SUN BLUEPRINTS
[illegible]
APPROVED

Sun Fire ™ Systems Design and Configuration Guide

Nathan Wiger

Roger Blythe

Sun Microsystems, Inc.
4150 Network Circle
Santa Clara, CA 95054 U.S.A.
650-960-1300

Part No. 816-7882-10
September 2002 Revision 04

Send comments about this document to: `docfeedback@sun.com`

Please
Recycle

Adobe PostScript™

CHAPTER 1

Designing Your System

Now that you have completed your statement of requirements you can work on the first half of designing a Sun Fire system—designing the logical server. By the end of this chapter, you will have completed a logical design containing a list of how many of each of the components you need, and a listing of your RAS requirements. Then, you can apply this configuration in Chapter 4 when you choose the physical system in which to place your design.

Systems design is done in this somewhat “backwards” manner for two important reasons:

- To make sure your requirements are clearly stated and met.
- *Multiple servers can be located inside one physical chassis* because Sun Fire systems support domains.

Following this process will also help ensure that you purchase a system with enough room for future expansion.

This chapter covers the following topics, which describe the logical design process:

- Understanding a Running System
- Design Rules of Thumb
- Analyzing an Existing System
- Designing for RAS
- A Logical Design Specification

Understanding a Running System

This section reviews the basics of a computer system. While this is likely all "refresher" material, many people misunderstand the real roles of computer components somewhat. As such, an analogy to a reception desk is used to help better illustrate the different role each major component plays. In this analogy we follow a receptionist answering various types of incoming calls to show how a computer manages the requests it receives.

Every computer system has three main components that can be configured:

1. I/O devices
2. CPUs
3. Memory

Of course, a Sun Fire system has many other components too, including repeater boards, the Fireplane, and so on. However, in the Sun Fire system (as with most computer systems), these are part of the fundamental architecture of the machine and cannot be configured by the customer. This fact means that to design your system, you should pay close attention to the decisions you make regarding CPUs, memory, and I/O because these decisions will directly affect the effectiveness of your design.

Notice the use of the term CPUs. Because the Sun Fire system board is sold with a minimum of two processors, it is not possible to buy a single-CPU Sun Fire system. All Sun Fires are multiprocessor systems.

I/O Devices

The Sun Fire system uses the PCI bus for all I/O. The I/O is what allows you to do anything productive with the system. Without I/O, you would have no keyboard, no network connection, no disks, and so forth.

Understanding the impact I/O has on the system is important. When something has to be done with I/O, an interrupt is generated. The CPUs must handle this interrupt. Frequently, I/O is the single-biggest resource sink on a system. This fact is especially true when you have multiple types of I/O running heavy loads concurrently, which generates a large number of interrupt requests.

For example, consider a backend database server that is front-ended by a dozen or more concurrent web servers. When a web server needs some dynamic data, it has to make a request via the network to the server, which then must do the appropriate database selects and retrieve the data from its local disk, finally shuffling the reply back across the network to the web server that requested it. This can result in a number of I/O interrupts, as the system must handle all of the network packets as well as all of the disk seeks to get the database information off disk.

When you multiply one request times a dozen or more web servers, each request times a dozen or more clients, you can see that the database server could easily become swamped with I/O interrupts, which excludes the computing power needed to run the operating system, manage memory, and run the database itself.

To tie everything together, think of I/O as each individual phone call received by a receptionist. Each phone call generates an interrupt that the receptionist must handle. Depending on the request, it may result in a lot of data transfer (talking) back to the caller. More calls generate more interrupts. Eventually, the phone system (server) hits a limit either in the amount of concurrent requests that it can handle (memory), the speed with which the requests can be fulfilled (by the CPUs), or how fast the caller and the CPUs can communicate (I/O speed).

CPUs

The CPU is actually responsible for much more than computation. Anything that puts a load on the system, including databases, web servers, email, NFS, NTP, and general network and user traffic, requires a lot of CPU power. The CPU does not do as much *thinking* as it does *handling*. Any time the system must do anything, it must ask the CPU, which has to prioritize the task, schedule it, and allocate resources for it, and do so in a way that allows all the other multitude of things going on to continue running too.

In this way, the CPU can be thought of as a busy receptionist. The receptionist has a number of standard routines. These may include forwarding calls to employees, taking messages, setting up appointments, and even providing direct responses to simple requests, “What is your address?” When an incoming phone call is received, the receptionist executes the proper routine, and completes the request if possible. If the request cannot be fulfilled in a reasonable amount of time, the receptionist may have to place the caller on hold temporarily to handle some other tasks and free up some time.

In some cases, the receptionist may receive a request that is too complex to be handled by standard routines. For example, the receptionist may receive a call that the boss is running late, and that several meetings need to be rescheduled. Here, the receptionist must do some thinking to determine which meetings can be moved to when. At the same time, the receptionist must still pay attention to other incoming calls, to ensure an important request is not missed.

If things get too busy for one receptionist to handle, you may need two or more receptionists. Some callers may even get frustrated and hang up. Even for those that do get through, there will likely not be enough time to properly answer their queries.

So, it is important to consider not only the difficulty of each request, but the volume too. In our analogy, each incoming request requires a certain baseline of time to handle properly. Typically, the receptionist will have to press a button to pick up the appropriate line, answer the call with a greeting, listen and analyze the request, then prioritize it and complete it appropriately. Even if a request consists of nothing more than “Is Mr. Johnson in?”, it still takes a certain amount of time to fulfill the request.

Memory

In the Sun Fire system, the system memory is dynamic random access memory (DRAM). The system uses memory to store things that it is using actively such as the operating system, programs, and their data.

When asked to execute a program, the system must allocate space in memory to hold an image of the program and its associated data. This space can grow or shrink as the program runs, since its resource requirements may change. In reality, most applications grow over time because they do a poor job of cleaning up after themselves.

When a system is under a very heavy load, it may run out of room in memory to hold all the information it needs. In this case, it uses predetermined disk space, known as *swap space*, to temporarily store lesser-used things from memory temporarily to make room for other things. This is known as *paging*, since it involves selectively moving specific data out of memory in sections know as *pages.* When those pages are needed, the system incurs a *page fault*, and the data is moved from disk back into memory.

In extreme situations, the system may undergo *swapping*. In this case, memory images of entire programs are moved from memory out to disk. This is a significant performance hit, and if the system starts swapping, some serious problems may occur. Unfortunately, the terms paging and swapping are often used interchangeably, perhaps because the disk storage is called “swap space,” but they are really very different.

Do not undervalue how important memory is to a running system. Not having enough memory is perhaps the single greatest cause of performance problems.

With the receptionist analogy, you can think of memory as the number of incoming phone lines available. Even if you have five receptionists (CPUs), it will not help the situation if you only have four phone lines (memory). The phone system will still be slow, since you have a bottleneck in the amount of requests you can handle concurrently. To accept another call, the current caller will have to be placed on hold (*page-out*) in order to get back to the first caller (*page-in*).

If the load gets too heavy for the phone system, and no more lines can be put on hold, calls will have to be disconnected (swap-out) to make room for others. The receptionists will then have to call the person back (swap-in), a much more time-intensive process.

Design Rules of Thumb

You can use a number of *rules of thumb* to design a system. Properly using these rules requires a firm grasp on your needs—that you have completed a statement of your requirements using the information and tables in Chapter 1 and Chapter 2.

This section describes the following design rules of thumb:

- Spread your I/O devices across as many PCI buses as possible.
- Decide how many CPUs the system needs.
- Decide how much memory the system needs.
- A well-designed system should seldom page, and never swap.
- The system should always have some idle time.
- Whenever you add additional CPUs, you should also add memory.

I/O Devices

You should always determine your I/O design first, as this along with your application needs determines your computing requirements. To get the best performance and reliability from your Sun Fire server, you should lay out the I/O carefully. An easy rule of thumb is:

Spread your I/O devices across as many PCI buses as possible.

Doing so distributes your I/O load across as many different controllers as possible, thus improving performance. In addition, you are reducing the number of single points of failure that could cause your data to go offline. Unfortunately, this rule of thumb has many caveats. Unlike CPUs and memory, the layout of your I/O intimately affects the reliability of your machine, and whether or not you can use features such as dynamic reconfiguration (DR). Chapter 4 discusses the issue of I/O design in detail, taking all of these factors into consideration.

