a file system after a failure, mount the file system with **fsck** logging turned on. With **fsck** logging, the system logs file system modifications to reduce the amount of time that **fsck** requires to verify the integrity of the file system. This feature is desirable for systems where rapid recovery and high availability are crucial. However, running with **fsck** logging turned on will incur a runtime performance penalty.

If you select **fsck** logging, the operation later prompts you for log size. Specify the log size in 512-byte blocks (32 or 64 blocks is average). There is a tradeoff in performance between log files of different sizes. A large log file improves run time performance but prolongs recovery time. A small log file degrades run time performance but reduces recovery time.

Other Concerns

Other concerns about structuring virtual disks for efficiency are bad blocks, write verification, and file synchronization.

Bad Blocks

The presence of bad blocks can mean that the bad blocks are getting remapped, thus decreasing performance. You can check for bad blocks by using the **sysadm** operation Device–>Disk–>Physical–>Bad Blocks–>List.

When the DG/UX system detects a flaw on a disk, it flags the block (a 512-byte portion of disk space) as bad. If a write operation was performed, DG/UX finds a good block to replace the bad block. The operating system takes care of redirecting reads and writes intended for the bad block so that they go to the replacement block instead. A part of the disk called the *bad block remap area* contains good blocks reserved specifically for this purpose: to replace blocks that go bad elsewhere on the disk.

With DG/UX System 5.4 Release 3.00, you can choose not to install remapping on disks that do it themselves, such as CLARiiON disk arrays. This increases performance slightly because blocks do not need to be checked before I/O to see if they are remapped.

Write Verification

In applications where data integrity is vital, you can benefit from write verification. By turning write verification on for a physical disk (use the **dkctl** command with the **wchk** option), you can be sure that data written to the disk is readable. You can enable write verification only for SCSI disks that support the feature; see your disk hardware documentation.

The disk hardware verifies every write operation by reading the written data off of the disk and comparing it to the data as originally received. However, the additional verification overhead in the hardware can have a significant negative impact on the performance of write-intensive applications.

File Synchronization

If you specify the **fsync_on_close** option to the **mount** command, whenever a file in the mounted file system is closed, all modified data and attributes of that file are written to disk. This option decreases the likelihood of data loss in the event of a system crash, but may degrade performance.

File System Tools

There are several tools available to help you tune file systems. What follows is an overview of each. For more information, consult the online manual pages.

Take care when tuning file systems, since file system performance may be improved for one application, but degraded for another.

mkfs

This utility is used to create an empty file system on a virtual or physical disk. The **mkfs** utility limits a file system to a maximum of 60 DARs by default (you can override this number), but if you grow the file system, this may increase. There are several options you can use to override the default file system settings such as the following:

- Inode density, which controls the number of inodes per DAR. The default is one inode per 3500 bytes (the actual density will be rounded down to an integral multiple of 64 inodes per DAR).
- Size, in blocks, of the DARs. The default DAR size is 1/60th of file system size if the file system is greater than or equal to 241,920 blocks; the smallest DAR size is 4032 blocks.
- Data and index element size for files
- Data and index element size for directories
- First anniversary size; the default is (DAR_size + 32) / 63
- Second anniversary size; the default is (DAR_size + 2) / 4

tunefs

This utility allows you to alter the data and index element sizes and anniversary sizes of any already existing file system. Running this utility will not affect files that are already stored in the file system, but the altered parameters will be applied to subsequent files. Note that the file system must not be mounted.

dg_mknod

With this system call, you can create a file and specify the data and index element sizes. You can control the data element size for a single file with **dg_mknod**, whereas **mkfs** and **tunefs** affect a complete file system.

dumpfs

This utility displays information about a file system. It shows you the settings of various file parameters, as well as a map of how blocks are allocated on a file system.

cpd

This utility displays the current allocation and the maximum allocation of blocks and file nodes for each control point directory named on the command line. You can also set the allocation limits for a control point directory. Since the root of every file system is a control point directory, you can use this command to change the maximum number of file nodes and blocks that a user can allocate.

Disk Arrays

In the context of disk subsystems, a disk array is a collection of one or more groups of disk modules and one or more SCSI busses that participate in a RAID redundancy scheme. Each group in an array appears to the operating system as a single physical disk. To improve performance on systems with disk arrays, the following may be helpful:

- Partition write-intensive portions of applications onto mirrors; this is using hardware disk mirroring, not software disk mirroring. For example, for typical database applications, you would want to have a mirrored pair for the system disk, work directories, and the database journal file, and one or more RAID5 groups for the read-intensive database files. A RAID5 group should not contain a swap area; that should be included in the system disk mirrored pair. For information on hardware disk mirroring, see your disk array documentation.
- The hardware stripe size should be larger than the data element size. Partitions should be a multiple of this size and should start on a stripe boundary. For information on hardware data striping, see your disk array documentation.
- Generally, if your hardware is striped or mirrored, striping or mirroring the software is redundant. However, software striping can be combined with hardware mirroring or RAID5. Such a combination allows better I/O balance among the various disks and can improve performance. A software striped RAID5 configuration has been described as a “plaid” configuration. This type of configuration requires very careful planning and alignment.

Tables 4–1 and 4–2 show the advantages and disadvantages of different RAID levels:

Table 4–1 RAID Level Performance Advantages

RAID 0	RAID 1	RAID 3	RAID 5
Maintains high I/O throughput in write-intensive applications	Maintains high I/O throughput in write-intensive applications	Maintains high data throughput in large file transfer applications	Maintains high I/O throughput in read-intensive applications
No write penalty from storing parity	No write penalty from storing parity	Parity information can be stored with a single write operation	See disadvantages

Continued

Table 4–1 RAID Level Performance Advantages

RAID 0	RAID 1	RAID 3	RAID 5
	Most efficient for combined read/write operations	Most efficient for transferring large contiguous files	Most efficient for random reads of many small files
	Suited for "fault tolerant" environments where cost is less of an issue	Suited for data transfer applications such as imaging and graphics	Suited for transaction-oriented applications
	Can mirror root and swap virtual disks	Cost-effective data redundancy for high-availability solution	Cost-effective data redundancy for high-availability solution

Table 4–2 RAID Level Performance Disadvantages

RAID 0	RAID 1	RAID 3	RAID 5
Failure/reliability not addressed	100% overhead for data redundancy	All drives are locked together, eliminating benefits of multiple actuators	Write penalty when storing data; 4 operations instead of 1
Not capable of automatic rebuild of data if one disk lost	Most expensive redundancy solution	Performs poorly with high numbers of small random reads	Minimum performance benefit with single-threaded I/O operations
Loss of one disk means loss of everything in a stripe	Requires a higher number of drives to achieve redundancy	Dedicated parity disk can become a bottleneck	

Disk I/O Statistics

Descriptions of disk I/O statistics are given below. Note that the application that you use or the command that you type is shown to the left of the colon; the field of interest to you is on the right of the colon and should not be typed in:

Busy

UX/RPM: %busy
sscope: Percent Busy
nsar –d: %busy
sar –d: %busy

The percentage of time the specified disk subsystem has spent handling requests over the interval between samples.

Directory blocks

UX/RPM: Directory Block Reads
sscope: Directory Block Reads/Sec
nsar –a: dirblk/s
sar –a: dirblk/s

Number of reads of directory blocks. This rate should probably be under 100 per second.

File System Cache misses

UX/RPM: Iget Calls
sscope: Iget Calls/Sec
nsar –a: iget/s
sar –a: iget/s

Number of times there was a file system name cache miss. It does not indicate the number of times the file system media was accessed. If this number seems high (perhaps over 100 per second), the file system may contain large directories—it possible, avoid searching large directories and put them at the end of your search path. Also, a search path variable (**path** or **PATH**), which lists the directories that the system should search when looking for a command, may be too long or not efficient.

File table size

UX/RPM: File Table Size
sscope: File Table Size
nsar –v: file–sz
sar –v: file–sz

Number of distinct open file descriptors. It includes descriptors for sockets, pipes, and local and NFS–mounted files.

Inode table size

UX/RPM: Inode Table Size
sscope: Inode Table Size
nsar –v: inod–sz
sar –v: inod–sz

The size of an internal local file system hash table. This value has very little application-level interpretation.

Metadata

Logical reads

UX/RPM: Log. Block Reads
sscope: MD Logical Reads

Number of blocks of metadata transferred by logical reads. This includes blocks that are cached and blocks that require physical reads from the disk. Metadata includes inodes and directory blocks.

Logical writes

UX/RPM: Log. Block Writes
sscope: MD Logical Writes

Number of blocks of metadata transferred by logical writes. This includes blocks that are cached or physically written to the disk. Metadata includes inodes and directory blocks.

Physical reads

UX/RPM: Phys. Block Reads
sscope: MD Physical Reads — Blocks

Number of blocks of metadata transferred by physical reads. These are blocks that are read from disk.

Physical read requests

UX/RPM: Phys. Read Requests
sscope: MD Physical Reads — Reqs
nsar –b: pread/s
sar –b: pread/s

Number of metadata read requests from the disk.

Physical writes

UX/RPM: Phys. Block Writes
sscope: MD Physical Writes — Blocks

Number of blocks of metadata transferred by physical writes to the disk.

Physical write requests

UX/RPM: Phys. Write Requests
sscope: MD Physical Writes — Reqs
nsar –b: pwrit/s
sar –b: pwrit/s

Number of metadata write requests to the disk.

Read cache

UX/RPM: Read Cache Hits
nsar –b: %rcache
sar –b: %rcache

The read cache hit ratio, i.e., the fraction of the number of logical reads which were found in the buffer cache.

Write cache

nsar –b: %wcache
sar –b: %wcache

The write cache hit ratio, i.e., the fraction of the number of logical writes which were found in the buffer cache.

Pathnames

UX/RPM: Pathname Searches
sscope: Pathname Searches/Sec
nsar –a: namei/s
sar –a: namei/s

Number of times pathnames have been resolved. This rate should probably be under 100 per second.

Queue

UX/RPM: aver queue
sscope: Average Queue
nsar –d: avque
sar –d: avque

The average number of requests waiting to be serviced. This is the ratio of change in response time to change in busy time.

For good system response, it is important for the disk subsystem to be able to keep this value in the low single digits.

Read system calls

Bytes transferred

UX/RPM: Total Characters Read
sscope: Chars Transferred by Read
nsar –c: rchar/s
sar –c: rchar/s

Number of bytes transferred by the **read** system call. This does not include all input to user applications, only that done by **read**(2) and **readv**(2).

This number can range from zero to a few million and includes terminal I/O as well as disk I/O. See Write System Calls.

Number of reads

UX/RPM: Read System Calls/Sec
sscope: Read System Calls/Sec
nsar –c: sread/s
sar –c: sread/s

Number of **read** system calls. The ratio of the number of bytes transferred by reads to this statistic indicates the average size of **read** calls. A small number of bytes per read may indicate a performance bottleneck because system call overhead contributes most to the cost of **read**. If the circumstances permit an application to read many bytes at once, it would probably perform better.

Reads

Number of reads

UX/RPM: blocks read
sscope: Reads — Blocks/Sec
nsar –W Odrblock: rblks/s

The number of 512-byte blocks read from the specified disk.

Requests

sscope: Reads — Reqs/Sec
nsar –W Odrreq: reads/s

The number of separate read requests from the specified disk.

Reads and Writes

Blocks

sscope: Reads+Writes — Blocks/Sec
nsar –d: blks/s
sar –d: blks/s

The number of blocks read from and written to the specified device. This is an indication of how much time the disk subsystem is spending actually transferring data. Some disk subsystems read much faster than they write. On such disks, you should consult the individual read and write statistics listed below.

Requests

UX/RPM: # of requests
sscope: Reads+Writes — Reqs/Sec
nsar –d: r+w/s
sar –d: r+w/s

The number of read and write requests to the specified device. Because a single request may transfer a small or large amount of data, the number of blocks transferred should also be consulted. The number of requests is an indication of time the disk subsystem may spend seeking.

Response time

UX/RPM: aver resp
sscope: Average Response

The average time (in milliseconds) a disk request spends being processed. This is the sum of wait time and service time on the specified disk subsystem.

Computed by dividing the change in response time by the number of requests in the interval.

Service time

UX/RPM: aver serv
sscope: Average Service
nsar –d: avserv
sar –d: avserv

The average time (in milliseconds) a disk request spends being processed by a disk unit. This does not include time spent in the device driver or in the disk queue. This is a measure of the actual disk speed, which varies with seek distance, amount of data transferred, and characteristics of the specified disk subsystem. Values between 20–25 milliseconds are considered very reasonable for most individual disks.

Computed by dividing the change in busy time by the number of requests in the interval.

Wait

sscope: Average Wait
nsar –d avwait
sar –d: avwait

The average time (in milliseconds) a disk request spends waiting to be serviced. This is a measure of contention for the specified disk subsystem. If there is a sudden burst of I/O on a disk's file systems, the wait time on that disk may become very large for a short time. To verify this, check the disk's average queue—it will be larger than normal.

Computed from the change in response time minus the change in busy time (which represents only requests that are waiting, not those being serviced) divided by the number of requests in the interval.

Write system calls

Bytes transferred

UX/RPM: Total Characters Written
sscope: Chars Transferred by Write
nsar –c: wchar/s
sar –c: wchar/s

Number of bytes transferred by the **write** system call. This does not include all output from user applications, only that done by **write**(2).

This number can range from zero to a few million (includes terminal I/O as well as disk I/O), but is usually less than the rate of bytes transferred by read system calls under a normal system load. Compare the two values to get an idea of the percentage of reads versus writes for an application. For example, you might have an application that issues ten **read** calls and five **write** calls; however, each application **read** call may physically read from disk twice—the ratio would be four reads for every write. If you change the application, check these values to monitor the effects of your changes.

Number of writes

UX/RPM: Write System Calls/Sec
sscope: Write System Calls/Sec
nsar –c: swrit/s
sar –c: swrit/s

The ratio of the number of bytes transferred by writes to this statistic indicates the average size of **write** calls. A small number of bytes per write may indicate a performance bottleneck because system call overhead contributes most to the cost of **write**. If the circumstances permit an application to write many bytes at once, it would probably perform better.

Writes

Number of writes

UX/RPM: blocks written
sscope: Writes — Blocks/Sec
nsar –W Odwblock: wblks/s

The number of 512-byte blocks written to the specified disk.

Requests	sscope: Writes — Reqs/Sec nsar –W Odwreq: writes/s

The number of separate write requests to the specified disk.

File System Configuration Variables

The file system configuration variables are also listed in **/usr/etc/master.d/dgux**. These variables set the file system parameters shown in the list below.

