Chapter 7

Finding and Removing Bottlenecks

So far, this book has focused on up-front design and configuration of an email server. The goal of this chapter is to provide some methodology and explain the use of tools that will assist the email administrator in determining the cause of poor email server performance and rectifying the situation.

Over the years, operating systems have grown increasingly sophisticated. Today, several levels of data caches are typically internal to the operating system kernel. Most internal data structures are dynamically sized and hashed, so that they no longer have fixed extents and table lookups remain rapid as the amount of data they contain grows. This increased sophistication generally has been for the best, as these additional variables ensure that operating systems require less manual intervention to get them to work well in a high-performance capacity. Nevertheless, a system administrator needs to be aware of two facts. First, these benefits can have side effects. Second, this extra tuning often makes troubleshooting more difficult.

7.1 Kernel Parameters Run Amok

Let's consider a real-life example. The Solaris operating system from Sun Microsystems contains a kernel table called the Directory Name Lookup Cache (DNLC). The DNLC is a kernel cache that matches the name of a recently accessed file with its vnode (a virtual inode, an extra level of abstraction that makes writing interfaces to filesystems easier and more portable) if the file name isn't too long. Keeping this table in memory means that if a file is opened once by a process, and then opened again within a short period of time, the second `open()` won't require a directory lookup to retrieve the file's inode. If many of the `open()`s performed by the system operate on the same files over and over, this strategy could yield a significant performance win.

The DNLC table has a fixed size to make sure that it consumes a reasonable amount of memory. If the table is full and a new file is opened, this file's information is added to the DNLC and an older, less recently used entry in the table is removed to make space for the new data. The size of this table can be set manually using the `ncsize` variable in the `/etc/system` file; otherwise, it's derived from `MAXUSERS`, a general sizing parameter used for most tables on the system, and a variable called `max_nprocs`, which governs the total number of processes that can run simultaneously on the system. In Solaris version 2.5.1, the equation used to determine `ncsize` was

```
ncsize = (max_nprocs + 16 + MAXUSERS) + 64
```

In Solaris 2.6, this calculation changed to

```
ncsize = 4 * (max_nprocs + MAXUSERS) + 320
```

In Solaris 2.5.1, unless manually set, `max_nprocs = 10 + 16 * MAXUSERS`. I do not know if this calculation changed in Solaris 2.6.

If MAXUSERS is set to 2048, which is typical for large servers running very large numbers of processes, the DNLC on Solaris 2.5.1 would have 34,906 entries. On Solaris 2.6, using the same kernel tuning parameters, the DNLC could contain 139,624 entries. In Solaris 8, the calculation of this parameter had been changed to be more similar to the Solaris 2.5.1 method.

Performance on the new Solaris 2.6 system was horrible. File deletions on Network File Systems (NFS) took a very long time to complete, and it required a great deal of time to diagnose the problem. As it turns out, for some reason that I still don't fully understand, if one attempts to delete a file over NFS, and the DNLC is completely full, the operating system makes a linear traversal of the table to find the appropriate entry. The more entries the table holds, the longer this traversal takes. If it has nearly 140,000 entries, this operation can take considerable time. With the same `/etc/system` parameters on similar hardware running Solaris 2.5.1, these lookups did not cause a noticeable problem.

In my case, a colleague who had encountered this problem before suggested setting `ncsize` explicitly to a more moderate value (we chose 8192) in the `/etc/system` file. We then rebooted the system, and performance improved dramatically.

This is a pretty exotic example, but it indicates the following points:

1. Today's operating systems are complex. It's always possible that a server slowdown might occur under certain circumstances because of the misbehavior of some obscure part of the operating system.

2. It's always possible that very slight changes in an operating system—even changing minor version numbers or adding a single kernel patch—can have far-reaching consequences.
3. Larger cache sizes are not universally a good thing, especially if the system designers do not fully explore the consequences of having large caches.
4. There's no such thing as having too much knowledge about the hardware and software system that a site uses to operate a high-performance server.