CPUs

Regardless of what your tasks are—NFS service, CAD simulations, or compiling software builds—handling each request requires a certain baseline of time, as the receptionist example shows. Not only is the type of request important, but the quantity of requests is important too. In fact, it is often harder for a system to handle 100 small requests than 10 large ones, due to the inherent overhead of handling each request.

How many CPUs are enough? The rules of thumb you can use to help you determine how many CPUs you need are:

- One-half CPU per network card
- One-eighth CPU per I/O device (disk or tape)
- Two CPUs per application for mostly I/O-based applications (NFS, web servers and so forth)
- Four+ CPUs per application for mostly CPU-based applications (simulations, databases and so forth).

These figures assume a moderate load on your system. If you are expecting a high load on certain aspects, you should double the corresponding numbers. For example, if a system is going to have a lot of network traffic, you should have one CPU for every network card to handle the interrupts. Conversely, if you are designing a system you expect to have a very light load, cut the numbers in half or consider whether the tasks that system is going to be performing could be combined with another server to lower overhead.

To get an idea of how many CPUs you need, add up each of the criteria that affect you, then round up to the nearest multiple of two. We recommend that you only buy the four-CPU boards for your Sun Fire system. Purchasing a two-CPU board limits your future expansion room, since it takes up the same amount of space as a four-CPU board. However, there are merits to the 2-CPU board if you do not need expansion room, and the examples later in the book demonstrate a good use for it.

So, if you are designing an NFS server with a gigabit network card and six Sun StorEdge T3 arrays, you have the following CPU requirements (TABLE 3-1).

TABLE 1-1 CPU Requirements

Description	Number
Gigabit network card (1)	1/2
Disk arrays (6)	3/4
NFS server	2
Total	3 1/4

Rounding up, you should buy a four-CPU board to run this system.

Note – These rules work well for average systems. However, for high-intensity applications such as online transaction processing (OLTP), data mining, and so forth, you should research your needs more carefully. For details, see “Analyzing an Existing System” on page 8.

When buying CPUs, you should make sure you have enough memory to accommodate them, or else you run the risk of *thrashing*. This means that the system spends all its time moving things around in memory, and never does any real work. This is like the receptionist who spends time picking up phone lines and saying ”Please hold,“ without actually fulfilling any requests. ”Memory” discusses this in detail.

Finally, in terms of speed, getting the fastest processor you can buy is always an advantage. In addition to the speed of the processor, you also should consider the size of its cache. Generally this is decided for you based on the processor model, but you want to make sure to get as large a cache as possible. The cache determines how many operations can be handled at one time by the processor without having to make a trip back out to system memory. Processor cache is several orders of magnitude faster than memory, so *a large cache is always beneficial*.

Memory

Memory is perhaps the single most important part of a computer system, and has the most direct impact on performance. The more memory you have, the more things you can do, and the faster you can do them, since less disk access is needed. There is usually a greater correlation between perceived performance and memory than processors. With relational databases, for instance, being able to fit as much of the database in memory as possible can yield a big improvement in performance.

If your system is running slowly, you should probably buy more memory, not more processors. It is more likely that your system is running out of memory, not processor cycles, and is having to use swap space to run your applications.

It is possible to waste money and overbuy memory as well, though, so here some specific rules. On the Sun Fire system, memory is tied to a processor (see TABLE 1-5 in Chapter 1). So, you cannot buy a board with just memory and no CPUs. This actually simplifies the design process considerably because there are only two decisions to make:

- Whether to half-populate or fully-populate each CPU board
- Whether to buy larger or smaller DIMMs

Fully-populating a CPU board allows you to put more memory on it. In addition, though, you get better interleaving, which increases performance. Thus, the rules of thumb for memory are:

- For I/O-based applications, half-populate the CPU/Memory board.
- For CPU-based applications, fully-populate the CPU/Memory board.

Then, choose the appropriate DIMM size to provide enough memory for your application. Following these rules will naturally lead to smaller memory sizes in NFS servers (where memory is basically solely used for the file buffer cache), and larger, faster memory configurations in database and compute servers. Most systems tend to be in one category or the other, but if there is a mix, fully-populate all boards.

Remember that, as discussed previously, paging is undesirable. So, another good rule of thumb is:

A well-designed system should seldom page, and never swap.

It is possible, in fact, to run a large memory system with very little (if any) swap space. This fact is somewhat different from other commonly available information. One commonly-used phrase is “Your swap space should be double the size of your physical memory.” Consider this for a moment. You can easily design a Sun Fire system that has 64 gigabytes of memory. If you were to follow this advice, you would have to have 128 gigabytes of swap space. While a few vendors may require you to have a large swap space, you should not rely on swap for real-world memory usage, as it is too slow. When designing a system, make sure that you purchase enough memory so that your system does not swap. If it does, you need more memory.

Analyzing an Existing System

Often, the purpose of designing a new system is to replace an existing system in your infrastructure. If so, you can benefit from analyzing your existing system because this analysis will give you a better idea of what problems you are facing. This analysis is also useful if you are trying to upgrade a Sun Fire server. A proper analysis will ensure that you are upgrading the right parts of the system to address the issues.

Before you go any further, you should revisit your design goals discussed and developed in Chapter 2. Doing so will help you properly formulate your statement of the performance problems you are encountering. A good problem statement is:

When many users are logged in, NFS performance is very slow.

A bad problem statement is:

There is a large number in the `w` column of the `vmstat` output.

Always start with the perceived problems and requirements. An improvement in these areas is the only way you can tell if your design is a success. You can only make use of statistics if you know what you are looking for.

The easiest way to analyze a system is by using the `stat` commands that ship with the Solaris OE, and which can be used to monitor performance of a running system. You can get a full list of the available commands by typing the following command in a shell prompt:

```
# ls /usr/bin/*stat
```

This command will display a series of commands, such as `vmstat`, `iostat`, `netstat`, and so on.

You should never use the `uptime` command to analyze a system. You can use it to show how long your system has been up, but the notion of a *load* is very outdated and fairly useless in the Solaris OE. Most notably, load varies widely from system to system; a load of 10 may indicate a lack of activity on one machine, but extreme activity on another. We recommend you get in the habit of using `vmstat 5` instead of `uptime` when a machine seems sluggish.

Some `stat` commands are more useful than others, so the following sections focus on the useful commands (TABLE 3-2).

TABLE 1-2 Useful `Stat` Commands

Command	Description
`/usr/bin/vmstat`	Virtual memory/paging statistics with CPU/process summaries
`/usr/bin/mpstat`	Extensive per-processor statistics
`/usr/bin/iostat`	I/O and NFS statistics
`/usr/bin/netstat`	Network statistics
`/usr/bin/prstat`	Summary of active processes very similar to the `top` utility

Collecting and understanding the output from these commands should give you a good idea of what problems your current system is having, and how to improve upon these problem areas in the design of your new server.

The following sections review each command in turn, along with how to properly use each one, so you can gather the best statistics possible. It is important to note that not all options of a given `stat` command produce useful—or even trustworthy—output in all situations. The focus is on the specific parts of the output of each command that are the most important.

How and When

How and when you monitor a system is just as important as what commands you use and why you use them to collect statistics. You should make sure that you are monitoring the system when it is doing what you want it to do.

In some situations, this is relatively straightforward, such as on a multiuser interactive system. In this case, you want to run your stats during the day, when everyone is doing their normal work. Conversely, if you have a system that serves mainly as a database server, and the load gets very heavy at night when batch jobs are running, you should gather your stats overnight.