ACCTON, ACCTOFF

If the free space in the file system in which the accounting file resides becomes less than the percentage specified by ACCTOFF, no further accounting records will be written. When the free space reaches the percent specified by ACCTON, the writing of accounting records will resume. ACCTOFF should always be smaller than ACCTON. The default for ACCTON is **5**. The default for ACCTOFF is **2**.

MAXBUFAGE

Specifies the maximum amount of time, in seconds, that modified system data will remain in the buffer caches before being written to stable storage. The default is **60**. Setting the value higher lets data remain in the cache for longer periods without being cleaned, which can result in higher performance. However, because it is not flushed as often, the data may not make it to disk if there is a crash of the system.

MAXSYSBUFAGE

Specifies the maximum amount of time, in seconds, that modified system data will remain in system buffer caches before being written to stable storage. If this parameter is **0**, the default, the system uses the same maximum age value for system buffers that it uses for user buffers (as set in MAXBUFAGE). If you set MAXSYSBUFAGE to a value greater than MAXBUFAGE, the system ignores it and uses MAXBUFAGE for all buffers.

PERCENTBUF

Specifies the maximum percentage of physical memory that can be occupied by data files. The default is **100**, meaning data files can use as much physical memory as their usage pattern dictates. There should be no need to configure this variable unless heavy data file usage is hampering system performance by provoking an undesirable amount of other paging. For example, this condition might occur on an NFS server whose physical memory resources are small relative to the amount of file data accessed over a period of several minutes. With large memory configurations, setting PERCENTBUF too low will cause additional file page cleaner overhead.

PERCENTSYSBUF

Specifies the percentage of physical memory (after initialization) that is reserved for system buffers. System buffers hold directory, inode, file index, and bitmap data. The default, **0**, causes the system to select a reasonable value for the system.

HOGFILESIZE

Specifies the maximum number of bytes of physical memory that can be used by a given data file before that file will be treated unfavorably for physical memory resource allocation. The default is **262144**. When the system is forced to page data out to meet requests for memory, data files having more than HOGFILESIZE bytes buffered will be aggressively chosen to be paged out, and will be restrained in their ability to consume free memory.

NFSDEMONSCPUMASK

Specifies the set of CPUs on which the NFS daemons may run. You can use this to keep NFS daemons from running on certain CPUs in the system. For example, on a multiple CPU system, disk I/O interrupts may be serviced by CPU0. To avoid running the NFS daemons on CPU0, set the mask to 65534 (decimal). This parameter has no effect if the system is not a multiprocessor system, or if the mask does not specify any processors to run on. Note that this parameter can improve performance only on heavily loaded systems that are dedicated NFS servers; otherwise, you should not attempt to use this mask.

CDLIMIT

Specifies the maximum size in bytes that a nonsuperuser file may attain. The default is the constant **INT32_MAX**, which is equal to 2,147,483,648.

NFSLOCKUSERLIMIT
Specifies the maximum number of remote processes that can hold record locks concurrently. The default is **512**.

USERLOCKLIMIT
Specifies the maximum user locks that a process can hold. The default is **2048**.

FREEINODE
Specifies the maximum ratio of in-use inodes to free inodes in the system. To improve performance on systems where you open a large number of files repeatedly, set this parameter to a higher value. The default is **4**.

FREERNODE
Specifies the maximum ratio of in-use rnodes (remote-mounted inodes) to free rnodes in the system. To improve performance on systems where you open a large number of remote files repeatedly, set this parameter to a higher value. The default is **4**.

NCLIENTOPS
Specifies the maximum number of handles of a given RPC type available on the system. The default is **6**.

CHOWN_REST
A Boolean variable indicating whether or not the POSIX feature **_POSIX_CHOWN_RESTRICTED** is present for the system. The default is **0** (FALSE). Use **1** for TRUE. If the feature is present, then additional POSIX-style restrictions are placed on the **chown**(2) system call.

SRVNOTNEEDED
A Boolean variable indicating whether or not the DG/UX file system should hold on to buffers read from NFS clients. The default is **0** (FALSE). If this value is **1** (TRUE), then the server will mark buffers read by NFS clients as not needed anymore, which makes them more likely to be reused than other pages in the cache.

FULL_ISO9660
A Boolean variable indicating whether or not Full ISO 9660 filenames will be used for files on High Sierra compact discs. Unless this behavior is turned on, the High Sierra file manager will map upper case characters in filenames to lower case, and map ";" characters in filenames to "-" characters. The default is **0** (FALSE). Use **1** for TRUE. The default mapping is advantageous in a DG/UX environment.

End of Chapter

5 Terminal I/O

This chapter discusses terminal I/O (including statistics) and pseudo-terminals.

Just as the disk or any other device can become a system bottleneck, the asynchronous I/O controllers can slow terminal performance. Although the I/O rates of the controller will vary according to line discipline settings, the controller should be able to output several thousand characters per second. Check rate of output characters (outch/s using **nsar –y** or Output Characters/Sec using UX/RPM or **sscope**).

Input rates are much slower. Look at the number of characters read from terminal devices and processed in canonical mode vs. raw mode.

If you notice higher than normal characters/second rates after installing a new modem or multiplexor, check that it is not caught in an infinite loop. Also check that the setup is correct in **/etc/ttydefs**. For more information, refer to the online manual pages for **ttydefs**.

The most effective way to prepare a new device description is by imitating the description of a similar device in **terminfo** and building up the new description gradually, testing whether **vi** works with the compiled description. Begin by creating a terminfo source terminal file; you can obtain the source description for a given device by using the –I option of **infocmp**. You may copy and edit this description to accurately describe the device that you wish to enter into the terminfo database. Next, use the **tic** command to recompile the terminfo source terminal file. By default, the resulting binary files are placed under the directory **/usr/share/lib/terminfo**. However, if the environment variable TERMINFO is set, the compiled results are placed under the directory specified by the value of that variable.

For maximum terminal performance, make sure that your terminals and TERM environment variables are set to the most powerful emulations. For example, if you have a terminal that provides DEC VT52, VT100, and VT320 emulations, take advantage of the VT320 emulation. In this case, ensure that the TERM environment variable is set to "vt320" instead of the default "vt100."

Another problem that causes poor terminal performance is issuing a high amount of **stty** setting commands. Applications should use library calls (as few as possible) to change terminal settings.

Also, avoid changing back and forth between canonical and raw input modes as much as possible.

Terminal Lines

You should not add a ttymon port service on a line that is not connected to a terminal. Lines connected to devices such as printers, the asynchronous port on an uninterruptible power supply unit, or ports used for **mterm**(1) connections may produce noise on the line. Unterminated lines also chatter back to the system. Noise on the line can cause the port monitor to consume inordinate amounts of CPU time.

Terminal Port Interrupts

During normal operation, you should monitor the size of **/etc/wtmp** because this is the file from which the connect accounting is generated. If the file grows rapidly, execute **acctcon1** (see the manual page) to see which tty line is the noisiest. Use a command line similar to this:

```
acctcon1 -l outfile < /etc/wtmp
```

The output file *outfile* helps track line usage, identify bad lines, and find software and hardware oddities. Termination of **login**(1) and termination of the login shell generate logoff records, so the number of logoffs is often three to four times the number of sessions.

Generally, interrupts occur when a port service is first enabled and the system goes to multi-user mode. If the number of logoffs exceeds the number of sessions by a large factor, it usually indicates a faulty or failing multiplexer, modem, or cable connection. An unconnected cable dangling from the multiplexer can cause this as well. If interruption is occurring at a rapid rate, it can affect general system performance.

Editread

The **editread** command line history and editing facility of the Bourne shell (**sh**) and C shell (**csh**) can affect general system performance. When the Bourne or C shell prompts you to enter a command, **editread** changes the terminal settings to process characters in raw mode. After you finish entering the command line, **editread** changes the terminal settings back to canonical input mode before the shell runs the command. If many people actively use **editread**, the frequent changing of input modes can put additional load on the system. Conversely, high system load can cause poor **editread** performance.

To see if **editread** is contributing to a high system load, have the system users turn off **editread** and then you can monitor system performance without it. To turn off **editread** for an individual shell session, type this command:

```
% enable=OFF^R ↵
```

The "^R" represents **editread**'s reconfig command.

To turn off **editread** for several shell sessions, put the following commands in the indicated files:

```
EDITREAD="enable=OFF" export EDITREAD # /etc/profile

setenv EDITREAD "enable=OFF"         # /etc/login.csh
```

To turn off **editread** for the long term, delete the **.editreadrc** files in users' home directories.

The Korn shell (**ksh**), which provides **emacs** and **vi** interfaces, performs better than the Bourne shell with the **editread** interface. In particular, **vi** command line editing uses raw input mode only when you specifically activate it by typing the control mode key (the Escape key). For this reason, the Korn shell **vi** editing facility has a lower impact on system performance than **editread**.

If your users prefer the C shell interface and you decide that **editread** is causing a decline in system performance, system users can still access the history and command line substitution features of **csh**. While the line-oriented interface to these features may not be as convenient as **editread**'s interface, using **csh**'s features has a negligible impact on system performance. Consult the **ksh**(1) and **csh**(1) manual pages for more information on these alternative history and line editing mechanisms.

Terminal I/O Statistics

Descriptions of terminal I/O statistics are given below. Note that the application that you use or the command that you type is shown to the left of the colon; the field of interest to you is on the right of the colon and should not be typed in:

Characters

Canonical mode

UX/RPM: TTY Canonical Input Chars
sscope: Canonical Input Chars/Sec
nsar –y: canch/s
sar –y: canch/s

Number of characters read from terminal devices and processed in canonical mode. If this number is over a few hundred, investigate modem or multiplexor setups. See Raw mode.

Output

UX/RPM: TTY Output Characters
sscope: Output Chars/Sec
nsar –y: outch/s
sar –y: outch/s

Number of characters output to terminal devices.

Raw mode

UX/RPM: TTY Raw Input Characters
sscope: Raw Input Chars/Sec
nsar –y: rawch/s
sar –y: rawch/s

Number of characters read in raw mode from terminal devices. Note that SYAC devices do not distinguish raw input characters from canonical input characters. See Canonical mode.

Modem interrupts

UX/RPM: Modem Interrupts
sscope: Modem Interrupts/Sec
nsar –y: mdmin/s
sar –y: mdmin/s

Number of modem interrupts.

Interrupts

Received

UX/RPM: Receive Interrupts
sscope: Receive Interrupts/Sec
nsar –y rcvin/s
sar –y: rcvin/s

Number of receive interrupts from terminal devices.

Transmitted

UX/RPM: Transmit Interrupts
sscope: Transmit Interrupts/Sec
nsar –y: xmtin/s
sar –y: xmtin/s

Number of transmit interrupts to terminal devices.

Pseudo–Device Unit Count Variables

The PTYCOUNT configuration variable controls the number of pseudo-terminals that get created at system initialization time; the default is **64**. Each pseudo–terminal causes a pair of device entries to appear in the **/dev** directory upon system booting. This variable can cause an impact on database management software (DBMS).

- If the DBMS must search through the **/dev** directory for tty or other device entries, having many pseudo-terminals may increase the search time. Directories are searched linearly, so the search time increases linearly with the number of pseudo-terminals. This is the most likely cause of DBMS performance impact.

- Creating more pseudo-terminals causes a modest amount of additional kernel memory to be allocated. It is possible that the additional kernel memory used by pseudo-terminals could somehow affect DBMS performance.

End of Chapter

6 Networking

Connecting computers and peripherals together with a network is a useful strategy for a large number of user communities. The network enables sharing and better utilization of the available resources and may allow data coherency to be more easily maintained. The advantages of networking must be weighed against the potential disadvantages. I/O across the network will generally take longer than the same operation performed on the local host. The extra delay may be so small as to be imperceptible, or so large as to be intolerable. This chapter attempts to provide you with enough information so that the latter situation is avoided or at least minimized. It also suggests uses of the network that will minimize the load on the local and remote computer systems.

Networks, like most systems, have greater delays as utilization increases. Optimizing network performance can be difficult for a number of reasons such as:

- Connections are made over different media that have different characteristics such as Ethernet, Token Ring, and FDDI.
- You may need special equipment, such as a Network Analyzer, for measuring utilization.
- Network traffic passes from the local host, through intermediate hosts and equipment, to the remote host and then passes back.
- Different vendors' hardware and software may perform in different ways.

It is therefore critical that you take an analytical approach and utilize the tools available to you as fully as possible.

After reading this chapter, you should know the most important and common factors affecting network performance. You will also learn how to perform basic network traffic analysis and how to take steps toward improving performance.

This chapter covers the following topics:

- Network analysis
- Network environment
- Local system environment
- TCP/IP utilities
- NFS
- STREAMS

Here are some of the basic terms that are used in this chapter:

A *network* enables two or more computer systems to communicate. A network includes the hardware and software that make up the interconnections between computer systems and between a computer and its peripheral devices. Network communication between computers takes place over media such as coaxial cable, twisted-pair phone lines, or microwave.

A *host* is a computer system to which a graphics screen or a number of terminals or other smaller computers are connected, and which provides computation access to files and other services. A *local host* is one to which you are directly connected. A *remote host* is one you access through a network.

A *local area network* (LAN) is a network within a small area, such as within a building. A *wide area network* (WAN) is a network of hosts that are far apart. A WAN usually requires connections through public communication facilities (such as the phone company). The *Internet* network is a collection of local networks and gateways that use TCP/IP to function as a wide area network.

Nearly every network system has its layers set up hierarchically. The number of layers and each layer's function vary from network to network. In all networks, each layer provides services to higher layers, without the higher layers knowing the details of how the services are provided. An *interface* consists of the types and forms of messages that each layer uses to communicate with the layer above or below it. A *protocol* specifies how programs on different computers but at the same layer communicate. The set of layers, interfaces, and protocols that govern communication is called the network's architecture.

Introduction to Network Analysis

When you analyze your network, you're not just analyzing a single system. You have a local host, a remote host, and possibly one or more routers or bridges. To find the source of the problem or bottleneck, the best way is through the process of elimination.

If operations that do not use the network take place at a normal rate, but network operations seem slow, suspect a network problem. For instance, if it takes minutes to save a file over the network or you repeatedly receive messages such as "NFS server not responding," that is a sign that there are some problems.

First, a remote system that you are trying to reach through the network must be "up." For DG/UX systems, this means that the system is connected to the network and is at run level 3 or higher. To check on a system, use the command **ping** *remote_machine*, which should display "*remote_machine* is alive". If instead you get the message "no answer from *remote_machine*," and you have reason to believe that the remote system and all intermediate systems are up, and that some of the systems are heavily loaded or the connections are over long distances (such as across the country), you may want to try **ping** with a larger timeout argument. For example, **ping** *remote_machine* 120 would allow 120 seconds for the system to receive a response.