This also leads to some obvious conclusions:

1. On a high-performance or mission-critical server, make even small changes with extreme caution.
2. One can never have too much expertise. Fancy tools rarely solve the problem, but a tool that can deliver one good insight during a crisis can prove exceedingly valuable.
3. Trust the data. When something is going wrong, the reasons may not be evident, but they will be consistent within their own logic. Although it may not seem obvious at times, once all the factors are understood, it will be apparent that computers are deterministic objects.
4. Test, test, and test again before deploying production system. Make the test as close to "real life" as possible. There's never enough time to be as thorough as one would like, but if there isn't enough time to test at least some extreme cases, there isn't enough time to do it right. In the preceding situation, we did test, but with data set sizes less than 140,000 files (the size of the DNLC). Because we never filled the DNLC, we never tripped over the bug. Obviously, our testing wasn't "close enough" to "real life." Chapter 8 will explore testing issues in more detail.

This example was not intended to criticize Solaris. Probably every operating system vendor makes comparable changes from release to release, and many similar stories could have been told that focus on other vendors.

There isn't enough space in this book to cover general troubleshooting methodology, but one aspect should be mentioned because it causes many people difficulty. In trying to focus on a problem, the troubleshooter often assembles a great deal of data. Some of it is relevant to the problem at hand, and some of it is tangential. Determining which data are relevant and which aren't often poses the most difficult aspect of solving a problem. There's no magic to categorizing data in this way;

rather, experience and instinct take over. However, when faced with a problem that isn't easily solved, it's often helpful to ask, "What would I think of this problem if any one of the facts involved was removed from the equation?" If one arrives at a conclusion that can be tested, it is often worthwhile to do so, or at least to reexamine the datum in question to make sure it is valid. This sort of analysis is difficult to do well, but in very difficult situations, it can prove a fruitful line of attack.

Another troubleshooting technique is preventive in nature—baselining the system. To understand what's going wrong when the system behaves poorly, it is crucial to understand how the server should behave when the system performs correctly. One cannot overemphasize this point. On a performance-critical server, an administrator should record data using each diagnostic tool that might be employed during a crisis when the server is in the following states:

- Idle, with services running but unused
- Moderately loaded
- Heavily loaded, but providing an adequate response

Then, when the server begins to perform badly, one can determine what has changed on the system. "What is different about the overloaded system from the state where it is heavily loaded, but providing quality service?" This is a much easier question to answer than the more abstract, "Why is this server performing poorly?"

The complexity of today's operating systems exacerbates this need. On most contemporary operating systems, it's much more difficult to tell the difference, for example, between normal memory paging activity and desperation swapping. It's difficult to know objectively what a reasonable percentage of output packet errors on a network interface would be. It's difficult to tell objectively how many `mail.local` processes should be sleeping, waiting for such esoterica as an `nc_rele_lock` event to wake them up. As with people, on computer systems many forms of unusual behavior can be measured only in relative terms. Without a baseline, this identification can't happen.

Previously, I mentioned how important it is to distinguish information related to a present problem from incidental information. Without a baseline, it can be difficult—if not impossible—to tell whether a given piece of information is even out of the ordinary. When something goes wrong, while looking for the source of the problem we've all encountered something unexpected and asked ourselves, "Was this always like that?" Baselining reduces the number of times this uncertainty will arise in a crisis, which should lead to faster problem resolution.

Run baseline tests periodically and compare their results against previous test runs. Going the extra mile and performing a more formal trend analysis can prove very valuable, too. It offers two benefits. First, it enables one to spot situations that slowly are evolving into problems before they become noticeable. Of course, not all changes represent problems waiting to happen, but trend analysis can also spot secular changes in the way a server operates, which may indicate new patterns in user behavior or changes in Internet operation.

Second, formal trend analysis allows administrators to become more familiar with the servers they are charged with maintaining, which is unequivocally a good thing. More familiarity means problems are spotted sooner and resolved more quickly. System administrators responsible for maintaining high-performance, critical servers who do not have time to perform these tasks are overburdened. In this case, when something fails not only will they be unprepared to deal with the crisis, but other important tasks will go unfulfilled elsewhere as a consequence.