When collecting stats unattended (such as overnight), use a simple shell script that writes to a log file in `/var/tmp` with periodic timestamps. You can use something like the script in CODE EXAMPLE 1-1 to run the `stat` commands mentioned previously:

CODE EXAMPLE 1-1 `nightstats`—Script for Unattended `Stat` Collection

```
#!/bin/sh

# nightstats - Script for unattended stat collection

if [ $# -lt 2 ]; then
    echo "Usage: $0 stat args ..." >&2
    exit 1
fi

# some basic vars holding date, etc.
stat=$1
shift
date=‘date +%Y%m%d‘
logfile=/var/tmp/$stat.$LOGNAME.$date

# run the stat, writing output to our logfile
exec 1>$logfile
echo "Running ’$stat $@’ as ’$LOGNAME’"
while true
do
    date
    $stat "$@"
done
```

The way this script works you will get timestamps at each interval count you specify. So, if you run:

```
# nightstats vmstat 5 12
```

you get a timestamp every 12 repetitions. Note that the nightstats script loops indefinitely so you must manually kill it when you want it to stop. You can use this script to kick off stats in the background, either using cron or before you go home. For example:

```
# nohup nightstats vmstat 5 300 2>/dev/null &
# nohup nightstats iostat -xcnz 5 300 2>/dev/null &
```

Then, when you come to work the next day, you will have a log file in /var/tmp for each stat, with timestamps every five minutes. Each file will be named with the name of the stat command, the date, and your user name ($LOGNAME is automatically set to your user name by the shell). This will allow you to collect stats during the times when your system is under the type of load you care about.

Note – You also want to collect some stats when your system is *not* busy, which you can then use as a baseline for comparison. Otherwise, you will not be able to tell what stats change when the load increases.

Simulating Loads

Trying to simulate loads is not very useful. In general, trying to simulate a load gives you a poor—if not misleading—picture of what the system is trying to do. For example, the common practice of using dd to write to disks is usually a misrepresentative measure of I/O load. While dd reads and writes sequentially, most real-world disk access is random, and is an unpredictable combination of reads and writes. Thus, your configuration could look good on paper, and work well when running dd, but work poorly in a real-world application.

To get an accurate picture of your requirements, you should monitor a system that is running what you want it to be running. If you need to simulate this, the best way is to create a test environment that mirrors what you want to design as closely as possible. If you cannot do so, then we recommend that you use the design rules of thumb, and avoid analyzing a dissimilar system as this can cause you to make poor decisions.

What and Why

Now that you understand how and when to measure your system, the following sections examine each of the different stat commands and what they tell you.

prstat Command

When looking at your stats, the first thing you should know is what your system is doing. Seeing a large amount of disk activity by itself does not tell you anything other than the system is undergoing a large amount of disk activity. This is where the prstat command comes in. It shows you what processes are active on the system, along with how much CPU time they are using, what processor they are bound to, their size in memory, their priority, and more. If you have used the freeware tool top, the output should look very familiar.

Unlike all of the other stat commands, to use prstat you just type the command with no arguments:

```
# prstat
```

The display fills the terminal window and refreshes every 5 seconds. You should launch prstat in a separate window and keep it going as you use each of the following stat commands. That way, you can correlate the performance of your system with what is actively running on it.

vmstat Command

Memory is always the first place to start. If you have a memory bottleneck, then all of your other stats are going to be unreliable, since the system will be introducing extra delays trying to manage memory. Often memory problems are misdiagnosed as I/O or CPU problems, since disk access or applications seem slow to the user. In reality, these operations are slow because the system is paging or even swapping.

So remember: *Always start by looking at memory*. Repeat that over and over as a kind of mantra whenever you are analyzing or designing a system.

The virtual memory management algorithms in the Solaris OE are complex. Basically everything is seen as a page of memory, including files. While this is a benefit as far as the system is concerned, it makes analysis more difficult. Therefore, properly analyzing memory takes several steps.

The simplest way to look at memory is by specifying a time interval to the `vmstat` `command`, and letting it run until you press `Ctrl-C` to interrupt it. The following `vmstat` command monitors the system in five-second intervals:

CODE EXAMPLE 1-2 How to Use the `vmstat` Command

```
# vmstat 5
 procs     memory            page            disk          faults      cpu
 r b w   swap  free  re  mf pi po fr de sr s0 -- -- --   in   sy   cs us sy id
 0 0 20 1461688 510080 37 185 30 1 2  0  0  1  0  0  0  667  650  292  4  2 94
 0 0 64 1468888 197976 8 43  0  0  0  0  0  0  0  0  0  638  571  269  1  1 98
 0 0 64 1469320 198528 0  0  1  0  0  0  0  0  0  0  0  642  467  256  0  1 98
```

The first line of the `vmstat` output is a summary.

Note – Always ignore the first line of any `stat` command. It does not provide any useful information because it is a summary for as long as the system has been up. Summaries span too long a period of time, and they give you no indication as to the use of the system during that time.

When looking at the output from `vmstat` (CODE EXAMPLE 1-2), you will notice a lot of columns. You should ignore all the fields about disks and device interrupts, as there are better tools for monitoring these stats, which we will describe in subsequent sections. In fact, only some of these columns (TABLE 3-3) are really useful.

TABLE 1-3 Important `vmstat` Command Output Columns

Column Heading	Meaning
`r`	Number of runnable processes (waiting for CPU time)
`b`	Number of blocked processes (waiting for I/O, paging, and so on)
`w`	Number of runnable but swapped-out processes (normally 0)
`re`	Page reclaims (memory pages taken from other processes)
`mf`	Minor page faults
`pi`	Kilobytes paged in (including process startup and file access)
`po`	Kilobytes paged out (should be close to 0)
`sr`	Pages scanned by page-out scanner (also close to 0)
`us`	Percentage of CPU time spent in user mode
`sy`	Percentage of CPU time spent in system mode
`id`	Percentage of CPU spent idle

First, look at the `procs` headings. Normally, the `r`, `b`, and `w` columns are fairly low numbers, if not 0. This is because, generally, these columns only become nonzero if a process is waiting for something, either a CPU (`r`), I/O (`b`), or enough memory (`w`). Large numbers in these columns are usually bad.

One caveat is that you may occasionally see a steady, unchanging number in the `w` column. This means that the Solaris software has decided these processes have been idle so long they should be swapped out to make room for other things. Do not be concerned about this.

The `cpu` columns give you a good *system-at-a-glance* snapshot of what the system is doing, averaged across all processors. In general, non-idle time should be spent in roughly a 2-to-1 ratio in *`usr`-to-`sys`* modes. Also, if idle time (`id`) is close to zero consistently, you probably need some additional CPUs, especially if the `r` column is a large number. Beyond this, to get a good view of your CPUs you should use the `mpstat` command, as explained in "`mpstat Command" on page 18`.

On to memory. First, note that the `free` column should be completely ignored, as it does not in any way correspond to what is thought of as *free memory.* Because of the way the Solaris software manages memory, the `free` list does not properly count multiple processes sharing the same pages, or unused pages that have yet to be reclaimed. In addition, the file cache grows to consume most of free memory to improve performance.

Consequently, the free list tends to decrease steadily over the uptime of a system, when in fact the system is efficiently reclaiming and reusing memory.

If you want a better picture of available virtual memory, you can use the `swap` command:

```
# swap -l
swapfile             dev  swaplo blocks   free
/dev/dsk/c0t1d0s0 227,6      16 4093712 4093712
# swap -s
total: 494360k bytes allocated + 35568k reserved = 529928k used,
25137440k available
```

If both the `free` column from the first command, and the `available` column from the second command are nonzero, the system is all right. Beyond that, you can ignore the concept of free memory.

Instead, the most important column of `vmstat` is the scan rate (`sr`). This column shows the number of pages scanned in an attempt to free unused memory. The pageout scanner starts running only when free memory goes below the kernel parameter *`lotsfree`*, which is a small percentage of physical memory. When you see an increase in the scan rate, you should also see a jump in the page-outs (`po`), indicating that pages are being moved from physical memory to swap space. If you see this consistently, it is evidence of a memory shortage—*the system needs more memory*. If this only happens occasionally, then you should explore whether better job scheduling or `/etc/system` tuning could help. If not, you need more memory.

Note – A high number in the page-ins (`pi`) column is not necessarily significant. This is because when a new process starts, its executable image and data must be read into memory. Also, file system access appears in the pi column too. A large number in the `pi` column is only relevant if the `po` column is large too.

Here is an example of a system that is undergoing heavy paging because it is reading in a large file.