If you get a messages other than "*remote_machine* is alive" you should probably operate on the assumption that the problem is not one of performance; consult the troubleshooting section of *Managing TCP/IP on the DG/UX™ System.*

Once you know that your system and a remote system can, in fact, communicate, continue by using some analysis tools.

Alternatively, you can use the command

traceroute –w 5 –m 24 *remote_machine*

to determine if a host or an intermediate host is having problems. This command's output shows the path that a packet takes to reach a remote host.

Analysis Tools

The best way to minimize network problems is to analyze your computing tasks and properly plan the network layout and equipment that you need. You can contact your Data General Sales representative for information on the network planning and installation services offered by Data General ,which may help you avoid problems.

This section describes tools you can use to analyze your network: a network analyzer, **traceroute**, **tcpdump**, **nfc**, **netstat**, **ttcp**, and SNMP. The **nfsstat** command, which analyzes NFS usage, is described in the section "NFS" later in this chapter.

Network Analyzer

A Network Analyzer, sometimes called a LANalyzer, is a specialized piece of equipment that can monitor, capture, and generate network traffic. Depending on the network media, you will require a specific network analyzer (for example, FDDI) or a specific network analyzer board (for example, Ethernet and Token Ring). They can aid in characterizing network performance in ways that general purpose computers cannot. Though not inexpensive, you may want to investigate leasing or buying a Network Analyzer if your network is large enough or down time and poor performance would be serious problems.

If you suspect that the network is shorted or open, a network analyzer may be used to verify that condition. A network analyzer sends signals down the network and waits for them to bounce back (a technique known as Time Delay Reflectometry); by measuring the interval between the initial signal and its reflection, the analyzer can tell you approximately how far away the problem is.

Network analyzers are also capable of determining the percentage of the media's theoretical maximum bandwidth being utilized. Determining network utilization is an important part of optimizing network performance. Checking for fragmented packets, misaligned packets, monitoring communication between host pairs, and interpreting the network protocol information are among the other useful features that network analyzers provide.

If you have a service contract, are able to capture an instance of the problem with the network analyzer, and need assistance in interpreting the data or correcting the problem, send your support center the output on a diskette (make sure the support center has hardware that can read your diskette) and also send the output on hardcopy.

traceroute

Use **traceroute** to display the route that packets take to reach a network host. The **traceroute** command launches probe packets and then listens for replies from a gateway; it continues until it either reaches the host or reaches the maximum number of hops (either specified or the default (30)). When viewing this command's output, you should consider these questions:

- Is the path optimal? The number of routers that the packet must take should be minimized.
- Are there alternative routes that may be faster?

For more information about this command, see the **traceroute**(1M) man page.

tcpdump

The Berkeley Packet Filter (BPF) is a kernel interface to allow raw data to be read from the network. By default, this interface is available on the DG/UX system. If the entry **/dev/bpf0** does not exist, see the **bpf**(4) **man page.**

The command **tcpdump** provides a user interface to BPF. This command allows any machine on the network to function as an Ethernet, FDDI, or Token Ring sniffer. You can choose to decode a subset of packets or store them for later decoding. Note that you must be superuser to use this command.

By default, all packets are dumped. You can choose for **tcpdump** to filter packets by giving the command a filter expression. The filter expression (a Boolean expression) can consist of a protocol name (such as ip, tcp, or telnet), a command (such as **host mach1**), or an arbitrary comparison of the contents of each packet.

When **tcpdump** terminates, the number of packets dropped by the kernel is printed. The following is a list of possible solutions that may prevent dropped packets and possibly improve system performance:

- If you are only interested in packets sent to or from the local host, use **tcpdump**'s **–p** option to prevent the interface from being put into "promiscuous" mode (promiscuous mode makes **tcpdump** see all packets). This will improve system performance as well as reduce or eliminate the number of dropped packets.
- Use **tcpdump**'s **–w** option to write packets to a capture file. You can then use the **–r** option to decode and print the packets from the capture file later. Decoding and printing packets can be time consuming and cause other packets to be dropped.
- Use the filter expression to filter out as many packets as possible. For example, the expression 'tcp' will filter out more packets than 'ip' because TCP packets are a subset of IP packets.
- Increase the priority of **tcpdump** (see Chapter 2).

In this example, **tcpdump** writes the information about two NFS packets sent to or from the local host to the file **/tmp/dumpfile**.

```
# tcpdump -c 2 -p -w /tmp/dumpfile 'nfs' ↵
tcpdump: listening on dgen0
42 packets received by filter
0 packets dropped by kernel
```

Next, the information is decoded and printed:

```
# tcpdump -r /tmp/dumpfile ↵
22:25:16.37
        ip: saturn->bojangles
        udp: 1011->nfs
        nfs: e310d: 112 getattr fh
383a303a.31623a66.663a303a.3738000a
22:25:16.38
        ip: bojangles->saturn
        udp: nfs->1011
        nfs: e310d: reply ok 96
```

For more information, see the **tcpdump**(1M) man page.

nfc

The network file converter (**nfc**) command converts files containing raw network data from one format to another. The current file conversions that are supported are:

- Converting from a **tcpdump** capture file to a Network General Sniffer save file.
- Converting from a Network General Sniffer file to a **tcpdump** file.

This command makes it possible to capture network traffic at one site using **tcpdump** and to later analyze that information at another site using the Network General Sniffer. See the **nfc**(1M) man page for more information.

netstat

The **netstat** command provides network status information such as per–interface statistics, per–protocol statistics, per–socket statistics, and routing table entries. To monitor the traffic on interfaces, use **netstat** followed by an interval argument (the number of seconds you want **netstat** to pause before updating the display). This command shows the number of incoming packets, outgoing packets, incoming errors, outgoing errors, and collisions (meaningful on Ethernet only).

```
% netstat  10 ↵
    input    (cien1)     output          input    (Total)     output
packets errs  packets errs  colls packets errs   packets errs   colls
7175938 0     6995810 0     26868 11753764 4     10033847 1     129987
64      0     24      0     0     67       0     26       0     0
3       0     3       0     0     13       0     4        0     0
66      0     64      0     0     88       0     77       0     1
53      0     50      0     2     55       0     50       0     2
6       0     6       0     0     15       0     7        0     0
13      0     13      0     0     26       0     15       0     0
10      0     11      0     0     14       0     13       0     0
```

This display shows two groups of columns: one for the first interface on the interface list (cien1 here), and one for all interfaces. The first line of information contains a summary of activity since the system was last rebooted. Subsequent lines of output show values accumulated over the preceding interval. When you do not want to see any more statistics, press Ctrl–C.

Issue the **netstat –s** command to display statistics for the ip, icmp, tcp and udp protocols. Reported parameters include input datagrams dropped because of flow control, bad header checksums, packets with incorrect lengths, fragments, retransmitted data packets, and window probe packets.

```
% netstat -s | more ↵
...
udp:
        0 bad header checksums
        0 incomplete headers
        0 bad data length fields
        8152926 datagrams received
        78198 datagrams received for non-existent port
        7064176 transmit datagrams requested
        74 input datagrams dropped because of flow control
```

If a system cannot store input datagrams because it cannot perform the operation quickly enough or because it has run out of buffer space in which to store the data, that information is displayed "*xxx* input datagrams dropped because of flow control." Make note of the number *xxx*. The ratio of dropped packets to the total number of datagrams received should be low (0.05%). A larger percentage than that indicates overload.

You can also issue **netstat** with no options:

```
% netstat ↵
Active connections
Proto Recv-Q Send-Q  Local Address        Foreign Address     (state)
tcp        0      0  host-1588            xyz-telnet           ESTABLISHED
tcp        0      0  host-1038            def-1059             ESTABLISHED
tcp        0     94  host-login           ghi-1023             ESTABLISHED
tcp        0      0  host-login           mno-1023             ESTABLISHED
```

The report shows one line for every currently active connection using the Internet protocol family. The important statistic is the number of bytes in the send queue. This number should be 0 for most of the connections that **netstat** lists. However, **ftp** transfers can cause large numbers to appear in the send queue—this does not necessarily mean that your network is congested. Conversely, NFS endpoints always report 0 bytes in the send queue. Issue the **netstat** command repeatedly; if the send queue is consistently large for many of the connections, the network is congested.

Issue the command **netstat –i** and look at the Ierrs and Oerrs columns. Sample output follows:

```
% netstat -i ↵
Name  Mtu   Network       Address       Ipkts   Ierrs Opkts   Oerrs Collis
hken1 1500  one-lan       asystem-alt  7111958 0     6928265 0     26753
hken0 1500  two-lan       asystem      4519759 4     2999458 1     103033
loop0 4136  loopback-net  localhost    7910    0     7910    0     0
```

Input errors includes all errors that occurred as a result of receiving packets from the network since the system was last rebooted. If an input error occurs, the network interface just discards the packet and trusts the sender to replace it. A large number of input errors usually means that there is faulty hardware on the network. Faulty hardware can mean anything from another computer system that is generating packets improperly to a bad connector or terminator.

A large number of output errors means that your system's network interface is faulty. The problem may be in your system's network controller, the LAN drop, or virtually anything between your CPU and the main LAN cable. Output errors are not caused by other systems. If you see output errors, the problem is local.

In normal operation, the acceptable number of input or output errors is extremely low: 0.025% of the total number of input or output packets. You can expect to see higher numbers if you are plugging or unplugging cables, or if a power failure or some other situation causes all of your systems to boot simultaneously. In these situations, a large number of output errors is normal.

SNMP

SNMP (Simple Network Management Protocol) can be used in place of **netstat** to collect information from various hosts on the network. You can use the following commands to query an SNMP agent for statistics:

- **snmpgetone**
- **snmpgetnext**
- **snmpgetmany**
- **snmpgettab**

It is possible to collect statistics from any host from a central location.

You can use the **snmpgettab** command to get the interface table from a host, as shown in this abbreviated example (usually, there are multiple index numbers—this only shows index 1):

```
% snmpgettab hostname public ifTable↲
ifIndex.1 = 1
ifDescr.1 = loop0
ifType.1 = 24
ifMtu.1 = 4136
ifSpeed.1 = 0
ifPhysAddress.1 =
ifAdminStatus.1 = 1
ifOperStatus.1 = 1
ifLastChange.1 = 0
ifInOctets.1 = 351960231
ifInUcastPkts.1 = 772424
ifInNUcastPkts.1 = 0
ifInDiscards.1 = 0
ifInErrors.1 = 0
ifInUnknownProtos.1 = 0
ifOutOctets.1 = 360540
ifOutUcastPkts.1 = 67394
ifOutNUcastPkts.1 = 0
ifOutDiscards.1 = 0
ifOutErrors.1 = 0
ifOutQLen.1 = 0
ifSpecific.1 = nullSpecific
```

This section describes objects of particular interest from the above example. You can view descriptions of other objects in **/usr/etc/snmp/rfc1213.mib**. This file is compressed (denoted by the **.Z** extension); to restore the file with **uncompress**, see the **compress**(1) man page.

ifInOctets – The total number of octets received on the interface, including framing characters.

ifInUcastPkts – The number of subnetwork-unicast packets delivered to a higher-layer protocol.

ifInNUcastPkts – The number of non-unicast (such as subnetwork-broadcast or subnetwork-multicast) packets delivered to a higher-layer protocol.

ifInDiscards – The number of inbound packets that were chosen to be discarded (even though no errors had been detected) to prevent their being deliverable to a higher-layer protocol. One possible reason for discarding such a packet could be to free up buffer space.

ifInErrors – The number of inbound packets that contained errors preventing them from being deliverable to a higher-layer protocol.

ifInUnknownProtos – The number of packets received via the interface that were discarded because of an unknown or unsupported protocol.

You can also use SNMP to check collision rates. SNMP provides more thorough information than **netstat** about Ethernet devices; in particular, it can differentiate between single, multiple, and excessive collisions.

First, get the names of the interfaces in order to get their corresponding index numbers.

```
% snmpgetmany hostname public ifDescr ↵
ifDescr.1 = loop0
ifDescr.2 = pefn0
ifDescr.3 = cien1
ifDescr.4 = cien0
```

There are two Ethernet interfaces, cien1 and cien0, which have index numbers of 3 and 4, respectively.

The next example shows how to display the Ethernet statistics table. There are two groups of statistics, one for interface 3 and the other for interface 4.

```
% snmpgettab hostname public dot3StatsTable ↵
dot3StatsIndex.3 = 3
dot3StatsAlignmentErrors.3 = 10
dot3StatsFCSErrors.3 = 2
dot3StatsSingleCollisionFrames.3 = 356641
dot3StatsMultipleCollisionFrames.3 = 409168
dot3StatsSQETestErrors.3 = 5572824
dot3StatsDeferredTransmissions.3 = 7
dot3StatsLateCollisions.3 = 7
dot3StatsExcessiveCollisions.3 = 176
dot3StatsInternalMacTransmitErrors.3 = 0
dot3StatsCarrierSenseErrors.3 = 1
dot3StatsFrameTooLongs.3 = 5
dot3StatsInternalMacReceiveErrors.3 = 0

dot3StatsIndex.4 = 4
dot3StatsAlignmentErrors.4 = 9
dot3StatsFCSErrors.4 = 3
dot3StatsSingleCollisionFrames.4 = 340505
dot3StatsMultipleCollisionFrames.4 = 296619
dot3StatsSQETestErrors.4 = 13287186
dot3StatsDeferredTransmissions.4 = 7
dot3StatsLateCollisions.4 = 7
dot3StatsExcessiveCollisions.4 = 1
dot3StatsInternalMacTransmitErrors.4 = 0
dot3StatsCarrierSenseErrors.4 = 111
dot3StatsFrameTooLongs.4 = 0
dot3StatsInternalMacReceiveErrors.4 = 0
```

These statistics are of interest:

dot3StatsSingleCollisionFrames – A count of successfully transmitted frames on a particular interface for which transmission is inhibited by exactly one collision.

dot3StatsMultipleCollisionFrames – A count of successfully transmitted frames on a particular interface for which transmission is inhibited by more than one collision.

dot3StatsExcessiveCollisions – A count of frames for which transmission on a particular interface fails due to excessive collisions.

When excessive collisions are present, it means that the Ethernet interface attempted to transmit a packet sixteen times, and each attempt failed due to a collision. Excessive collisions result in dropped packets, resulting in performance degradation.