In the "old days," many guru-level system administrators could tell how, or even what, the systems in their charge were running by looking at the lights blink or listening to the disks spin or heads move. They could *feel* what was happening in the box. Today's trend toward less obtrusive and quieter hardware has been part and parcel of the considerable improvements made in hardware reliability. This is a good thing. However, through these hardware changes, as well as the aforementioned increasing operating system complexity and the much larger quantity of boxes for which a system administrator is responsible, we've largely lost this valuable *feel* for the systems we maintain. Now the data on the system state are likely the only window we have into the operational characteristics of these servers. It should be considered an investment to periodically get acquainted with the machines we maintain so as to increase the chance of finding problems before they become readily apparent, and to give us the insight necessary to reduce the time to repair catastrophic problems when they do occur.

7.2 The Quick Fix

Despite our preventive measures, let us suppose a server does get itself into a jam and email backs up. Further suppose that we know the problem is that the disk on which the queue resides is not fast enough to handle the load. We may already have another disk ready and an outage window scheduled to perform the upgrade when the system will be less busy, perhaps after most people have gone home from work for the day. Once we can take the machine down, we plan to carefully back up the data on the queue disk, verify the backup, add the second disk, stripe the old and new disks together using software RAID, bring the system back up, test it, restore the backed-up data, verify that this restoration went well, restart services, monitor them for a while, and call it a success. All in all, this strategy sounds like a well-reasoned upgrade plan.

The question is, What should we do *now*? The upgrade window may be hours (or perhaps days) away, the system is running slowly at this moment, and users or management may be asking if something can be done in the short term. Sometimes a quick fix is possible. If the server normally serves other functions, perhaps they can be suspended temporarily. With a POP server, perhaps incoming email could be turned off or at least dialed back long enough so that users can read the email they already have. Temporarily turning off lower-priority services is a reasonable reaction to a short-term performance crunch.

In some circumstances, one might be tempted to try to make short-term alterations to the server to get through the crisis. One could attempt to move older messages out of the queue and into another queue to expedite processing of the main queue. One could lower the `RefuseLA` parameter in the `sendmail.cf` file to try to lower the load on the system. Many other things could be attempted as well. In reality, these attempts at short-term fixes rarely help. Usually, it's best to just let the server work its way out of a jam.

Some assistance, such as rotating the queue or perhaps changing the queue sort order to be less resource intensive, can prove beneficial, but most of the other problems won't be mitigated by just stirring the pot. For example, if one wants to move messages from one queue to another queue on a different disk, what operations must happen on the busy disk? The files will be located, read, and then unlinked. This is exactly the same load that will be put on the disk if the message is delivered. If the message will be delivered on the next attempt, we gain *nothing* by trying to move it. If the message will not be delivered for a while, we can lower the total number of operations on the disk by rotating the queues, and then suspending or reducing the processing of the old queue temporarily. Performing a queue rotation requires far fewer disk operations while deferring or reducing the number of attempts that will be made to deliver queued messages.

Similarly, attempting to reduce the number of processes, the maximum load average on the system, or otherwise trying to choke off one resource in order to reduce the load on another typically arises from a spurious assumption. One may be able to reduce the load on the queue disks, for example, by reducing the number of `sendmail` processes that run on a server. However, reducing the load on the disk doesn't solve the problem, because the load on the disk is a *symptom* of the problem. The real problem is that more email is coming in than the server can handle. In this case, having a saturated disk is a *good* thing. It means that the disk is processing data as fast as it can. If we lower the amount of data it processes, the server will process less email. The external demand on the busy server will not decline because of our actions, but rather increase because we have voluntarily decreased the server's ability to process data, which is the last thing that we want to do.

Some administrators might voice concerns that a system under saturation load will run less efficiently and, therefore, process fewer messages per unit time than a less busy server does. With some types of systems, this concern is well founded, but two points make this possibility less of an issue for the sorts of email systems discussed in this book than for other types of systems.