CODE EXAMPLE 1-3 `vmstat 5` Command Output Reading a Large File

```
 # vmstat 5
  procs     memory            page            disk          faults      cpu
  r b w   swap   free  re  mf pi po fr de sr f0 s0 s6 s7  in   sy   cs us sy id
  0 0 0  2406032 431280 8  72  2  0  0  0  0  0  1  0  1 121   87  202  1  5 94
  0 0 24 2489472 643792 0  0  1  0  0  0  0  0  0  0  0  328   86  108  0  2 98
  0 0 24 2489472 643784 61 252 483 0 0 0  0  0  5  0  9  466  718  260  3  8 90
  0 0 24 2452936 605616 1396 1753 10950 0 0 0 0 0 9 0 77 1266 2363 801 50 45  5
  0 0 24 2383216 531176 1484 1860 11822 0 0 0 0 0 53 0 40 790 1897 357 55 33 12
  0 0 24 2309576 458256 1435 1773 11475 0 0 0 0 0 69 0 23 697 1791 247 51 30 19
  0 0 24 2236608 391168 1374 1761 11008 0 0 0 0 0 52 0 40 775 1613 235 49 35 17
  0 0 24 2165824 324224 1411 1700 11291 0 0 0 0 0 75 0 16 751 1652 239 47 32 21
  0 0 24 2097680 253816 1378 1720 11012 0 0 0 0 0 0 0 87 746 1687  246 47 33 20
  0 0 24 2028800 184168 1330 2020 10614 0 0 0 0 0 73 0 11 719 1608 239 52 33 16
  0 0 24 1948016 110880 1350 1649 10790 0 0 0 0 0 56 0 37 764 1605 246 49 32 19
  0 0 24 1886176 48208 1282 1666 10187 8 8 0 13 0 1 0 89 793 1934  312 44 37 19
  0 0 24 1835416 7280 688 836 5598 5328 5529 0 6586 0 94 0 47 1088 889 238 24 30 46
  0 0 24 1803768 6680 353 675 3052 6657 6808 0 6749 0 80 0 80 1287 478 435 16 26 58
  0 1 24 1790704 15856 236 393 832 4470 4481 0 1665 0 68 0 107 2579 792 1416 11 41 48
  0 1 24 1784160 18152 35 388 812 3136 3144 0 839 0 60 0 92 1473 724 800 10 29 62
  0 1 24 1777192 18488 29 317 988 2536 2540 0 634 0 66 0 97 988 446 422 7 15 78
  0 1 24 1770768 18664 20 326 942 2334 2345 0 616 0 77 0 77 953 518 409 7 18 75
  procs     memory            page            disk          faults      cpu
  r b w   swap  free  re  mf pi po fr de sr f0 s0 s6 s7   in   sy   cs us sy id
  0 0 24 1764704 18528 37 339 820 2636 2648 0 699 0 105 0 48 961 509 343 8 20 71
  0 1 24 1757544 18640 30 264 1051 2206 2214 0 602 0 124 0 43 963 331 398 5 15 80
  0 1 24 1753544 18248 19 255 1081 2048 2056 0 880 0 97 0 70 960 323 412 5 13 81
  0 1 24 1749440 18664 20 258 1046 2048 2062 0 632 0 99 0 63 974 491 443 8 14 77
  0 1 24 1744720 18720 17 255 1009 2152 2153 0 552 0 102 0 58 1012 344 449 6 15 79
  0 1 24 1739992 18920 16 256 1008 1974 1982 0 529 0 101 0 55 929 324 379 6 16 78
  0 1 24 1735416 18800 16 261 998 2048 2052 0 536 0 107 0 55 966 315 379 5 15 80
  0 0 24 1729704 18768 54 268 833 2177 2179 0 546 0 83 0 55 862 352 338 18 13 69
  0 1 24 1728480 18816 105 403 1140 1971 1974 0 552 0 110 0 62 1027 569 492 7 11 83
  0 1 24 1728600 18888 48 196 1118 1484 1489 0 470 0 110 0 53 1014 261 484 5 10 85
  0 1 24 1728496 18832 42 191 1304 1536 1544 0 525 0 123 0 51 1000 160 455 2 8 89
  1 0 24 1728344 37712 372 143 946 1176 1178 0 318 0 103 0 43 789 84 335 3 21 76
  0 0 24 2489048 652144 3 78 32  0  0  0  0  0  6  0  5  427  310  168  1  4 95
  0 0 24 2488840 651872 0  1 11  0  0  0  0  0  1  0  1  351  134  138  0  2 98
  0 0 24 2488792 651776 0  0  3  0  0  0  0  0  0  0  0  349  114  128  0  2 98
```

Notice that for the first half of the output, there is a large number of `pi`, but no `po`, due to the file system activity of reading the file. As it progresses, though, notice the abrupt jump in `po` as well as the `sr.` Also notice how much the `pi` and user time (`us`) drop. The system is spending an inordinate ratio of time managing memory, slowing down how quickly it can read in the file.

As with all stats, brief periods of paging are not important. The purpose of having virtual memory is to allow you to temporarily exceed your available physical memory. You just want to make sure the system is not paging continuously for extended periods of time.

By now you should have a rough idea of what your system is doing. To really understand what is going on, though, you must be able to differentiate between file system pages, executable pages, and so on. To do this you can use the `vmstat -p` option.

`vmstat` Command `-p` Option

Using the `vmstat -p` option fundamentally changes the type of data reported by the command. The `-p` option replaces the columns on processes, CPUs, disks, and interrupts with extended statistics on memory and paging, and displays for executable, anonymous and file system `pi`, `po`, and `pf`.

Examine each of the three types of pages shown by the `vmstat -p` option:

TABLE 1-4 Page Types Shown By `vmstat -p` Option

Page type	Meaning
`executable`	Images of executable programs and their data
`anonymous`	Used for a process heap space, stack, and private pages
`filesystem`	Files mapped into address space through the `mmap` command

Under each page type heading are the following fields, where ? is replaced with the first letter of the page type:

TABLE 1-5 Page Stats Shown By `vmstat -p` Option

Column Heading	Meaning
`?pi`	Kilobytes paged in
`?po`	Kilobytes paged out
`?pf`	Page faults

As with the `vmstat` output, the key field is still `sr,` showing the scan rate. The benefit you get with `-p` is that you can now see what types of pages need the space, allowing you to better understand what the system is doing.

Look again at the system that is reading in a large file, only this time with the `vmstat -p` option.

CODE EXAMPLE 1-4 `vmstat -p 5` Command Output Reading a Large File

```
# vmstat -p 5
     memory           page          executable      anonymous      filesystem
   swap  free  re  mf  fr  de  sr  epi  epo  epf  api  apo  apf  fpi  fpo  fpf
 2406040 431296 8  72   0   0   0    0    0    0    0    0    0    2    0    0
 2489992 630792 0   0   0   0   0    0    0    0    0    0    0    0    0    0
 2480080 620344 785 1021 1  0   0    6    0    0   67    0    0 6174    1    1
 2417296 557472 1514 2830 0 0   0    0    0    0    0    0    0 10777   0    0
 2349520 493576 1330 2515 0 0   0    0    0    0    0    0    0 9523    0    0
 2293456 459296 1295 2684 0 0   0    0    0    0    0    0    0 9088    0    0
 2230072 399424 1256 1751 0 0   0    0    0    0    0    0    0 9881    0    0
 2164832 334864 1403 1700 0 0   0    0    0    0    0    0    0 11212   0    0
 2097432 267288 1415 1716 0 0   0    0    0    0    0    0    0 11212   0    0
 2021736 192344 1330 2024 0 0   0    0    0    0    0    0    0 10638   0    0
 1947168 122688 1330 1604 0 0   0    0    0    0    0    0    0 10558   0    0
 1883288 59216  1324 1658 0 0   0    0    0    0    0    0    0 10568   0    0
 1832784 12056 836 863 3936 0 5059   1    0   76    1 3548 3846 6808    4   12
 1798648  8656 207 654 6531 0 6519   0    0   72  353 6374 6446 1502    1   12
 1787016 17864 49 461 4094  0 927    6    0    6  646 4076 4084   12    3    3
     memory           page          executable      anonymous      filesystem
   swap  free  re  mf  fr  de  sr  epi  epo  epf  api  apo  apf  fpi  fpo  fpf
 1776040 18448 38 530 3036  0 678    8    0    6  774 3020 3028    3    0    1
 1766800 18488 32 319 2592  0 625    4    0    3  952 2585 2585    1    1    3
 1761080 18696 32 309 2465  0 549    0    0    1  963 2457 2460    1    0    3
 1754696 18600 31 302 2420  0 534    0    0    8  937 2406 2412    1    0    0
 1748608 18640 30 308 2488  0 534    3    0    3  945 2483 2484    1    0    0
 1742504 18784 23 285 2318  0 508    3    0    6  968 2304 2307    3    1    4
 1736960 18784 21 291 2268  0 491    3    0    8  979 2252 2259    3    1    1
 1731008 18584 94 291 2369  0 535    0    0    9  811 2355 2358    3    0    1
 1729800 18744 75 214 1697  0 497    0    0    4 1112 1689 1692    1    0    0
 1729840 18664 57 202 1601  0 538    0    0    4 1156 1587 1595    0    1    1
 1881984 149608 470 122 984 0 366   30    0    6  728  972  976    0    0    1
 2490440 672488 0   0   0   0   0    0    0    0    0    0    0    4    0    0
 2490744 672672 10 168  0   0   0    8    0    0    6    0    0   16    0    0
 2490768 672512 0   2   0   0   0    0    0    0   16    0    0    0    0    0
```