See **/usr/etc/snmp/rfc1398.mib** for a description of other objects in the statistics table. This file is compressed (denoted by the **.Z** extension); to restore the file with **uncompress**, see the **compress**(1) man page.

ttcp

ttcp is a traffic generator that can be used for testing end-to-end throughput. Cooperating processes are started on two hosts. They open a tcp connection and transfer a high volume of data. Then, delay and throughput are calculated. Contact your Data General Sales representative for information on how to obtain **ttcp**.

Network Environment

This section reviews some of the choices that you must make regarding network installation and configuration that affect performance. We then discuss some performance problems that can arise in a networked environment. It is beyond the scope of this manual to offer more than an introduction to some of the resources available to you.

One of the first choices you must make is which type or types of media to use. The relative performance of different media must be weighed against the relative cost of the equipment and installation. You may also be presented with choices regarding whether to have a general purpose computer perform a particular networking task or whether specialized hardware would better suit your needs. The network must provide enough bandwidth to satisfy the needs of the network's users. If the network does not have enough bandwidth, the amount of time needed to transfer data between any two points gets excessively long.

While the next sections on different media note the maximum speed of each medium, it is not valid to assume that the time it takes to perform the same task on networks of different media will be proportional to their media speeds. For example, characters will probably not be echoed ten times faster for a telnet session over a FDDI ring versus one over Ethernet.

The following sections go into more detail about the network environment.

Ethernet

Ethernet was developed in the late 1970's at Xerox PARC and is comparatively simple and inexpensive. An Ethernet network is a type of local area network that consists of cable and interface hardware that connects hosts. The Ethernet transmission speed is 10 megabits per second (10 Mbps).

Before transmitting a message, a system waits until it hears no other system transmitting. If it senses no carrier from another system, it will then send the message. Since there is the possibility that another system will decide to transmit at roughly the same time, the sending system monitors the network to hear if its own message appears on the network ungarbled. If it detects a collision with another message, it intentionally sends a jam signal to ensure propagation of the collision throughout the Ethernet to all other transmitting systems. The colliding systems each back off a random length of time before attempting transmission again.

To see if your network is consistently overcrowded, use SNMP to retrieve the dot3StatsTable (as described previously) and examine the **dot3StatsExcessiveCollisions** statistic. Collisions are normal events and do not indicate hardware problems. However, when a network is overloaded, the number of excessive collisions increases and packets take longer to get through the network. To solve this problem, you can try to rearrange the network and applications that use the network in a way that reduces traffic.

Token Ring and FDDI

Token ring networks were first developed in the early 1980's by IBM. Systems are connected into a ring, but cabling is simplified by using a single cable for each system's interface that goes to a passive concentrator, or multiple access unit (MAU). A transmission speed of 4 Mbps was originally used for token ring, but more recent versions use 16 Mbps. Data General's VME Token Ring Controller (VTRC) can be jumpered to operate in rings of either speed.

Only one system on the ring is allowed to transmit at a time; its message is retransmitted by each system around the ring. When no system on the ring has any data to transmit, a short 3-byte message known as a free token circulates around the ring. When a station decides to transmit a packet, it waits for the free token and then transmits its packet instead of the free token. When its message comes back around the ring, the originating system reinserts the free token onto the ring. In addition to the simple token passing algorithm, token ring uses a priority scheme implemented with a 3-bit priority field and a 3-bit reservation priority field that are present in every message.

The time required to repeat the frame at a station is called latency. The ring latency is the sum of the latency at each station plus the propagation delay around the ring. As the number of stations in the ring and the size of the ring increase, so will the ring latency.

Fiber Distributed Data Interface (FDDI) is a newer technology than Ethernet and token ring, uses fiber optic cable as the transmission media, and has a 100 Mbps transmission speed. Like token ring, it uses a ring topology, uses token passing to control access and is subject to the same type of ring latency. FDDI networks usually have a dual, counter-rotating ring topology for reasons of greater fault tolerance, fault isolation, and higher availability. Should the primary ring fail, the other ring ensures that the network stays in operation.

FDDI is designed to handle large amounts of data and is often used as a backbone LAN which connects other LANs.

FDDI performance is affected by the distribution of I/O between IOCs (I/O controllers). If more throughput is required, you might want more than one IOC.

Subnetting

Regardless of which types of media you use, you may want to consider whether the use of subnets may improve network performance. Subnets are an extension of the Internet addressing scheme that allows a site to use a single Internet address portion of its host address field as a subnet field. Outside the site, routing divides the destination address into an Internet portion and a local portion. Routers and hosts inside a site that uses subnets interpret the local portion of the address by dividing it into a physical network portion and a host portion. Thus, a site can present a single local network number to the world, but still maintain distinct physical networks and routing internally.

When a large number of systems must be connected to the network, dividing the systems among several subnets has several performance advantageous over having them all connected to the same LAN. First, since there are fewer systems on the subnet, the time it takes for a system to gain access to the LAN is reduced because of less contention. In the case of Ethernet, once a system begins transmitting there is a lower probability of a collision. In the case of token ring and FDDI, there are fewer systems introducing the delay needed to repeat the message. Finally, physically shorter LANs have shorter propagation delays.

In planning which systems should be on which subnet, the main principle to follow is that a system should be on the same subnet as the systems it shares data with routinely. The best example of this is operating system servers and their diskless clients. The division often falls along lines of different departments having different subnets. Each time systems on different subnets interact, traffic is introduced on at least two subnets, any intermediate routers introduce some delay, and the load on each router is increased. Of course the total cessation of information flowing between subnets and departments is probably not desirable because of possible consequences more serious than a little network congestion.

One of the goals of planning subnets is to minimize the number of hops that any two systems must make to reach each other. Taken to an extreme, this would mean all systems would make just one hop to their destinations (i.e., just one LAN); this goal must be balanced against having too much traffic on any subnetwork. One solution is for each subnet to have a router that connects it to a backbone LAN populated with only routers. Thus each system is 3 hops, at most, from the destination.

Routing Considerations

Any time traffic needs to flow between different subnets or networks, give special attention to the routing of network traffic in order to minimize delays and the load on intermediate subnets and routers. Improperly administered routing tables can cause packets destined for a system across a building to follow a path across the country and then back again to reach their destination, though somewhat later than the optimum.

TCP/IP for AViiON supports two dynamic routing programs, **routed**(1M) and **gated**(1M). **gated** is recommended because it supports multiple routing protocols, implements superior algorithms for route path fault detection, and provides more flexibility. For more information about these programs, see *Managing TCP/IP on the DG/UX™ System.*

Use **traceroute** to see the route a packet takes between two hosts.

```
% traceroute toe.cs.berkeley.edu ↵
traceroute to toe.cs.berkeley.edu (128.32.149.117), 30 hops max, 40 byte
packets
 1  dgrtpgw2-2 (128.11.2.39)  10 ms  0 ms  10 ms
 2  /cisco (128.11.1.204)  20 ms  0 ms  10 ms
 3  dgrtpgw3-1 (128.11.250.8)  50 ms  10 ms  0 ms
 4  wing-alt3.us.dg.com (128.12.123.2)  20 ms  20 ms  30 ms
 5  sprint-gw.us.dg.com (128.12.137.29)  30 ms  30 ms  40 ms
 6  sl-dc-1-s2-384k.sprintlink.net (144.228.11.97)  40 ms  40 ms  40 ms
 7  icm-dc-2-f0-100m.icp.net (144.228.1.36)  40 ms  40 ms  50 ms
 8  icm-fix-e-h0-t3.icp.net (192.157.65.122)  50 ms  50 ms  40 ms
 9  192.203.229.246 (192.203.229.246)  50 ms  40 ms  40 ms
10  t3-2.washington-dc-cnss58.t3.ans.net (140.222.58.3)  80 ms  50 ms
40 ms
11  t3-3.washington-dc-cnss56.t3.ans.net (140.222.56.4)  40 ms  50 ms
50 ms
12  t3-0.new-york-cnss32.t3.ans.net (140.222.32.1)  50 ms  140 ms  50 ms
13  t3-1.cleveland-cnss40.t3.ans.net (140.222.40.2)  60 ms  60 ms  60 ms
14  t3-2.chicago-cnss24.t3.ans.net (140.222.24.3)  70 ms  70 ms  70 ms
15  t3-1.san-francisco-cnss8.t3.ans.net (140.222.8.2)  110 ms  110 ms
110 ms
16  mf-0.san-francisco-cnss9.t3.ans.net (140.222.8.193)  120 ms  110 ms
110 ms
17  t3-0.enss128.t3.ans.net (140.222.128.1)  110 ms  110 ms  130 ms
18  su-a.barrnet.net (192.31.48.200)  110 ms  110 ms  130 ms
19  ucb.barrnet.net (131.119.5.2)  120 ms  150 ms  110 ms
20  inr-108-dmz.berkeley.edu (192.31.161.22)  120 ms  110 ms  120 ms
21  inr-111.berkeley.edu (128.32.120.111)  130 ms  130 ms  130 ms
22  toe.cs.berkeley.edu (128.32.149.117)  110 ms  130 ms  130 ms
```

Use **netstat -r** to check the routing tables.

```
% netstat -r ↵
Destination      Gateway       Flags     Refcnt Use       Interface
localhost        localhost     UH        39     2671599   loop0
default          dgrtpgw2-2    UG        0      0         hken0
doc-lan          dgrtpgw2-2    UG        1      33349     hken0
      .
      .
      .
```

The flags field shows the state of the route ("U" if up), whether the route is to a gateway ("G"), or whether the route is to a particular host ("H").

The system will automatically set up routes for each local interface listed in **/etc/tcpip.params**. Additional routes may be managed through **sysadm** using one of two methods. You may add individual static routes using TCP/IP –> Routes or you may start a routing Daemon using TCP/IP –> Daemons.

Gateways and Data Transfers

Gateways between networks are an additional source of errors. Use the **netstat –s** command to find out if gateways are causing data corruption. The command output includes **bad checksums** fields under the **udp**, **tcp**, **icmp**, and **ip** headings; these fields indicate packets that were corrupted while flowing through a network gateway.

When your system is not operating on Ethernet media, **checksum** errors may be the result of media errors; there may not be a problem on the gateway. For example, a SLIP link may introduce **checksum** errors.

```
%  netstat -s ↵
udp:
        0 bad header checksums
        8155017 datagrams received
...
tcp:
        703826 packets received
        0 discarded for bad checksums
...
icmp:
        0 bad checksums
...
ip:
        11460668 total packets
        6 bad header checksums
```

Gateway corruption should be an extremely small percentage (hundredths of a percent or less) of the total number of packets received.

If your system is using only the Ethernet network interface, the error percentage should be less than 2×10^{-16}.

When a system receives a bad packet from the network, it detects the error and usually drops the bad packet. The sender does not receive the expected acknowledgement and must retransmit the packet. When a large percentage of network packets are damaged en route, performance declines for two reasons:

- Timeout delays (while the sender waits for the receiver to acknowledge the packet).
- Increased load due to retransmissions.

Your system network must be able to transfer data correctly. If it cannot, you must isolate the faulty equipment (i.e., a network interface, transceiver, connector, or cable). Carefully follow the guidelines for LAN hardware installation, including such things as cable length and cable grounding. Also, make sure that cables are attached securely to their connectors.

Network Connections

In some cases, the number of packets dropped by the network can be greatly reduced by moving server machines from a multidrop box onto a direct transceiver connection. The **netstat** command can help you identify networking problems. As discussed in the previous section, use **netstat –s** to look for the number of tcp retransmissions, checksum errors, and timeouts. Using **netstat –i**, view the Ierrs and Oerrs columns.

Local System Environment

The local environment of each system through which network traffic travels has some effect on the time it takes to perform an operation over the network. For example, when transferring a file from the local system to a remote system, all of these play a part:

- the time it takes to read data from the local disk,
- the time it takes the local LAN interface to transmit the data,
- the time it takes for the destination to receive the data,
- and the time it takes the destination to write the data to its disk.

If routers are involved there will also be time required for each one to write the data to buffers and then read from the buffers in addition to receiving and transmitting the packets on the LAN. This section examines the impact of different hardware components on tasks performed over network.

CPU Performance

The availability of CPU cycles on the local system, remote destination, and any intermediate computers that perform routing affect performance. If a system's CPU is heavily loaded, the instructions to read and write network data may be performed slowly. In the case of receiving data from the network, if a write to a buffer cannot be performed quickly enough, the data is dropped. This event introduces a delay for the sending system to note the lack of an acknowledgement during a specified period (a timeout) and for the sender to retransmit the data.

For AViiON systems you can refer to the CPU performance chapter of this manual to find methods of analyzing and improving CPU performance. For other vendors' systems, consult that vendor's reference material.

Disk Performance

When network tasks involve reading and writing large amounts of data to and from disk, the disk performance can play a significant part in the completion time for the task.

For AViiON systems you can refer to the disk performance chapter of this manual to find methods of analyzing and improving disk performance. For other vendors' systems, consult that vendor's reference material.

Diskless Client Performance

If you have users on diskless workstations, bear this in mind: diskless workstations use the network for all paging and swapping activity. If these workstations do not have enough local memory to minimize paging and swapping, they can easily consume a lot of network bandwidth. Workstation users will notice sharply degraded performance whenever their workstation is paging.

If you suspect that a diskless workstation is paging too much, you should consider adding more memory to the workstation or offloading some of the processing from the diskless system to the server. For example, large X Window System applications may perform better when running on the server and displaying to the workstation than when running on the workstation.

Use subnets to reduce LAN traffic. Diskless workstations should always be on the same subnetwork as their server.

Since diskless workstations remote mount their file systems, you may wish to see the section "NFS" later in this chapter.

TCP/IP and Its Utilities

TCP/IP stands for Transmission Control Protocol and Internet Protocol. The Internet Protocol is a kernel-level protocol that defines unreliable, connectionless delivery of datagrams. An IP datagram contains the addresses of its source and destination, and the data transmitted. Connectionless service means that the protocol treats each datagram as a separate entity; the protocol can deliver packets out of sequence, or can drop packets. IP defines the exact format of data as it travels through a network, but delivery of data is not guaranteed.

Transmission Control Protocol is a kernel-level protocol that defines reliable, end-to-end delivery of datagrams. TCP is connection-based because it establishes a connection between communicating hosts before transmitting data. TCP allows a process on one host to send data to a process on another through a byte stream. TCP uses the Internet Protocol to transmit information along an Internet network and also provides flow control, which ensures that either side of the connection will send data at a rate that is acceptable to the other side.

An understanding of your applications that run over TCP/IP may help you to optimize performance. A discussion of telnet, rlogin, and ftp follows.

Telnet and rlogin

Telnet and rlogin are user-level protocols accessed through the **telnet** and **rlogin** commands, respectively. With both, you can interact with a remote host as if your local host's terminal were directly connected to the remote host.