The first point is that while there exist a large number of fixed resource pools on an email server (CPU, memory, disk I/O, and so on), each process on that server remains largely independent. Thus one process running slowly generally does not cause another process to run slowly, other than through the side effect that both may compete for a slice of the same fixed resource pie. For example, if a system is running so slowly that the process table fills and the master `sendmail` daemon can't fork off a new process to handle a new incoming connection, this issue doesn't cause a currently running `sendmail` process to stop working. These events are largely independent of one another. It doesn't matter to a remote email server whether an SMTP session with the busy server couldn't be established because the master daemon cannot fork a child process or whether the connection is rejected by policy to avoid loading the server.

On the other hand, if, for example, `sendmail` were a multithreaded process running on the server with one thread handling each connection, and it didn't have internal protection against running out of memory, then the process running out of available memory could affect any or all other `sendmail` threads of execution, which could have catastrophic consequences. Fortunately, today's UNIX versions do a remarkably good job of isolating the effects of one process on another. If yet another process is competing for a fixed resource, that conflict may cause the other processes using that resource to run more slowly, but the resource will almost always be allocated fairly and *the total throughput of the system will stay roughly constant*, which is what we really want to happen.

The second point is that even if the system is processing less total data in saturation than it would under a more carefully controlled load, maneuvering the system to achieve higher throughput is very tricky. If some threshold, such as `MaxDaemonChildren`, is lowered too little, it will have no effect on the system's total throughput. If it is lowered too much, resources will go unused, which will lower aggregate throughput—a disaster. The sweet spot between these two extremes is often very narrow, hard to find, and, worst of all, time dependent. That is, the right value for `MaxDaemonChildren` might differ depending on whether a queue runner has just started, how large the messages currently being processed are, or how user behavior contributes to the total server load.

In summary:

1. When a server gets busy, the most important thing is to find the real cause of the problem and schedule a permanent fix for it at the earliest convenient moment.
2. In the meantime, one might be able to do some things to help out in the short term, such as temporarily diverting resources away from lower-priority tasks.
3. Making configuration changes to overcome short-term problems is difficult at best, and will often cause the total amount of data processed by the server to go down, not up, which is not desirable.
4. Because an email server generally allocates limited resources fairly, even when saturated with requests, the best course of action is often to let the server regulate its own resources, as it will likely do so more efficiently than it would with human intervention.

When a real fix for a saturated email server can't be implemented immediately, it's usually better to let the server stay saturated and, hopefully, work its way out of a jam rather than to try to interfere.

7.3 Tools

In this section, we examine just a few of the tools that are likely to be useful to the email administrator. Many possibilities are available—many more than are listed here. Some are very specific, whereas others have broad applications. The tools discussed here are both generally useful and widely available. Email administrators interested in tuning, or just understanding, an email system should not restrict their studies to just the utilities mentioned here. Magazine articles, books, Web sites, and other system administrators can all provide insight into very helpful tools.

Each tool discussed has different options and displays slightly different information on each operating system version. While this inconsistency is annoying, some of the differences are tied to the internal workings of the operating system and are unavoidable. Also, some of the less common options are the most useful. It's just not practical to limit the use of these utilities to their common flag and output subsets, so that won't be done here. Instead, this section will generally provide examples using the FreeBSD (version 4.5) operating system utilities, throwing in some examples specific to other operating systems.

A final note: As we know from science, it is impossible to measure a system without affecting it. Just by running a tool we necessarily change the behavior of the very computer we're monitoring. These utilities consume memory and CPU time, they open sockets and files, and they read data off disks. Therefore, we can never be entirely sure that a problem that we observe on a system isn't at least partially influenced by the fact that we're monitoring it. Although this is rarely the case, it's a good idea to not go overboard by continually running `top` or by having scripts run `ps` every five seconds to capture the state of the machine. A much more modest approach to capturing data (running `ps` every five minutes, for example) will provide equally useful information without adding substantially to the server's load.

7.3.1 `ps`

The venerable `ps` utility comes in two flavors: the Berkeley flavor (found on BSD-based systems and Linux) and the System V flavor (found on AIX, HP-UX, and other systems). Solaris provides the System V flavor in `/usr/bin`, and the Berkeley flavor appears in `/usr/ucb`. My preference is for the Berkeley-style output of `ps`; I like the information it provides and the way that the Berkeley `ps -u` sorts the data. Essentially the same information is available from either version, however, so other than remembering which option does what, one shouldn't be handicapped by any particular flavor.