As you can see, this makes what is happening to the system much clearer. The system starts by paging in the file very effectively, until it hits the *lotsfree* limit and the page-out scanner starts. At this point, there is a big jump in the `sr` column. Also notice the abrupt shift from file system page-ins (`fpi`) to anonymous `pi`, `po`, and `pf`. This means that pages are being taken from other processes to make room for the file in memory. Thus, if you see a lot of activity in the `apo` and `sr` columns, you need more memory.

While memory analysis can be complicated if you pay attention solely to the `sr` and `po` columns, you should be able to tell if your system needs additional memory.

mpstat Command

The Sun Fire system is designed to be a multiprocessor system, as evidenced by the fact that you cannot even buy a system with only one CPU. Even though you are looking at CPUs secondarily, being processor-bound is the least likely candidate for bad performance. If anything, you are exploring CPUs secondarily so that you can double-check this assumption, and rule it out as a possible factor. CPUs usually only become a factor in heavily loaded systems that are doing lots of interactive or transactional processing. In most other cases, if you buy enough system boards to hold all your memory, the CPUs that are included are usually sufficient.

As mentioned previously, the `cpu` columns of the `vmstat` output are a good place to start. Generally, a large percentage of idle time indicates that your processing power is sufficient. However, measuring idle time across a lot of processors can mask situations such as one processor getting swamped with interrupts while the rest do nothing. So, it is important to look at your CPUs in detail to make sure you are not missing anything.

Like `vmstat`, just launch `mpstat` with a time interval and let it run:

CODE EXAMPLE 1-5 How to Use the `mpstat` Command

```
# mpstat 5
CPU minf mjf xcal  intr ithr  csw icsw migr smtx  srw syscl  usr sys  wt idl
  0  372   2  836   447  300  393   47   25   26    1   918   23  10   0  67
  1  370   2  622   543  523  301   40   23   35    0   932   24  11   0  65
  2  376   2  527   151  100  396   48   25   26    0   926   24  10   0  66
  3  372   2  531   151  100  397   48   25   26    0   921   23  10   0  67
CPU minf mjf xcal  intr ithr  csw icsw migr smtx  srw syscl  usr sys  wt idl
  0  229   0  546   400  300  458    0   12   13    1   563    2   9   0  89
  1  132   0 2018   585  585  111    0    9   16    0   621    4   8   1  88
  2  265   0  199   100  100  354    1    9   15    0   770   21   9   1  68
  3  363   0  491   101  100  671    1   14   18    0  1339   22  11   0  67
CPU minf mjf xcal  intr ithr  csw icsw migr smtx  srw syscl  usr sys  wt idl
  0  155   0  445   400  300  495    0   12    7    0   398    1   6   0  92
  1   99   0  145   348  347  134    1   10   10    0   487   13   4   0  83
  2  154   0  401   101  100  255    1    8    4    0   723   21   5   0  73
  3  307   0  227   100  100  178    0   11    9    0   989   23   8   1  69
```

This command produces a lot of columns, only some of which you care about:

TABLE 1-6 Important `mpstat` Command Output Columns

Column Heading	Meaning
`xcal`	Interprocessor cross-calls
`intr`	Interrupts
`csw`	Context switches
`icsw`	Involuntary context switches

TABLE 1-6 Important `mpstat` Command Output Columns *(Continued)*

Column Heading	Meaning
`smtx`	Spins on mutex locks
`usr`	Percent user time
`sys`	Percent system time
`wt`	Percent wait time
`idl`	Percent idle time

A cross-call (`xcal`) is a call used by a processor to tell other processors to do something. Cross-calls are used for a variety of things, such as delivering a signal to another processor or ensuring virtual memory consistency. This latter use is very common, as it happens during file system activity. Heavy file system activity (such as NFS) can result in a lot of cross-calls. Also, it is not unusual for the `boot proc` to show thousands of `xcal`s, as it maintains lots of information about the others.

An interrupt (`intr`) is the mechanism that a device uses to signal to the kernel that it needs attention, and some immediate processing is required on its behalf. I/O is the major contributor of interrupts, although there are also “special” interrupts such as the system-wide clock thread that occurs regularly. Interrupts, unlike everything else, are not distributed across all CPUs. Instead, the Solaris OE binds each source of interrupts to a specific CPU.

The term context switch (`csw`) refers to the process of moving a thread on and off a CPU. Context switches are a normal but somewhat expensive occurrence because switching context involves certain overhead, such as populating the stack. Normally, a context switch occurs when a process is done with the CPU and another process is given a chance to run. Thus, a steady number of context switches is insignificant.

Involuntary context switches (`icsw`), on the other hand, are much less favorable. When a process is given access to the CPU, it is has a limited time window in which to run, depending on how many other processes are running, what their priority is, and so on. This is the nature of scheduling. An involuntary context switch means that the process was forcibly stopped by the scheduler before it was finished; the time allotted was too short for the process to finish in, or a higher-priority thread preempted it. A few of these is nothing to be concerned about, but getting a large number of these regularly indicates that the system does not have enough processing power to handle all of the things that need to run. You need additional CPUs.

Finally, a spin on a mutex lock (`smtx`) happens when a thread cannot access a section of the kernel that it needs on the first try. The term mutex is short for a *mutual exclusion* lock, and is used in multithreaded operating systems like the Solaris OE to allow multiple threads to run concurrently in system mode. When a thread enters system mode, it locks the part of the kernel it is using by acquiring the mutex lock for that section. Once the thread is finished, it releases the lock so other threads can have access. A spin happens when two threads want the same section of the kernel, and one of them has to wait for the other to finish. A few of these are perfectly normal, but a large number means there is contention for the kernel resources.

The difficult thing about CPU performance analysis is that it is nearly impossible to provide hard-and-fast rules of thumb. For example, a high number of interrupts is not necessarily a negative. If you have a lot of I/O, you are going to have a lot of interrupts. Instead, it is the interaction of several statistics that can tell you if you are operating efficiently or having serious problems.

When looking at the `mpstat` command output, you always want to take into account the last four columns—the amount of CPU time spent in the different modes. The only real rule of thumb is:

The system should always have some idle time.

Having consistently low idle time means that your system is getting pushed to its limits in one way or another. Even if your system is properly tuned and running at peak efficiency, if you ran anything else on it you would be out of processing capacity.

However, it is also possible for the system to have a significant amount of idle time, but the system still needs more CPUs. Why? Because the amount of time a CPU spends handling device interrupts cannot be seen by the operating system as device interrupts are higher priority than the clock interrupt. This means the time it takes a CPU to handle network and I/O interrupts will be reported as idle time. The `intr` column only lists disk-based interrupts. (This is different from wait time, which is after an interrupt has been handled and the CPU is waiting for a response from the device.)