If you use either utility extensively, you should understand how they work and how their performance can be affected. Typically, input and output for telnet and rlogin is sent a character at a time in separate packets across the network. If telnet performance is very poor and other means of improving performance have not helped or do not seem appropriate, some vendors (including Data General) enable you to specify the use of line-at-a-time echoing. However, using line-at-a-time echoing in a multi-vendor heterogeneous environment may result in unexpected side effects.

One definition of latency is the amount of time needed to send a single byte of data between two systems, which is sometimes referred to as per–packet overhead. Given that, you can appreciate that latency is important to telnet and rlogin performance. Contributing factors include software efficiency and CPU speed. Benchmarks that estimate the number of round trips per second may help estimate how telnet and rlogin will perform.

To improve telnet's software efficiency, Data General has moved some of the server functionality into kernel space. Therefore, if both telnet and rlogin are available on your client system, choosing to connect to a Data General server with telnet (rather than rlogin) produces less load on the server.

Finally, you may want to consider alternatives to using the host-based telnet–LAN controller combination that might relieve CPU load (for example, the VTC/256 controller). You can obtain more information on the VTC/256 from the **syac**(7) manual page and from your Data General Sales Representative.

FTP

FTP is a user-level protocol accessed through the **ftp** command. You can transfer files from one host to another with FTP.

If you use FTP extensively, you should understand how it works and how its performance can be affected. When ftp is used to transfer a file, generally the data must be read from disk and transmitted by the LAN controller on the source system, received, buffered, and transmitted by any intermediate routers, and then received and written to disk by the destination system. See the disk performance chapter for methods of evaluating and improving disk performance.

If large numbers of small files are transferred, latency may have significant affects on FTP performance. As transfers tend toward larger and larger files, the network *throughput* becomes increasingly important. Throughput is the amount of data that can be delivered over the network in a given amount of time. Contributing factors are the DMA capability of a communication controller and the efficiency of data-size-dependent software. Thus the per-byte overhead multiplied by each byte of a large file can become significant in an FTP transfer.

Some workarounds are available for reducing the time and impact of transferring data with FTP. Users that routinely transfer files over networks that span many hops and long distances find that compressing files with utilities such as **compress, gzip,** and possibly **pack** reduces the transfer time. Shifting transfers to times of low network and system loads is another alternative that you can implement automatically using the **cron** or **bftp** utilities.

A transfer of a large file may start out with no problems (for example, you can connect to the remote system's ftp daemon), but may tend to fail repeatedly prior to completion of the transfer, thus increasing the network traffic load because of your retries. You may want to consider using the ftp restart capability, which restarts the last transfer where it was aborted.

NFS

NFS is a service that allows many users to share file systems over a network.

When I/O is being done to an NFS mounted file system, there are several additional factors that need consideration. By default, NFS does all its I/O in 8-Kbyte blocks, regardless of the file's data element size. This can be overridden when mounting the file system, but experience shows that it takes no longer to transfer an 8-Kbyte block over a LAN than it takes to transfer a smaller block. The option of specifying NFS read and write block size was supplied to prevent swamping a client, server, slow link, slow/overloaded gateway, or slow/overloaded bridge with data.

To even out the network traffic, it may help to reduce the value of MAXBUFAGE on the client and decrease the value of FREEINODE on the server. See "File System Configuration Variables" in Chapter 4 for a description of MAXBUFAGE and FREEINODE.

Although NFS can do some operations asynchronously, other operations are synchronous and/or unbuffered. For example, NFS will read ahead and write behind data as well as buffer the data; thus, reads and writes (particularly sequential ones) can be relatively inexpensive.

On the other hand, closes, some lookups, and file status operations are synchronous. Closes will not be completed until all the data has been flushed to disk.

You can improve the performance of NFS exported file systems by using disk caching. Set up the cache to cache only file system metadata for the best performance. See the disk caching section of Chapter 4 for more details.

TCP/IP performance can have a big effect on NFS performance. Note that the network will retry operations several times (indicated by the console message "server not responding") before an error is returned to NFS, but if the network is frequently returning errors to NFS, the NFS backoff/retry can result in cumulative delays that are quite expensive. To minimize these delays, it is important to ensure your network is set up correctly.

Analyzing NFS Usage

The following sections tell you how to analyze your system's NFS usage.

Using nfsstat

You can obtain a summary of NFS operations and NFS visible network errors with the **nfsstat** command:

```
% nfsstat ↵
Server rpc:
calls      badcalls   nullrecv   badlen     xdrcall
4606507    0          0          0          0

Server nfs:
calls      badcalls
4606495    0
null       getattr    setattr    root       lookup     readlink   read
0  0%      2558832 55% 24784  0%  0  0%      505736 10% 5648  0%   364845  7%
wrcache    write      create     remove     rename     link       symlink
0  0%      1045531 22% 11957  0%  23750  0%  4617  0%   946  0%    2329  0%
mkdir      rmdir      readdir    fsstat
31  0%     330  0%    55475  1%  1684  0%

Client rpc:
calls      badcalls   retrans    badxid     timeout    wait       newcred
69067      0          25797      16         25797      0          0

Client nfs:
calls      badcalls   nclget     nclsleep
69067      4          77841      0
null       getattr    setattr    root       lookup     readlink   read
0  0%      33528 48%  19  0%     0  0%      20633 29%  328  0%    8465 12%
wrcache    write      create     remove     rename     link       symlink
0  0%      28  0%     7  0%      2  0%      1  0%      0  0%      0  0%
mkdir      rmdir      readdir    fsstat
0  0%      0  0%      5970  8%   86  0%
```

These values are only reset at boot time.

After issuing the **nfsstat** command, check the **retrans** field under "Client rpc"; this indicates the number of packets that this host retransmitted as an RPC client—the number of retransmissions it made while reading or writing an NFS file. If this field is greater than 5% of the total number of client NFS calls, you may have a problem. Compare the **retrans** number to the value of **badxid**, which reports the number of times a server's reply did not match the client's RPC call. If the numbers are about equal, one or more of the network's NFS servers is having trouble keeping up with the client's demands. If the **retrans** number is high, but the **badxid** number is relatively low (as shown in the example above), the problem is the network itself—the network is either slow or suffering from data corruption.

UX/RPM and AV SysScope Data

With UX/RPM and AV SysScope™, you can observe various current NFS usage statistics.

STREAMS

DG/UX TCP/IP is implemented using the STREAMS mechanism; in some cases, your network usage may call for tuning STREAMS variables. A Stream is a full duplex processing and data transfer path between a driver in the kernel space and a process in the user space. STREAMS provides a standard framework for network protocol stack implementation.

The DG/UX system dynamically allocates STREAMS data structures. If memory were allocated statically at system initialization, it would not be generally available and performance could suffer. To ensure that STREAMS does not use an overly large amount of the system memory, you can set the PERCENTSTR STREAMS configuration variable (which specifies the maximum percentage of system memory that can be allocated to STREAMS); it is, by default, 20% of the system's total memory. Although this should be sufficient to handle heavy network loads, you may want to experiment with larger values if your system often uses the X Window System and other network services. However, this parameter will only affect performance if you are running out of STREAMS memory.

The stream head module, the driver (usually), and each stream module in between the stream head and driver have a queue pair (a read queue and a write queue) allocated. A queue pair is used whenever a module is pushed on the stream such as when protocol stacks are built and when network connections are made. A queue pair is freed whenever a module is popped such as when a network connection is broken and a network protocol stack is torn down.

These are the network STREAMS requirements:

- Each socket requires 2 queue pairs.
- Each TLI (Transport Layer Interface) requires 3 queue pairs.
- Each telnet server connection requires 14 queue pairs, 2 mux links, and 1 pseudo tty.
- Each rlogin connection requires 9 queue pairs and 1 pseudo tty.

The NQUEUE STREAMS configuration variable specifies the maximum number of STREAMS queue pairs. You may want to investigate the need to adjust it upward if a large number of network connections must be made to or from your system on a routine basis and users are being denied access to network services with messages that suggest that a module could not be pushed.

STREAMS Configuration Variables

The STREAMS configuration variables are associated with STREAMS processing. These variables are also listed in **/usr/etc/master.d/dgux**. They set the STREAMS parameters shown in the following list.

PERCENTSTR
Specifies the maximum percentage of physical system memory (after initialization) that can be used for STREAMS buffers. The default is **20**.

NQUEUE
Specifies the maximum number of STREAMS queue pairs (STREAMS plus instances of STREAMS modules) that may exist at any one time on the system. A minimal stream contains two queue pairs: one for the stream head and one for the driver. Each instance of a module on a stream requires an additional queue pair. The default is **2048**. As a guideline, allow at least 5 STREAMS queue pairs for each tty used, 8 for each pseudo tty used, 10 for each concurrent rlogin session, 15 for each concurrent telnet session (includes pseudo ttys), and 2 for each socket or pipe used; round the total up to a power of 2. If that total is greater than the default value, increase NQUEUE.

NSTRPUSH
Specifies the maximum number of STREAMS modules that may be pushed on any one stream. This is used to prevent an errant user process from consuming all the available queue pairs on a single STREAMS module. The default is **9**.

STRMSGSZ
Specifies the maximum number of bytes allowed in the message portion of a stream. A module maximum packet size of **INFPSZ** (defined in **/usr/include/sys/stream.h**) defaults the maximum packet size to this value. The default is **0**, which means that there is no default maximum message length. In this case, the size of the message is only restricted by what the queues on the stream will accept.

STRMCTLSZ
Specifies the maximum number of bytes allowed in the control portion of a stream. The control part of a message created with **putmsg** is not subject to the constraints of the minimum or maximum packet size, so this value is the only way of providing a limit for the control part of a message. The default is **1024**.

NMUXLINK

Specifies the maximum number active multiplexors that may exist at any one time on the system. The default is **1024**. As a guideline, allow at least 2 multiplexors per pseudo tty used, 2 per communication stack (TCP/IP, UDP, etc.), and 2 per network interface (loop, dgen, etc.); round the total up to a power of 2. If that total is greater than the default value, increase NMUXLINK.

NLOG

Specifies the maximum number of log devices available. The default is **16**.

NPIPE

Specifies the maximum number of STREAMS pipe devices available. The default is **64**. Note that this does not affect IPCs created by using **pipe**(). Large NPIPE values may adversely affect **ttyname** lookup times and increase CPU usage.

BSIZE

Specifies the maximum number of log messages allowed to be enqueued on a log driver's read queue. The default is **20**. This prevents being swamped by log messages during peaks.

NSTREVENT

Specifies the maximum number of signal delivery requests (established with the **I_SETSIG** ioctl command) allowed to be enqueued on the signal event list. The default is **2048**.

End of Chapter

7 Common Sense Performance Tips

This chapter lists some things that you can do to improve performance. To ensure maximum system performance, you should first monitor system performance on a daily basis—establish your criteria for good system performance and note your system thresholds. Then you should check for:

- Jobs running during peak hours that could just as well run during off-peak hours; less important jobs interfering with more important jobs.
- Backups of data during critical times.
- The efficiency of the PATH environment variable
- Directories that are very large

Job Scheduling

During the busiest times of the day for your system, see which processes are less critical and might be run at a different time of day. Note that there is generally increased activity just before lunch and at the end of the workday.

If you regularly run processes that take a very long time to execute, consider using **cron**(1M) or **at**(1) to execute the job during off-hours, or use **batch**(1) to execute the job when system load level permits.

Use the **crontab** command to examine users' crontab files to see if there are jobs scheduled for peak hours that could just as well run during off hours.

Encourage users to run large, noninteractive jobs (such as program compilations) at off-peak hours. You may also want to run such jobs with a low priority by using the **nice**(1) or **batch**(1) commands. As superuser, you can always change a job's priority with the **renice**(1M) command.

Whenever possible, do backups of data during off-peak hours.

Checking User Search Path Variables

Every shell process has a **path** or **PATH** environment variable that lists the directories that the system should search when looking for a command invoked by the user. Every time the user issues a command, the system scans the directories in the path to see where the command resides. If the command invokes other commands, the system has to scan the search path for them too. These searches require both processor and disk time; thus, changes here can help performance.

Some things that you should check for in user search path variables are:

Path Efficiency.
The system searches the path directories in the order listed, so your most commonly used directories should appear first in the path. Make sure that a directory is not searched more than once for a command.

Local versus NFS File Systems.
Putting directories that are on NFS mounted file systems after those that are on local disks improves performance.

Path Length.
In general, the search path should have the least number of required entries. Use link files where possible to avoid increasing search path size.

Large Directory Searches.
Avoid searching large directories if possible. Put any large directories at the end of the search path.

IMPORTANT: Do not put the current directory, represented by a dot (**.**), on any superuser search path. In user search paths, the current directory should always appear last, if at all. Placing the current directory on your path may cause you to inadvertently execute a nonsecure script or program that has the same name as a common command.

Checking Directory Size

If possible, break up large directories into smaller directories. Use the **find** command to search for large directories (those with more than 1000 entries) and consider breaking them up into smaller directories.

A common mistake that users make is to place an application, source code, and object code all in the same directory. To reduce directory search time, the solution is to create three directories: one for the application executables, one for the source code, and one for the object code. Then, for a normal user, only include the application executables directory in the search path.

Use the following **find** command to search for and list large local directories:

```
# find / -type d -size +15 -local -print | more
```

To get a quick file count for a large directory, **cd** to the directory and execute this command:

```
# ls -a | wc -l
```

End of Chapter

A Tuning Example

Use this appendix to get an idea of how you might go about tuning system performance. It is meant for information purposes only. There is no guarantee that something that caused increased performance on the system described here will cause increased performance on your system.

Example 1

CPU Type: AV5225 (25 MHz)
Number of CPUs: 2
Memory: 80 MB
PROM Revision: 5.4
DG/UX Revision: 5.4R3.10

Disks:	sd(cisc(),0) 662 MB
	sd(cisc(),1) 662 MB
	sd(cisc(),2) 662 MB
	sd(cisc(),3) 1.4GB

This system has 7 diskless clients (28 MB AV310 workstations) and 4 diskful workstations that use several of its file systems heavily.

The applications on this system are mostly desktop publishing applications. Many X Window System applications are used.

The statistics below were gathered during several periods of normal performance.

Processes

1. Is the rate of binds greater than the rate of forks? No.
2. Are there unbound runnable processes? No.
3. Are there any problems with CPU usage, per-process or system-wide? No.
4. What is the rate of context switches? 119/sec, which seems to be fine.
5. What is the load average? 0.6
6. What is the average number of processes? 127 out of 256.
7. Any process table overflows? No.