Because CPU analysis is so complicated, TABLE 3-7 can help you decipher the different stats. The terms "high" and "low" are used because the exact numbers are very system dependent.

TABLE 1-7 Analyzing the `mpstat` Command Output

If you see this...	It probably means...
High `intr` High `idl` Low `usr`	Your system is very busy handling I/O interrupts.
High `intr` High `sys` Low `usr`	If the system is an NFS server, this is perfectly normal. Otherwise, the system is very busy handling I/O interrupts.
High `intr` High `wt`	I/O requests are taking a long time to fulfill. This is likely an I/O performance problem, not a CPU resource problem. Check `iostat`.

TABLE 1-7 Analyzing the `mpstat` Command Output *(Continued)*

If you see this...	It probably means...
High `smtx` High `sys` or `idl`	Contention for kernel system resources exists.
High `icsw`	Contention for basic CPU resources exists.
High `csw` or `xcal` High `sys` Low `usr`	If this happens consistently, you may require more CPUs, depending on your applications. If you are not noticing any slowness in applications or system problems, however, ignore it.
High `sys` Low `usr` All other stats low	Your system is spending too much time managing resources. Check `vmstat` first.

One nice thing is that the solution to all of these problems is the same. The system needs more and/or faster CPUs. Once again though, the importance of having enough memory is emphasized here. When you add a CPU, you incur additional overhead in the form of more kernel space needed to manage that CPU, and space for that CPU to do its own work. Therefore, the rule of thumb is:

Whenever you add additional CPUs, you should also add memory.

Doing so will help prevent accidental memory shortages, which can actually make your system run *slower* as you add more CPUs.

`iostat` Command

Proper I/O layout is complicated; it is almost never done right the first time. Part of the reason for this is that usage patterns and requirements change over time. Also, where you add memory and CPUs is somewhat predetermined. Where you add disk devices and controller cards, though, has a big impact on the system. Therefore, it is important to make sure that the I/O layout is flexible enough to handle future changes and expansion.

Fortunately, I/O analysis is very straightforward. There is only one version of the `iostat` command to run, `iostat -zxcn`.

CODE EXAMPLE 1-6 How to Use the `iostat` Command

```
# iostat -zxcn 5
<summary omitted>
cpu
 us sy wt id
  0  1  5 93
                    extended device statistics
    r/s    w/s   kr/s   kw/s wait actv wsvc_t asvc_t  %w  %b device
    0.0    0.2    0.0    1.6  0.0  0.0    0.0    8.2   0   0 c0t0d0
    0.0    0.2    0.0    1.6  0.0  0.0    0.0   10.1   0   0 c5t0d0
    0.0   34.6    0.0 2201.0  0.0  1.0    0.0   27.9   0  97 c12t1d0
    0.0    0.2    0.0    1.6  0.0  0.0    0.0   12.2   0   0 c20t122d0
    0.0    0.2    0.0    1.6  0.0  0.0    0.0   14.1   0   0 c20t98d0
    0.0   14.2    0.0  113.6  0.0  0.1    0.0    5.5   0   8 c20t101d0
    0.0    0.2    0.0    0.5  0.0  0.0    0.0   30.1   0   1 c10t1d0
    0.0   58.4    0.0  135.4  0.0  0.3    0.0    5.6   0  30 c2t17d0
    1.0   12.8    8.0  135.0  0.0  0.2    0.0   11.6   0  13 c2t16d0
    0.0    3.4    0.0   19.2  0.0  0.1    0.0   17.9   0   4 c2t9d0
    0.0    0.4    0.0    0.8  0.0  0.0    0.0    5.3   0   0 c2t21d0
    0.0    1.8    0.0    1.4  0.0  0.0    0.0    4.2   0   1 c27t42d0
    0.4    9.2    3.2  155.9  0.0  0.1    0.0   10.9   0   8 c28t69d0
    0.0    9.0    0.0  157.5  0.0  0.1    0.0    9.0   0   6 c28t68d0
    0.0    1.8    0.0    1.4  0.0  0.0    0.0    4.8   0   1 c29t1d0
    0.0    9.0    0.0  157.5  0.0  0.1    0.0    9.5   0   7 c30t35d0
    0.0    0.4    0.0    0.8  0.0  0.0    0.0    5.1   0   0 c30t52d0
    0.0    9.2    0.0  155.9  0.0  0.1    0.0   10.3   0   7 c30t36d0
    0.0   58.4    0.0  135.4  0.0  0.4    0.0    6.5   0  35 c31t66d0
    0.4   12.8    3.2  135.0  0.0  0.2    0.0   12.1   0  12 c31t64d0
    0.0    3.4    0.0   19.2  0.0  0.1    0.0   17.7   0   4 c31t90d0
    0.0    0.0    0.0    0.0  0.0  0.0    0.0    0.0   0   0 tomax:/export/mirrors/
pkg.eng/export/pkg
    0.0    0.2    0.0    0.4  0.0  0.0    0.1    1.0   0   0 twinsun-n1:/export/workspace/
d0/nwiger
```

For this version of the `iostat` command, the output shows extended statistics for only those disk devices with nonzero activity, by physical device path instead of the logical kernel disk name (that is, `c0t0d0` instead of `sd0`). If you are using individual disk partitions, you may also want to use the `-p` option. However, most production environments manage their disks with some type of volume manager package, so in practice this option is not that useful.

As with the other stat commands, there are only a few columns you care about (TABLE 3-8).

TABLE 1-8 Important iostat Command Columns

Column Heading	Meaning
kr/s	Kilobytes read per second
kw/s	Kilobytes written per second
wait	Number of transactions waiting for service
wsvc_t	Average service time in wait queue, in milliseconds
asvc_t	Average service time for active transactions, in milliseconds

You can ignore two commonly used columns, %w and %b, which are supposedly the percentage of time spent waiting and busy, respectively. Because of the complexity of modern disks and controllers, these calculations are very inaccurate. Often the two will total more than 100 percent, which should be impossible. Besides, these columns do not tell you anything that you cannot find out by looking at wsvc_t or asvc_t.

Analogous to the mpstat command, when looking at iostat you should always watch the first two columns listed (kr/s and kw/s) to see how much activity the disks are undergoing. Then, basically, the last three columns should be as close to zero as possible. This indicates that the system has very fast disks, and that the I/O is laid out correctly to avoid controller bottlenecks.[1]

In practice, asvc_t will be nonzero for any disks undergoing activity, since it always takes some amount of time for a disk to fulfill a request. As with any stat, you will only be able to tell if the system is particularly busy after establishing a baseline. However, several facts are true:

1. Service times across equally active disks should be fairly even.
2. You should not see huge peaks and valleys under normal conditions.
3. You should rarely, if ever, see a nonzero number in wait or wsvc_t.

You may, occasionally, see a temporary jump in service times (asvc_t) even though there is nothing apparently going on (that is, kr/s and kw/s are almost 0). This is due to a somewhat strange behavior of fsflush, the daemon responsible for flushing disk buffers. Periodically, it will generate a long, random series of writes in a short time period. This results in a queue forming, which bumps up the service time, even though there is no real apparent activity on the disk. If you see this, ignore it.