Memory

1. Is the freeswap value 15–30% of total physical memory and swap area on the system?

   80 MB (physical memory) + 126 MB (swap) = 206 MB

   15% of 206 MB = 31 MB
   30 % of 206 MB = 62 MB

   The minimum freeswap value found was 156960 blocks (about 80 MB), and the average was 232500 blocks (about 119 MB). Swap space could possibly be decreased to recover disk space.

2. Is thrashing a problem? No.

3. Is the rate of hard faults acceptable? Yes, the average is 1.0/sec, which seems fine for this system.

File Systems and Disk I/O

1. What is the average %rcache ratio? The average is 99%; perhaps adjust PERCENTSYSBUF to bring this down to 97–98%.

2. What is the rate of inode entry searches? 10.2/sec, which is fine.

3. Does there seem to be a problem with fragmentation? Yes.

   The free data blocks histogram (displayed by **dumpfs**) show many small blocks and not very many large blocks for several file systems. This is due to files being deleted and created over time.

   There also is a problem with the "ping-pong" effect. Too many volatile file systems are on the third and fourth physical disks. The command **nsar –d –WD=v** was very helpful in determining virtual disk usage.

   A major disk reorganization would probably be a good idea. First, determine which virtual disks will go on which physical disks. Second, archive each file system carefully with **dump** (or **dump2**). Third, set up the virtual disk on the appropriate disk(s). Fourth, restore each file system.

4. Is the percentage used of each read-write file system under 80%? One file system that has a high number of writes is at 82% and should be below that number. Use the command **admfsinfo –o diskuse –l** to get a quick listing of local file systems and their inode and block usage.

5. How are the service, wait, and/or response times for the physical disks?

Table A–1 Service, Wait, and Response Times Before Tuning

Disk	Service Time	Wait Time	Response Time
sd(cisc(0),0,0)	25.4	35.7	61.1
sd(cisc(0),1,0)	24.0	7.9	31.9
sd(cisc(0),2,0)	26.9	34.0	60.9
sd(cisc(0),3,0)	24.1	52.2	76.3

The higher wait times are probably due to the fragmentation/load balancing problems described above. Those numbers will most likely go down after tuning.

6. Is the swap area on one physical disk? Yes. The system administrator should distribute swap area across multiple physical disks if possible.

Terminal I/O

1. Do the output and input character rates seem normal? Yes.
2. Do you regularly monitor the size and contents of /etc/wtmp? Yes.

Networking

1. Using the command **netstat –s**, is the ratio of input datagrams dropped to the total number of datagrams received below 0.05%? Yes.
2. Using the command **netstat**, is the number of bytes in the send queue 0 for most connections? Yes.
3. Using the command **netstat –i**, is the number of input errors 0.025% of the total number of input packets? Is the number of output errors 0.025% of the total number of output packets? Yes to both questions.
4. Using the command **snmpgettab** *hostname* **public dot3StatsTable**, are there excessive collisions present for any Ethernet interfaces? No.
5. Using the command **netstat –s**, look at the bad checksums fields under the udp, tcp, icmp, and ip headings. Are any of these numbers more than an extremely small percentage (hundredths of a percent) of the total number of packets received? No.

6. Using the command **nfsstat**, is the retrans field (under "Client rpc") greater than 5% of the total number of client NFS calls? If so, what is the badxid number?

   The retrans value (22,318) is 11% of the total number of client NFS calls (194,241). The badxid number is 369, indicating that the problem is with the network itself. In practice, one remote machine is very slow and is probably responsible for the 11% figure.

Tuning Possibilities

1. Archive and/or delete unnecessary files on the file system that is above 80%.
2. Distribute the swap area across all four physical disks. Possibly decrease size of swap area.
3. Prepare for a major disk reorganization.
   a. Determine which virtual disks will go on which physical disks.
   b. Archive each file system carefully with **dump** (or **dump2**).
   c. Set up the virtual disk on the appropriate disk(s).
   d. Restore each file system.
4. Adjust PERCENTSYSBUF to bring %rcache down to 97–98%.

   The system administrator will try these one at a time and check performance statistics after each is accomplished.

Tuning Results

1. Deleting and archiving unnecessary files on the file system improved performance on that file system as expected.
2. Distributing the swap area across all four physical disks helped a lot. The system administrator dumped a few file systems and deleted those virtual disks to make room for three additional swap areas (one per disk). She then created new virtual disks for the additional swap areas (and did not make file systems on these virtual disks), added the new swap areas with the **swapon** command, made entries in **/etc/fstab** for these swap areas, and edited the **/etc/dgux.params** file to read `swapon_ARG=" -a"`.

   Next, the administrator commented out the primary swap area from the **/etc/fstab** file, since she wanted to decrease the size of that area. She shut down the system, turned off the power, and then rebooted. When the system asked for a swap area to use, the administrator answered with the name of one of the new swap areas.

After the system was rebooted, the administrator could then shrink the primary swap area's virtual disk (since it was not in use anymore). She added this swap area with the **swapon** command and edited the **/etc/fstab** file so that the system would recognize this swap area. After creating virtual disks and file systems for the file systems that were dumped, the administrator restored those file systems. Finally, she shut down the system, turned off the power, and rebooted.

The size of the entire swap area did not change much. It was 246,000 blocks before the tuning and 240,000 blocks after (60,000 blocks on each physical disk). Significantly decreasing the size of the swap area might have been seen by some users as an invitation to use more disk space.

The administrator was able to confirm that the system did begin to spread the load evenly between all four swap areas.

3. Reorganizing four physical disks can be a challenge, particularly when only a small percentage of disk space is free. Before you can move a virtual disk, there must be room at its destination. To create additional free space in which to maneuver, you must dump file systems to tape, verify the tape contents, delete those file systems, and delete those virtual disks. Keep in mind that dumping a large file system can take a while. On this system, for example, dumping a 77% full, 2,240,063-block file system took about 1.6 hours.

   The system administrator monitored the overall activity on each physical disk, noted which virtual disks were on each physical disks, and used the command **nsar –d –WD=v** to get an idea of which virtual disks were most active during normal working conditions. She chose three large file systems to dump, verified the dump tapes, and then deleted the file systems and virtual disks to create free space.

   The administrator used the feature of moving virtual disks even while the virtual disks are mounted from one physical location to another. She moved some active virtual disks to physical disks with less activity, trying to create a similar ratio of active:inactive virtual disks on each physical disk.

   After reorganizing the disks, she created virtual disks and file systems for the file systems that were dumped. The administrator then restored the file systems.

   It is very helpful to have a written plan of what you intend to do. Use the command **admpdisk –o list –p** to show the current disk layout; make a hardcopy to which you can refer. Then decide which virtual disks you want to move. List their sizes and new starting blocks. Try your plan out on paper first before actually implementing it.

The disk reorganization really produced performance results. The system users even noticed and commented upon the positive changes. Compare these service, wait, and response times for the physical disks to those values shown in Table A–1:

Table A–2 Service, Wait, and Response Times After Tuning

Disk	Service Time	Wait Time	Response Time
sd(cisc(0),0,0)	23.5	4.6	28.1
sd(cisc(0),1,0)	26.9	27.7	54.6
sd(cisc(0),2,0)	25.0	11.3	36.3
sd(cisc(0),3,0)	21.6	14.1	35.7

The administrator will continue to monitor disk activity and will make adjustments in the ongoing quest to distribute system load fairly over all physical disks.

IMPORTANT: After you've dumped a file system, deleted the file system and virtual disk, created a new virtual disk and file system, and restored the file system, users on remote machines may get a message such as "NFS file handle no longer valid." They should try to remount the file system.

If that does not work (perhaps they get the message "Device busy"), they can either reboot or rename their client's mount point for the remote directory. Then a new mount point can be created and the new remote directory can be mounted. For example:

```
# mv /mnt/remote_mount_point  /mnt/remote_mount_point.old ↵
# mkdir /mnt/remote_mount_point ↵
# mount remote_host:/mount_point  /mnt/remote_mount_point ↵
```

The old mount point (**/mnt/remote_mount_point.old**) can be removed after the next reboot of the NFS client.

4. After some of the tuning, the %rcache value went down to 98–99%. Depending upon %rcache's average value over a period of a few weeks, the administrator may try adjusting PERCENTSYSBUF slightly the next time she reboots the system.

End of Appendix

Glossary

The terms defined in this section are important to performance measurement and tuning.

Channel In the context of the DG/UX operating system's kernel, channels provide I/O access paths between application programs and files. Channels are managed by the Channel Manager.

Chatter Random data generated by electrical noise or some other malfunction. Chatter usually appears on unused terminal lines that aren't terminated properly. Chatter can easily swamp a system because each meaningless character has to be processed.

CISC (Complex Instruction Set Computer) A computer that has a robust complement of instructions. Although a robust CISC instruction can perform a lot of work, the instruction can require many CPU cycles to do it. CISC instructions are frequently implemented in microcode, which requires the overhead of retrieving micro-instructions from ROM. Contrast this to a Reduced Instruction Set Computer (RISC), which has a relatively smaller set of simple instructions with a regular structure.

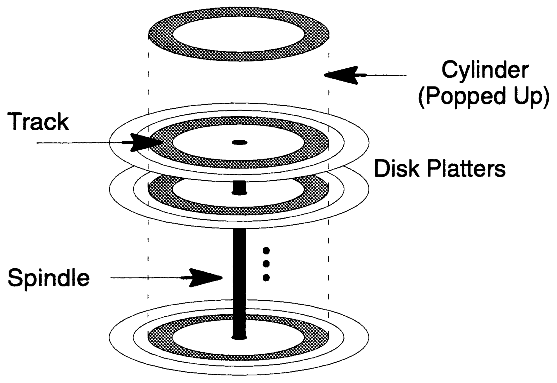
CLARiiON A disk-array storage system that provides a compact, high capacity, high availability source of disk storage. It offers high availability disk storage, in up to 20 disk modules that you can replace under power.

CMMU (Cache Memory Management Unit) The name of the logic circuitry or chip that controls the operation of the hardware cache in a CPU complex. A CMMU used in the Motorola 88K family is the 88200.

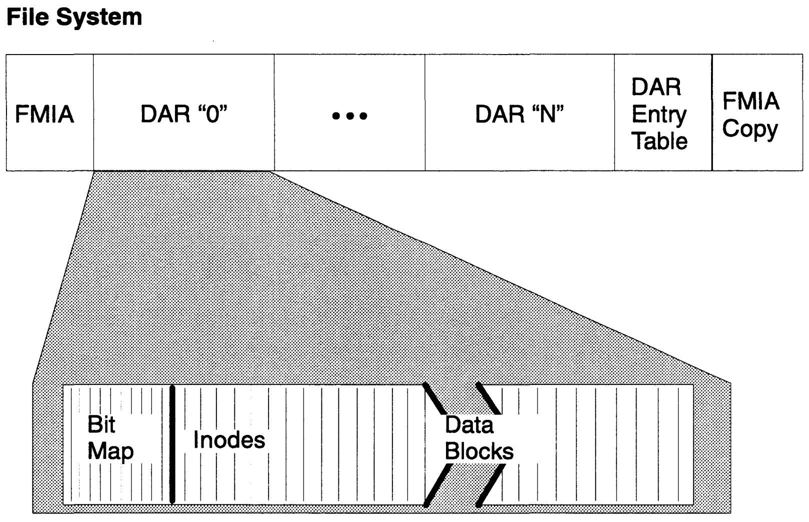
COFF (Common Object File Format) A set of System V de facto rules for portable object files (files produced by compilers and linkers). COFF in AT&T System V.4 (and in the DG/UX system) is being superseded by ELF (Executable and Linking Format). The DG/UX system continues support for COFF and enables you to link COFF and ELF executables.

CPU (Central Processing Unit) The hardware (and perhaps microcoded software) that fetches, decodes, and executes instructions, and performs arithmetic and logical operations. CPUs are classified as either CISC (Complex Instruction Set Computer) or RISC (Reduced Instruction Set Computer).

Cylinder In the context of disk hardware, a cylinder contains all of the sectors in like-numbered tracks on all of the disk's platter surfaces (data is typically stored on both surfaces of a platter). Some people use the terms *cylinder* and *track* interchangeably, but they're really not the same—a track is one concentric circle of data on a single platter's surface.



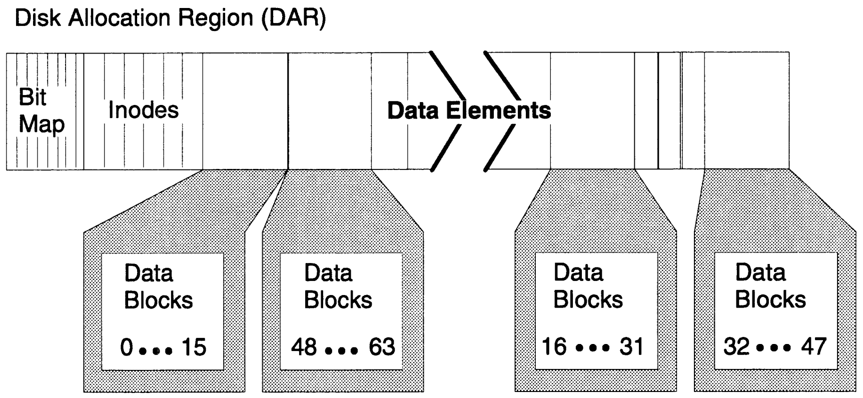
DAR (Disk Allocation Region) To increase DG/UX file system performance, the disk storage of a file system is divided into Disk Allocation Regions (DARs). To access a file, the file system alternately reads a file's inode (to find where the file's blocks are stored) and the blocks themselves. By using DARs, a file system can keep a file's data blocks and inodes physically close together, which minimizes physical disk mechanical latency (seek time).



The File Management Information Area (FMIA) contains information about the file system, including the DAR size, the number of inodes per DAR, and the default element size for files and directories. A backup copy of the FMIA is stored at the end of a file system.

Data block In the DG/UX system, a term used to describe a block of data that is stored in a virtual disk. Data blocks are 512 bytes, and are typically equal to the underlying physical disk's sector size.

Data element The logical granularity at which the DG/UX file system transfers a file's data. You can set the size of a file's data elements from 512 bytes to several megabytes. The default data element size on DG/UX systems is 8 KB, which is sixteen 512 byte disk sectors. As the figure shows, data blocks are stored as data elements in a Disk Allocation Region (DAR).



Daemon A program that is invisible to users but provides important system services. Daemons manage everything from paging to networking to notification of incoming mail. Daemons normally spend most of their time sleeping or waiting for something to do, so that they don't account for a lot of CPU load. However, some daemons can be a substantial load, such as *routed* and *rwhod.*

Disk array In the context of disk subsystems, a collection of one or more groups of disk modules and one or more SCSI busses that participate in a RAID (Redundant Array of Inexpensive Disks) redundancy scheme. Each group in an array appears to the operating system as a single physical disk.

Disk module (or spindle) A self-contained disk-drive unit. As opposed to the generic term *disk*, which could refer to a virtual disk or a physical disk.

Disk drive controllers can support different numbers of disk modules. On AViiON systems, ESDI controllers support three drives, SMD–E controllers support four drives, and SCSI controllers can currently support as many as seven devices (drives or other peripherals such as tape drives and CD–ROMs).

Disk sectors The granularity at which disk drives and controllers work. Disk sectors are a fixed size; in AViiON systems, a disk sector is 512 bytes. (Sectors for other random access devices, such as optical disks, can have different sector sizes.) Not to be confused with *data elements*.

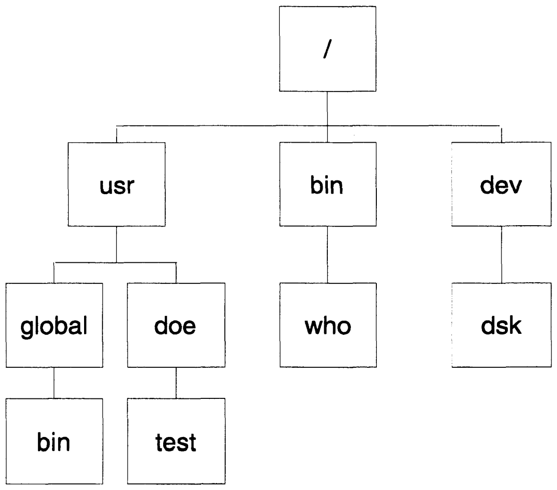
Disk transfer rate The time that it takes a disk to read or write data once the heads are over the proper track and sector. Typical disk transfer rates are in the range of 1MB/second to 2.5MB/second. A disk's transfer rate is a function of the number of sectors per track, interleaving, bit density, and rotational speed. At the same rotational speed and interleaving, a disk with more sectors per track has a higher transfer rate than a disk with fewer sectors per track.

The transfer rate of an entire disk subsystem must include the transfer rate (bandwidth) of the disk controller's data channel. On AViiON systems, typical channel transfer rates range from 1.2MB/second for ESDI controllers to 5MB/second for synchronous SCSI controllers.

ELF (Executable and Linking Format) The successor to the System V COFF (Common Object File Format), which are a set of System V de facto rules for portable object files (files produced by compilers and linkers). ELF extensions support dynamic linking and shared library capabilities. The DG/UX system supports both the older COFF format and the newer ELF format.

Ethernet A Local Area Network (LAN) standard, based on original work by Xerox, DEC, and Intel. Ethernet's commercial success resulted in the development of the IEEE 802.3 standard, which is based on Ethernet. Physically, Ethernet supports "thick" cabling, "thin" cabling, and twisted-pair cabling.

File system Within a file system, files are organized in a hierarchical, inverted tree structure, as shown in the following figure. The top of the inverted tree is the root, which is represented by the slash character (/).



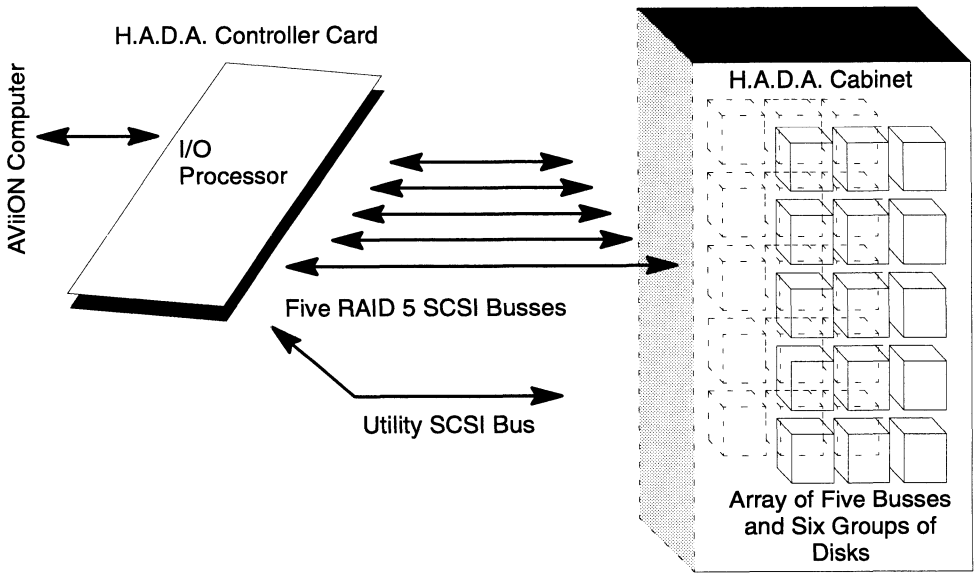
Frame In the context of main memory, a container that holds a page or part of a page. In current AViiON computers, the frame size is 4KB. Distinct from a *page*, which is a virtual memory construct.

FFM (Flat File Manager) The component of the DG/UX file system that stores all of a file system's files in one directory, where each file is identified by an inode. The FFM is not user visible.

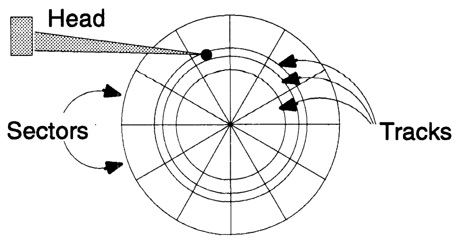
ftp (File Transfer Protocol) A TCP/IP-based application program used to transfer files among computer systems. **ftp** provides features for format translation and transfer control.

Group In the context of disk subsystems, three or more disk modules that work together in a RAID (Redundant Array of Inexpensive Disks) configuration. A group represents one column in a disk array. The operating system sees each group in an array as a physical disk.

H.A.D.A. (High Availability Disk Array) A Data General disk array implementation, which supports RAID (Redundant Array of Inexpensive Disks) configurations of as many as 30 disk modules. A H.A.D.A. disk system enables you to access all data in the disk system even if one of the disks in the array fails. The rows of the disk array are SCSI busses; the columns are groups of disk modules. The H.A.D.A. disk system supports repair under power (hot replacement of disk modules).



Heads, sectors, and tracks Disk systems read and write data with movable heads, which "fly" just off the surface of a disk's platters. Data is stored on cylindrical tracks, which are further broken up into sectors.

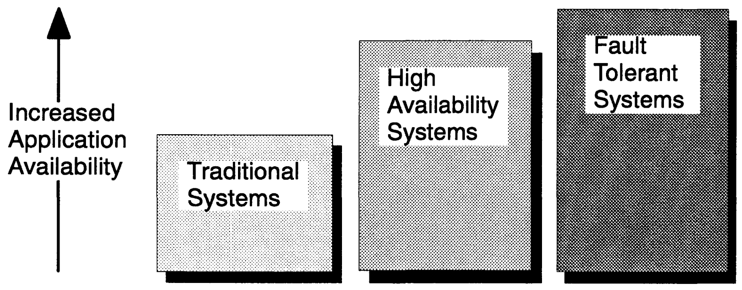


High-availability disk system A disk system that *minimizes* application downtime if a single component in the disk system fails.

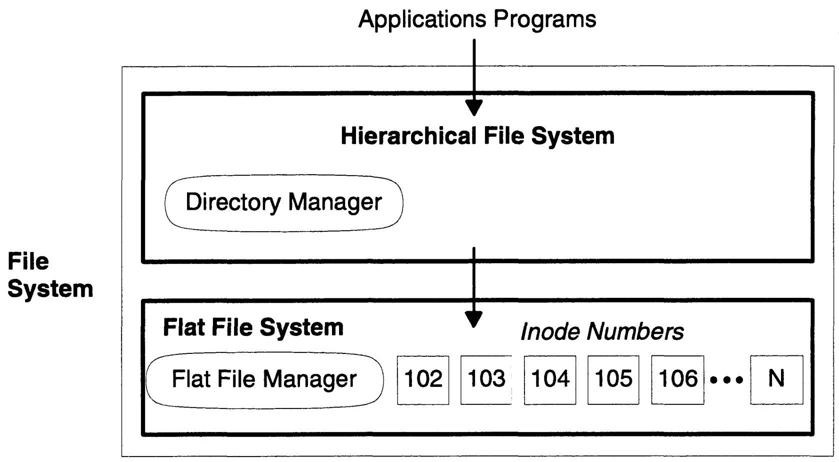
Application downtime can occur when an application cannot access its program or data files. For example, when:

- ❑ An application cannot access its files, even if the computer system is still running. Examples are when an application's file system is mounted on a disk system that failed or when a disk system bus fails.
- ❑ A computer system has to be shut down so that a component, such as a failed disk module, can be replaced.
- ❑ An application's data is being restored from a backup after a disk module is replaced.

Although high availability disk systems don't guarantee that they'll eliminate all downtime, they still provide much of the functionality of fault tolerant systems. High availability configurations provide a very cost-effective way of filling the gap between unprotected systems and fault tolerant systems.



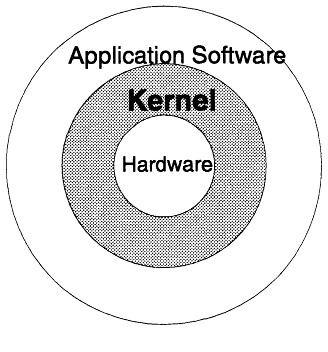
inode In a UNIX system's flat file structure, each file is identified by a unique index node (inode) number. Pointers in an inode tell the file system's Flat File Manager (FFM) where a file's data elements are stored.



Internet The global collection of publicly addressable data communication networks that are able to exchange data through a common set of protocols (such as TCP/IP) and gateways. The Internet consists of thousands of networks worldwide, containing possibly hundreds of thousands of computers.

IPC (Interprocess Communication) Any of a variety of mechanisms that enable processes to pass or share information or data. Examples in UNIX are message queues, sockets, pipes, and shared memory.

Kernel The core, hardware dependent, operating system in UNIX. Usually refers to the operating system code below the system call boundary.



Kernel Data Cache A small part of an AViiON computer's main memory that the DG/UX operating system uses to cache file system metadata. File system metadata is "data about data"—data that the file system uses to describe and locate files.

Maximum resident memory size The sum of resident shared and resident unshared memory of all processes. Shared memory is counted only once, even if it is used by multiple processes.

Memory mapping A technique that enables you to eliminate the overhead of copying data from memory into a program's data space (which occurs when programs use the **read** system call). With memory mapping (using the DG/UX **mmap** system call), application programs can map the virtual addresses of a file into their address spaces, rather than making copies of the data.

The **mmap** system call enables an application program to treat a region (or all) of a file as an array. A call to **mmap** maps the part of the file that you specify into virtual memory. You can access that part of the file directly (without using traditional **read** and **write** system calls).

Message queue A System V IPC (Interprocess Communication) mechanism, supported by the DG/UX system, that allows processes to exchange data by putting messages onto a message queue and getting messages from the queue. Message queues can be shared by more than two processes with the **msgget**, **msgrcv**, and **msgsnd** system calls.

Metadata "Data about data." In the context of the DG/UX demand-paged file I/O system, file system metadata is data, such as inodes, index elements, and directory information, that the file system uses to describe and locate files. File system metadata is cached in the kernel's data cache.

Mirroring The technique of writing the same data to separate virtual or physical disks at the same time. If one disk fails, the data is still available on the mirror-copy disk.

Mirrored disks can also perform read operations faster than single disks because the system can simultaneously read from each of the mirrors.

NFS (Network File System) A distributed file system product developed and licensed by Sun Microsystems. NFS is the key component in the Sun ONC (Open Network Computing) product set. NFS implements a subset of UNIX file system semantics, and has been ported to a variety of UNIX and non-UNIX systems. For most UNIX file system operations, NFS gives users the ability to access files over a communications link as if they were resident on the user's local machine. NFS is typically found running over Ethernet LANs.

NFS client A computer that accesses file systems hosted by other machines on the network (server machines). A single machine can act as both an NFS server and an NFS client.

NFS server A computer that hosts file systems accessed by users on other machines on the network (client machines).

NIS (Network Information Services (formerly Yellow Pages)) One of the ONC (Open Network Computing) services that provides network-wide lookup services identifying computer hosts and their users.

Page A unit of virtual memory with which a virtual memory system allocates and transfers data. Page size must be a multiple of the frame size. In the DG/UX system, the page size is 4KB (1 x frame size).

Paging area See *Swap area.*

Physical disk What the DG/UX operating system recognizes as a single disk. A physical disk can be a single disk module, a mirrored pair of RAID disks (RAID 1), or a group of disk modules in a RAID 5 array.

Pipe A UNIX interprocess communication (IPC) mechanism that connects two processes. The output from one process is used as input to the other, without the user having to manage temporary files. A pipeline can include two or more processes connected by pipes. The pipe mechanism can be used at the shell command level to string commands together on a command line. A procedural interface is also available for use within programs.

RAID (Redundant Array of Inexpensive Disks) A group or groups of disk modules that are connected together to offer high availability. There are six RAID implementations defined:

Level 0 —	disk array with striping, but no parity protection
Level 1 —	mirrored disk array with duplicated data
Level 2 —	disk array with bit-level striping and Hamming code protection
Level 3 —	disk array with byte-interleaved data and parity on one disk
Level 4 —	disk array with block-interleaved data and parity on one disk
Level 5 —	disk array with block-interleaved data and distributed parity

RAM disk A technique that uses main memory (RAM) as a virtual disk drive. The DG/UX file system supports RAM disks with the memory file system option. The RAM disk technique differs from disk caching in that a RAM disk is static—the memory is used by a few high usage programs or data files, which are loaded explicitly into RAM disk and stay there until they are explicitly removed. Main memory that is used for a disk cache is dynamic because it keeps in memory only the most recently used programs or data.

Retransmission A network's way of guaranteeing that all data arrives correctly. If a packet of network data doesn't make it to its destination or if it is damaged en route, the originator usually replaces the missing data by retransmitting the packet.

RISC (Reduced Instruction Set Computer) A name for a class of computer designs characterized by a regular and uncomplicated instruction set (typically, fewer than 100 instructions). For example, the Motorola MC88100 has 51 instructions. The uncomplicated RISC instruction set simplifies hardware design, allowing fast, highly pipelined implementations. Because of the instruction set simplicity, the instruction execution rate of a RISC machine (such as the Motorola 88K) is faster than that of a CISC machine (such as the Motorola 68K).