1. Without the -n option, wsvc_t and asvc_t are combined into a single svc_t column.

Now look at some output from a small NFS server undergoing a fairly heavy load from several different concurrent requests to read and write several files:

CODE EXAMPLE 1-7 `iostat` Command Output On an NFS Server

```
# iostat -zxcn 5
     cpu
 us sy wt id
  2 72 13 13
                    extended device statistics
    r/s    w/s   kr/s   kw/s wait actv wsvc_t asvc_t  %w  %b device
    0.2    0.2    1.6    1.6  0.0  0.0    0.0   11.9   0   0 c0t0d0
    0.0    0.2    0.0    1.6  0.0  0.0    0.0   13.0   0   0 c0t8d0
   24.6   42.4  390.9  675.5  0.0  0.9    0.0   13.9   0  34 c4t17d0
   25.6   42.6  409.4  681.3  0.0  0.9    0.0   13.6   0  34 c4t20d0
   25.4   43.6  403.4  694.5  0.0  0.9    0.0   13.0   0  35 c4t22d0
   24.6   43.0  393.5  687.7  0.0  0.9    0.0   13.9   0  34 c4t4d0
   25.0   42.4  397.8  676.1  0.0  1.0    0.0   14.3   0  36 c4t18d0
   24.2   42.8  385.5  684.5  0.0  0.9    0.0   13.4   0  33 c4t16d0
   24.8   43.4  393.9  691.3  0.0  0.9    0.0   13.8   0  34 c4t3d0
   25.2   43.8  403.0  700.5  0.0  1.0    0.0   13.9   0  36 c4t2d0
   25.6   43.4  409.4  694.1  0.0  0.9    0.0   12.9   0  32 c4t21d0
    0.0  132.9    0.0 7936.8  0.0  5.5    0.0   41.5   0  84 c4t6d0
   25.2   43.8  403.0  700.5  0.0  1.0    0.0   13.8   0  34 c4t1d0
   25.2   42.0  403.0  671.7  0.0  0.8    0.0   12.6   0  31 c4t19d0
     cpu
 us sy wt id
  1 63 18 18
                    extended device statistics
    r/s    w/s   kr/s   kw/s wait actv wsvc_t asvc_t  %w  %b device
    0.0    5.8    0.0   44.2  0.0  0.4    0.0   63.3   0   3 c0t0d0
    0.0    5.8    0.0   44.2  0.0  0.3    0.0   58.5   0   3 c0t8d0
   25.6   42.8  407.7  685.0  0.0  0.9    0.0   13.7   0  34 c4t17d0
   25.2   43.8  400.5  698.3  0.0  0.9    0.0   13.4   0  35 c4t20d0
   25.4   43.0  403.7  685.4  0.0  0.9    0.0   13.8   0  34 c4t22d0
   25.2   43.0  403.3  688.2  0.0  1.0    0.0   14.0   0  35 c4t4d0
   25.6   43.6  405.3  693.5  0.0  1.0    0.0   14.3   0  37 c4t18d0
   25.4   42.4  404.9  678.6  0.0  0.9    0.0   13.5   0  35 c4t16d0
   25.2   43.2  398.5  688.7  0.0  1.0    0.0   14.5   0  36 c4t3d0
   25.2   43.6  400.3  694.9  0.0  1.0    0.0   14.0   0  36 c4t2d0
   25.2   43.4  403.3  694.7  0.0  0.9    0.0   12.6   0  32 c4t21d0
    0.0  134.8    0.0 7922.1  0.0  5.6    0.0   41.7   0  86 c4t6d0
   25.2   43.4  400.5  691.9  0.0  1.0    0.0   14.5   0  37 c4t1d0
   25.4   43.2  402.0  687.0  0.0  0.9    0.0   13.2   0  34 c4t19d0
```

If you look at `kr/s` and `kw/s`, the disks on `c4` are moving a lot of data in reads and writes. On this server, these are laid out in a striped/mirrored logical volume mounted as `/export`. From the output, it appears that the volume manager software is doing a good job making sure that the load is spread evenly. The one hot spot is on `c4t6d0`, which happens to be the volume log. This disk is getting hit heavily because all of the transactions must be logged. While much higher than the others, this disk's performance is still well within acceptable numbers because the `asvc_t` is not even over 100. This means that the requests are being fulfilled in a very reasonable amount of time.

Notice that in the second set of output, there is a jump in the `asvc_t` on `c0`, even though there is no real activity (the disks on `c0` are set up as a mirrored root volume). This is likely due to the peculiar activity of `fsflush` mentioned earlier, so you can ignore it. Look at how the `asvc_t` is higher than the disk doing 7922kw/s.

Note that the majority of CPU time was spent in system mode (`sy`), based on the CPU snapshot (provided via the `-c` option). Since this is an NFS server, this activity is nothing for concern. If this was a system for local users doing interactive work, however, you might want to rerun `vmstat` and `mpstat` to make sure there are no memory or processor bottlenecks.

While finding I/O problems is fairly easy, solving I/O problems can be quite difficult. Requirements for I/O vary widely, so a trial-and-error approach is the only reliable way to get good performance from your I/O. Even within a single data center, different servers may undergo vastly different usage patterns and encounter different types of problems.

Do not try to micromanage hot spots on disks. By the time you get everything tuned correctly, the usage patterns will change. This means you will then have a *worse* configuration than you would have otherwise, since it is tuned to match your now-inaccurate requirements. Instead, use your time making sure that:

1. The volumes are spread across as many controllers as possible.

2. You use the proper type of volume for your requirements.

3. You use the proper stripe unit size to match your needs.

These simple steps are often missed, but will solve virtually any disk I/O performance problems. *Just changing the stripe unit size from its default can result in huge performance gains*.

The final caveat on I/O layout is fulfilling the RAS requirements of your system, including making sure it supports dynamic reconfiguration (DR) if you require it. Since these are both major concerns on their own, Chapter 4 discusses this issue and I/O design in detail, taking all of these factors into consideration.

`netstat` Command

Network analysis can be difficult only because the Solaris software does not currently have a solid network utility that really tells you everything you want to know. While you can get a general idea of number of packets, you cannot see things like octets, TCP/UDP throughput rates, or retransmissions. Fortunately, improving the network performance of a system usually amounts to installing an additional network interface card for more bandwidth, even if it is a bit of a “black box” approach.

Despite its limitations, you can tell several things from the `netstat` command output. Unlike the other stats, you must run the `netstat` command separately for each interface you have configured by specifying the `-I` option along with the interface name.

CODE EXAMPLE 1-8 How to Use the `netstat` Command

```
# netstat -I ge0 5
    input   hme0      output           input  (Total)    output
packets errs  packets errs  colls  packets errs  packets errs  colls
909076714 0     837319344 0     0      918674892 0     846917522 0     0
667     0     681     0     0      673     0     687     0     0
426     0     402     0     0      428     0     404     0     0
1886    0     3684    0     0      1886    0     3684    0     0
1878    0     3117    0     0      1882    0     3121    0     0
411     0     391     0     0      411     0     391     0     0
```

You can tell two things from this display:

1. Total number of packets received (input) and transmitted (output) during that interval, both for that interface (left set of columns) and for all interfaces (right set of columns). This is *not* an average per second, but a total count.

2. Number of errors and collisions, which should always be low or zero.

Network capacity is very difficult to gauge with this limited information. Without the sizes of each packet, it is impossible to know if you are anywhere near the throughput limits for the interface you are analyzing. Given this information, if the network seems slow, and you are seeing thousands and thousands of packets each second, try adding another network interface card to see if it helps. If not, you should examine your network as a whole to see if you have more widespread issues.

Many available freeware tools, such as the SE Toolkit and Multi Router Traffic Grapher (MRTG), provide better network analysis than `netstat`. You can use tools such as these to more properly gauge the bandwidth being used by each interface. MRTG is especially useful, as it graphs utilization over time so you can easily see when your network interfaces are getting busy, as well as how much bandwidth they are pushing.

Analysis Reveals...

By this point, you should have a good idea about where the system is weak. Make sure you have good notes, as you need this information in the next chapter when you design your new system.

Giving performance tuning a full treatment is beyond the scope of this book. True performance tuning gets exponentially harder; it is much more difficult to get the last 10 percent out of a system than the first 90 percent. If you are interested in high-end performance tuning, read *Sun Performance and Tuning—Java and the Internet, 2nd Edition* by Adrian Cockcroft and Richard Pettit (ISBN 0-13-095249-4) and “Application Performance Optimization” by Börje Lindh—Sun Microsystems AB, Sweden Sun BluePrints™ OnLine—March 2002.

Designing for RAS

This is the final step in the design process. By now, you should have a fairly clear understanding of what your requirements are, as well as any possible problems with your existing system. Up until now, this book focused mainly on performance because you should make sure any solution you develop can meet your fundamental application requirements. However, properly designing for RAS is just as important, and requires some thought.

Always keep three principles in mind when designing for RAS:

- The more RAS you want, *the more hardware you must add* to the system.
- RAS is not just a function of the Sun Fire server, but of your *entire site.*
- Maximizing RAS can decrease performance.