Most RISC machine architectures implement their instructions in silicon (hardwired logic), rather than in microcode, which provides even better performance when compared to a microcoded CISC machine. RISC machines require more sophisticated compiler optimizations to achieve high levels of performance.

Rotational latency The time that it takes for a disk sector (with the data that you want) to revolve under the head. Rotational latency is a function of the speed at which a disk spins. Currently, most popular disks spin at 3600 RPM, so their average latency time is 8.33 milliseconds—the time it takes for a disk to spin one-half revolution. Seek time and rotational latency, taken together, represent the time it takes a disk to actually find data on a disk and send the data to the disk controller. See also *Disk transfer rate*.

RPC (Remote Procedure Call) A layer of distribution service that intercepts a service request on one system in a network, packages the request for transmission over the net, unpackages the request on another system in the network for execution, then passes back the results to the original requestor. RPC is one of the bases on which a service such as NFS is built. Other system services can be built using RPC, and distributed applications may choose to use RPC.

Sector See *Heads, sectors, and tracks*.

Seek time The time that it takes to move a disk drive's heads between tracks. Seek times are usually expressed in terms of track-to-track (minimum), the average, and the maximum. The minimum seek time is the time it takes to move to an adjacent track. The maximum seek time is the time it takes to move the heads from the innermost track to the outermost track. On the AViiON series disks, the minimum seek times are in the 4 millisecond range and the maximum seek times are in the 30–40 millisecond range. Average seek times are typically 16–18 milliseconds.

Semaphore A System V IPC (Interprocess Communication) mechanism, supported by the DG/UX system, that allows processes to synchronize operations by signalling an event and waiting for an event (known as P and V operations). A process that performs a P operation increments the integer value of the semaphore; a V operation decrements the value of the semaphore. Semaphore system calls include **semctl**, **semget**, and **semop**.

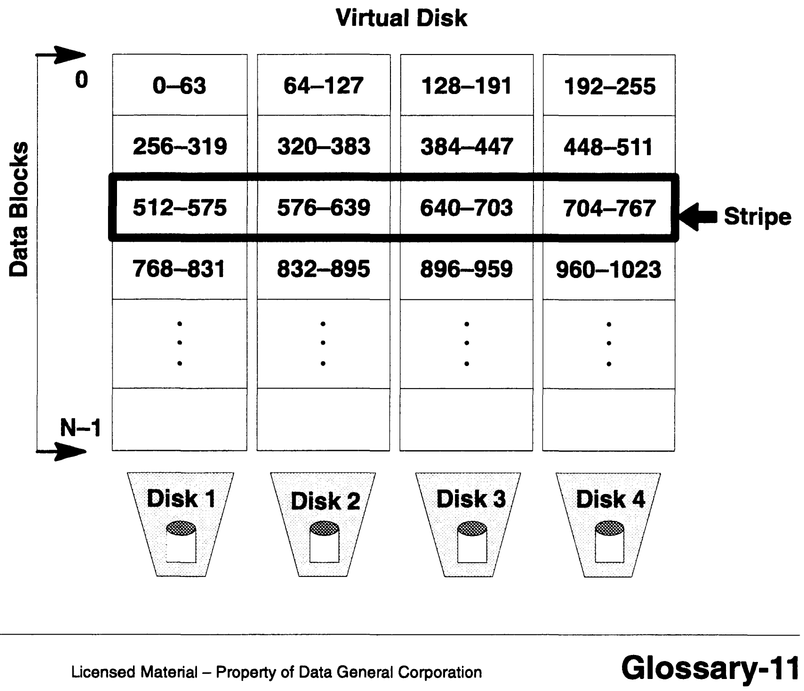
Shared library A way of storing, in one library, subroutines that are used by two or more programs. Shared libraries provide savings in disk space and in the time that it takes to load a program.

Shared memory A System V IPC (Interprocess Communication) mechanism, supported by the DG/UX system, that allows processes to communicate by having the same physical memory in their virtual address spaces. See also *Memory mapping*.

Spindle See *Disk module*

Stripe unit The amount of data that is stored on each disk module in a stripe. The stripe unit size in current high availability disk systems is 800 sectors (400KB). The stripe unit for soft striping is specified when you create a virtual disk.

Striping The technique of distributing (or interleaving) data across several disks so that data can be accessed in parallel, increasing disk I/O performance. The DG/UX operating system supports striping at both the software and hardware levels. Software-level striping works within virtual disks. The Data General high availability disk systems support hardware-level striping across disk modules in a disk group. Hardware-level striping is part of the RAID 5 design, which provides uninterrupted access to data if a disk module in the array fails.



Swap area	Refers to a part of a disk (or other secondary storage) that the kernel reserves and uses to store memory pages that must be removed from memory to make room for pages that an LWP wants to access.
Swap space	The system-wide resource for supporting virtual memory that includes physical memory and swap areas.
Tracks	See *Heads, sectors, and tracks*.

End of Glossary

Index

TIPS ORDERING PROCEDURES

TO ORDER

1. An order can be placed with the TIPS group in two ways:
   a. MAIL ORDER – Use the order form on the opposite page and fill in all requested information. Be sure to include shipping charges and local sales tax. If applicable, write in your tax exempt number in the space provided on the order form.

   b. Send your order form with payment to: Data General Corporation
   ATTN: Educational Services/TIPS G155
   4400 Computer Drive
   Westboro, MA 01581–9973

   c. TELEPHONE – Call TIPS at (508) 870–1600 for all orders that will be charged by credit card or paid for by purchase orders over $50.00. Operators are available from 8:30 AM to 5:00 PM EST.

METHOD OF PAYMENT

2. As a customer, you have several payment options:
   a. Purchase Order – Minimum of $50. If ordering by mail, a hard copy of the purchase order must accompany order.

   b. Check or Money Order – Make payable to Data General Corporation. Credit Card – A minimum order of $20 is required for MasterCard or Visa orders.

SHIPPING

3. To determine the charge for UPS shipping and handling, check the total quantity of units in your order and refer to the following chart:

Total Quantity	**Shipping & Handling Charge**
1–4 Items	$5.00
5–10 Items	$8.00
11–40 Items	$10.00
41–200 Items	$30.00
Over 200 Items	$100.00

If overnight or second day shipment is desired, this information should be indicated on the order form. A separate charge will be determined at time of shipment and added to your bill.

VOLUME DISCOUNTS

4. The TIPS discount schedule is based upon the total value of the order.

Order Amount	**Discount**
$0–$149.99	0%
$150–$499.99	10%
Over $500	20%

TERMS AND CONDITIONS

5. Read the TIPS terms and conditions on the reverse side of the order form carefully. These must be adhered to at all times.

DELIVERY

6. Allow at least two weeks for delivery.

RETURNS

7. Items ordered through the TIPS catalog may not be returned for credit.
8. Order discrepancies must be reported within 15 days of shipment date. Contact your TIPS Administrator at (508) 870–1600 to notify the TIPS department of any problems.

INTERNATIONAL ORDERS

9. Customers outside of the United States must obtain documentation from their local Data General Subsidiary or Representative. Any TIPS orders received by Data General U.S. Headquarters will be forwarded to the appropriate DG Subsidiary or Representative for processing.

TIPS ORDER FORM

Mail To: Data General Corporation
Attn: Educational Services/TIPS G155
4400 Computer Drive
Westboro, MA 01581 - 9973

BILL TO:	**SHIP TO:** (No P.O. Boxes - Complete Only If Different Address)
COMPANY NAME ____	COMPANY NAME ____
ATTN: ____	ATTN: ____
ADDRESS ____	ADDRESS (NO PO BOXES) ____
CITY ____	CITY ____
STATE ____ ZIP ____	STATE ____ ZIP ____

Priority Code ____________ (See label on back of catalog)

Authorized Signature of Buyer (Agrees to terms & conditions on reverse side)	Title	Date	Phone (Area Code)	Ext.

ORDER #	**QTY**	**DESCRIPTION**	**UNIT PRICE**	**TOTAL PRICE**

A SHIPPING & HANDLING

☐	UPS	ADD
	1–4 Items	$5.00
	5–10 Items	$8.00
	11–40 Items	$10.00
	41–200 Items	$30.00
	200+ Items	$100.00

Check for faster delivery

Additional charge to be determined at time of shipment and added to your bill.

☐ UPS Blue Label (2 day shipping)
☐ Red Label (overnight shipping)

B VOLUME DISCOUNTS

Order Amount	Save
$0–$149.99	0%
$150–$499.99	10%
Over $500.00	20%

Tax Exempt # or Sales Tax (if applicable) ____________

ORDER TOTAL	
Less Discount See B	–
SUB TOTAL	
Your local* sales tax	+
Shipping and handling – See A	+
TOTAL – See C	

C PAYMENT METHOD

☐ Purchase Order Attached ($50 minimum)
P.O. number is__________. (Include hardcopy P.O.)
☐ Check or Money Order Enclosed
☐ Visa ☐ MasterCard ($20 minimum on credit cards)

Account Number ☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐ Expiration Date ☐☐☐☐

Authorized Signature
(Credit card orders without signature and expiration date cannot be processed.)

THANK YOU FOR YOUR ORDER

PRICES SUBJECT TO CHANGE WITHOUT PRIOR NOTICE.
PLEASE ALLOW 2 WEEKS FOR DELIVERY.
NO REFUNDS NO RETURNS.

* Data General is required by law to collect applicable sales or use tax on all purchases shipped to states where DG maintains a place of business, which covers all 50 states. Please include your local taxes when determining the total value of your order. If you are uncertain about the correct tax amount, please call 508–870–1600.

Form 702
Rev. 8/87

DATA GENERAL CORPORATION TECHNICAL INFORMATION AND PUBLICATIONS SERVICE

TERMS AND CONDITIONS

Data General Corporation ("DGC") provides its Technical Information and Publications Service (TIPS) solely in accordance with the following terms and conditions and more specifically to the Customer signing the Educational Services TIPS Order Form. These terms and conditions apply to all orders, telephone, telex, or mail. By accepting these products the Customer accepts and agrees to be bound by these terms and conditions.

1. CUSTOMER CERTIFICATION
Customer hereby certifies that it is the owner or lessee of the DGC equipment and/or licensee/sub–licensee of the software which is the subject matter of the publication(s) ordered hereunder.

2. TAXES
Customer shall be responsible for all taxes, including taxes paid or payable by DGC for products or services supplied under this Agreement, exclusive of taxes based on DGC's net income, unless Customer provides written proof of exemption.

3. DATA AND PROPRIETARY RIGHTS
Portions of the publications and materials supplied under this Agreement are proprietary and will be so marked. Customer shall abide by such markings. DGC retains for itself exclusively all proprietary rights (including manufacturing rights) in and to all designs, engineering details and other data pertaining to the products described in such publication. Licensed software materials are provided pursuant to the terms and conditions of the Program License Agreement (PLA) between the Customer and DGC and such PLA is made a part of and incorporated into this Agreement by reference. A copyright notice on any data by itself does not constitute or evidence a publication or public disclosure.

4. LIMITED MEDIA WARRANTY
DGC warrants the CLI Macros media, provided by DGC to the Customer under this Agreement, against physical defects for a period of ninety (90) days from the date of shipment by DGC. DGC will replace defective media at no charge to you, provided it is returned postage prepaid to DGC within the ninety (90) day warranty period. This shall be your exclusive remedy and DGC's sole obligation and liability for defective media. This limited media warranty does not apply if the media has been damaged by accident, abuse or misuse.

5. DISCLAIMER OF WARRANTY
EXCEPT FOR THE LIMITED MEDIA WARRANTY NOTED ABOVE, DGC MAKES NO WARRANTIES, EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, WARRANTIES OF MERCHANTABILITY AND FITNESS FOR PARTICULAR PURPOSE ON ANY OF THE PUBLICATIONS, CLI MACROS OR MATERIALS SUPPLIED HEREUNDER.

6. LIMITATION OF LIABILITY
A. CUSTOMER AGREES THAT DGC'S LIABILITY, IF ANY, FOR DAMAGES, INCLUDING BUT NOT LIMITED TO LIABILITY ARISING OUT OF CONTRACT, NEGLIGENCE, STRICT LIABILITY IN TORT OR WARRANTY SHALL NOT EXCEED THE CHARGES PAID BY CUSTOMER FOR THE PARTICULAR PUBLICATION OR CLI MACRO INVOLVED. THIS LIMITATION OF LIABILITY SHALL NOT APPLY TO CLAIMS FOR PERSONAL INJURY CAUSED SOLELY BY DGC'S NEGLIGENCE. OTHER THAN THE CHARGES REFERENCED HEREIN, IN NO EVENT SHALL DGC BE LIABLE FOR ANY INCIDENTAL, INDIRECT, SPECIAL OR CONSEQUENTIAL DAMAGES WHATSOEVER, INCLUDING BUT NOT LIMITED TO LOST PROFITS AND DAMAGES RESULTING FROM LOSS OF USE, OR LOST DATA, OR DELIVERY DELAYS, EVEN IF DGC HAS BEEN ADVISED, KNEW OR SHOULD HAVE KNOWN OF THE POSSIBILITY THEREOF; OR FOR ANY CLAIM BY ANY THIRD PARTY.

B. ANY ACTION AGAINST DGC MUST BE COMMENCED WITHIN ONE (1) YEAR AFTER THE CAUSE OF ACTION ACCRUES.

7. GENERAL
A valid contract binding upon DGC will come into being only at the time of DGC's acceptance of the referenced Educational Services Order Form. Such contract is governed by the laws of the Commonwealth of Massachusetts, excluding its conflict of law rules. Such contract is not assignable. These terms and conditions constitute the entire agreement between the parties with respect to the subject matter hereof and supersedes all prior oral or written communications, agreements and understandings. These terms and conditions shall prevail notwithstanding any different, conflicting or additional terms and conditions which may appear on any order submitted by Customer. DGC hereby rejects all such different, conflicting, or additional terms.

8. IMPORTANT NOTICE REGARDING AOS/VS INTERNALS SERIES (ORDER #1865 & #1875)
Customer understands that information and material presented in the AOS/VS Internals Series documents may be specific to a particular revision of the product. Consequently user programs or systems based on this information and material may be revision–locked and may not function properly with prior or future revisions of the product. Therefore, Data General makes no representations as to the utility of this information and material beyond the current revision level which is the subject of the manual. Any use thereof by you or your company is at your own risk. Data General disclaims any liability arising from any such use and I and my company (Customer) hold Data General completely harmless therefrom.

Analyzing DG/UX™ System Performance

093–701129–02

Cut here and insert in binder spine pocket