The first point is almost always overlooked. As an example, to effectively use DR, you should *add* boards in your design beyond those required for your applications. Why? Because otherwise, when the system dynamically reconfigures a board out of the system, it will not have enough resources to run your applications. The system could start paging, or the CPUs could get too busy handling I/O interrupts to do any real work. *The requirements you have formed up to this point are the minimum you need for your system.*

As for the second point, purchasing redundant power supplies does not benefit you if your site has only a single power grid with no UPS system. RAS is a function of your entire site, not just one server in isolation. As with performance, getting that final 10 percent of reliability out of a site gets exponentially more difficult—and costly. Therefore, you should be realistic about both your requirements and expectations—and your ability to fund them.

Third, taking advantage of certain RAS features and methodologies can decrease the performance of your system. For example, if you mirror file systems, for each write the system must now perform *two* writes, one to each half of the mirror. Some of these effects can be mitigated, for instance by placing the two halves of the mirror on different I/O controllers.[1] However, such performance hits can add up, so it is important to realize it is impossible to maximize both RAS and performance.

1. In fact, many volume managers will "round robin" between the two halves of a mirror on reads, actually increasing your read performance over a single disk.

Uptime Requirements

You were first asked to consider your uptime requirements in Chapter 2, “What are the uptime requirements of the system?” To help answer this question, you can consider the following:

- How much time do you have available for planned maintenance?
- How long can you afford to be offline during an unplanned downtime?

There are two types of downtime—planned and unplanned. Planned downtime includes hardware and software upgrades, whereas unplanned downtime includes system crashes and emergency reboots. All computer systems have some amount of downtime; the goal of a good server design is to minimize the impact this downtime has on your organization.

For some organizations, scheduled maintenance is not an issue; the systems undergo heavy usage during the day from employees, so taking the machine down after-hours is a viable solution. Other organizations, however, serve a worldwide audience and can afford little scheduled maintenance due to time zone differences. Also, it is not uncommon to have a mix of different requirements for different systems at a single site. One thing that every organization has in common, though, is the desire to minimize unplanned downtime as much as possible.

There is no reason to differentiate between the two types of downtime, other than to help you come to a conclusion regarding your overall requirements. When you have a good idea of the uptime required for this system, TABLE 1-9 will help you determine what your design should include to ensure that its RAS properties meet your requirements.

Note – You should always purchase redundant SCs for a system to ensure availability in the event of a System Controller board failure. Without a functioning System Controller board, none of the domains in a system will work.

TABLE 1-9 RAS Design Decision Table

Allowable downtime	Your design should include...
Some	Redundant fan trays Redundant power supplies and transfer switches[1]
Little	Redundant CPU/Memory boards DR for CPU/Memory boards Volume management software (such as Solaris™ Volume Manager (SVM) or VERITAS Volume Manager (VxVM)
Very little	Redundant paths to I/O devices Multipathing software for I/O (such as Multipath I/0 (MPxIO) or VERITAS Dynamic Multipathing (VxDMP) Redundant network connections Multipathing software for networks—such as Internet protocol multipathing (IPMP) DR for I/O devices and networks
Almost none	Multiple instances of fully redundant systems Clustering software (such as Sun™ Cluster 3.0)

1. Remember, redundant power helps only if your site is equipped to supply it.

Note – Even though you can use DR to replace failed components, a critical component failure on a running system (such as a failed CPU) will still cause the system to crash. If you cannot afford this type of downtime, you fit in the *almost none* category, and should use a clustering product to guard against system failures.

For most organizations, the *little* downtime category is a good cost/benefit tradeoff. You will have a system that is resilient to failures and, if properly configured, relatively easy to service. You can use DR to add more CPU/Memory boards for increased capacity, or to replace failed components.

Make a note of what category your system fits into, as well as the additional components you will need. You are going to use this in the next chapter to design your system. You will also use it later in the book during the discussion on configuring the system to integrate with your site.

Finally, some closing words on RAS. It is very important that you do not sacrifice parts of your required configuration for additional RAS features. For example, do not decide to buy less memory so that you can afford additional fan trays. You should ensure that your base requirements are met, or else you will not benefit from additional RAS because your system will have fundamental shortcomings.Disk Redundancy and RAID Basics

To ensure the integrity of the data, some type of disk redundancy should be used on any system with important local data storage. The different schemes for achieving such redundancy are often denoted by their *RAID level*. The term RAID comes from Redundant Array of Inexpensive Disks, and there are numbers from 0 all the way up through 53 denoting different ways of laying out sets of disks.

For most applications, however, only three RAID levels are useful: 0, 1, and 5. Each of these allow you to combine multiple physical disks into a single logical volume. The operating system then sees this volume just like a normal disk, and it can be mounted and used in the regular manner.

RAID 0

RAID 0, commonly called *striping*, provides no additional data safety. Instead, it is designed to increase the speed of file system access. With striping, disks in a volume are interleaved at a certain data interval, called the *stripe unit size*. This means that when reading or writing data, multiple disks are accessed in parallel, decreasing the amount of time it takes to access the data. Striping is very common on any system that needs fast data access, such as database servers.

RAID 1

RAID 1, also referred to as *mirroring*, is just the reverse. It provides full data redundancy, but with some performance costs. In mirroring, twice the number of disks are used for the data that needs to be stored. These disks are then arranged in pairs, and identical data is stored on both disks. On a file system write, two physical writes must be performed, one to each disk of the pair. The advantage is you now have two complete copies of your data.

This means you can lose half of your disks and still continue running without data loss. In a large volume, this is obviously an advantage.

RAID 0+1

RAID 0+1, usually called *striping and mirroring*, is a combination of these two techniques. In a striped / mirrored volume, a set of disks is striped together to form each half. Then, these two halves are mirrored to one another. It is possible to design a striped/mirrored volume so that the performance is better than the individual disks (due to striping), and that fully half the disks can fail without impacting the volume (due to mirroring). This technique is widely-used in production systems.

RAID 1+0

RAID 1+0 is very similar to RAID 0+1, except the volumes are assembled in the reverse order. Here, pairs of disks are mirrored to one another, and then these mirrored pairs are striped together. Volumes created in this manner are slightly more complicated to manage, but are slightly more reliable because of the ways in which disks typically fail. Generally, vendors decide to implement either RAID 0+1 or RAID 1+0, but not both, so the choice of which to use is often made for you.

RAID 5

Finally, RAID 5 is one of the most economical forms of redundancy. In this scheme, a portion of each disk in a volume is used to hold parity. On a write, data is distributed across all the disks in the volume except one, with the parity being written to the remaining disk. This process is repeated in a "round robin" fashion, so that each write places the parity for that write on a different disk. In the event of a single disk failure, the parity is used to recreate data that was on the failed disk. This allows you to lose a single disk (the most common type of failure) and continue running without interruption. RAID 5 is somewhat slow, though, since it must perform all those additional writes for the parity.

While RAID 5 is not as reliable as RAID 0+1 (striping and mirroring), it can still be a good solution, especially for NFS servers. While you can only lose one disk, it is uncommon to lose a whole enclosure barring human error or a power failure, both of which will probably affect much more than your disks. To make use of RAID 5, you should consider only those enclosures that support hardware RAID, since otherwise it is too slow for many applications.

Once you have selected what type of RAID you wish to use for each of your different volumes, you should adjust your storage purchase accordingly. For example, if you want to mirror a set of data, you must purchase double the amount of disk you calculated above. You will need to make sure to increase your controller cards as well.

With RAID 5, check the enclosure you are considering purchasing to verify that it supports hardware RAID.

A Logical Design Specification

By now you should have available all of the information you need to create a logical design specification:

- Design rules of thumb
- Your existing system analysis (if applicable)
- Your RAS design requirements

As you did in the Statement of Requirements Worksheet in Chapter 2, formalize this information into a design for your logical system that will serve as an accurate picture of your needs, so you can use it in the next chapter to choose the appropriate physical system. List your specifications in TABLE 1-10.

TABLE 1-10 Logical Design Specification Worksheet

Item	Description

TABLE 1-10 Logical Design Specification Worksheet

Item	Description