A lot of information is available from `ps`, and it's especially useful for such tasks as tracking the number of certain types of processes running on a machine or seeing which processes are the largest resource consumers. A great deal of information is available from this program, which varies depending on the option flags selected. Everyone performing system troubleshooting would be well advised to become very familiar with the `ps` man page for the operating system that runs on their email server.

For both varieties of `ps`, some command-line flags require more processing to resolve than others. On Berkeley-type systems, it is more computationally intensive to resolve commands with the `-u` flag than without it. For System V versions, adding the `-l` flag requires more computational resources than if the command is run without it. Therefore, these flags, which produce extra output, should be used only when they relate important information. One thing that `ps` provides is rough process counts, for example:

```
ps -acx | grep -c "sendmail"
```

These sorts of data are useful, and periodic counts are often scripted. Especially in automated systems, it's worthwhile to make sure that they produce minimal strain on the server. Determining which options are more resource intensive than others isn't always straightforward, but the `time` command or shell built-in can aid in this calculation. On quiet servers, the response time for this command might be too fast to measure, so the aggregation of several commands may provide a more precise measurement. For example:

```
/usr/bin/time sh -c ‘for i in 1 2 3 4 5 6 7 8 9 10; \
    do ps -aux > /dev/null; done‘
```

Some `ps` output from the CPU-bound test server during one of the tests cited earlier in this book appears in Table 7.1. At the moment that this snapshot was taken, `syslogd` was the most active process. While it is a busy process on an email server, it rarely does the most work at any given time. However, unlike the MTA and LDA processes that move data, this persistent process reads data from the IP stack and writes it to disk on every delivery attempt.

Adding all numbers in the RSS column, they roughly equal the system's total main memory (only 32MB), which doesn't count RAM consumed by the kernel or the buffer cache. Because much of the memory consumed by the processes is shared, it provides enough space to keep the parts of the programs that run while resident in memory and still allow extra space for the kernel and the buffer cache.

On this machine, the `script` command is used to capture output from the `iostat` and `vmstat` commands, which will be discussed shortly. The `stat` entry is a home-built script that adds date and time information to the output of these two utilities. As we'd expect, most of the CPU time is consumed by `sendmail` and `mail.local` processes. Also as we'd expect, concurrent MTA processes outnumber LDA processes, even though the email is sent to this server over a low-latency local area network.

Most of the rest of the processes running on this server are either standard parts of the operating system or processes related to remote connections to the server.

7.3.2 `top`

Many UNIX operating systems include the venerable `top` utility, which is also one of the first Open Source programs installed on many other operating systems. The `top` utility lists the largest CPU resource consumers on a system and updates this list periodically, typically every few seconds. For understanding the general state of the system, some of the most valuable information appears in the first few lines of the program's display. A system consistently showing a CPU idle state at or near 0% is

Table 7.1. Sample `/usr/ucb/ps -uaxc` Output from the CPU-Bound Test Server

```
% /usr/ucb/ps -uaxc
USER         PID %CPU %MEM   SZ  RSS TT      S    START  TIME COMMAND
root       11302  1.3  3.3 3480 1004 ?       S 15:03:18  0:23 syslogd
root       23420  1.1  4.3 2296 1304 ?       R 16:05:56  0:02 sendmail
root       24881  0.9  5.6 2392 1700 ?       S 16:13:11  0:00 sendmail
root       24884  0.8  5.3 2352 1592 ?       R 16:13:11  0:00 sendmail
root       24861  0.7  5.6 2392 1700 ?       S 16:13:07  0:00 sendmail
root       11009  0.6  3.9 2012 1172 ?       S 14:47:30  0:08 nscd
root       24871  0.6  3.4 1552 1016 ?       S 16:13:08  0:00 mail.local
root       24886  0.5  5.0 2312 1516 ?       R 16:13:12  0:00 sendmail
root       24892  0.5  2.9 1140  860 pts/4   O 16:13:13  0:00 ps
test1      24890  0.5  3.4 1552 1012 ?       R 16:13:12  0:00 mail.local
root       24889  0.4  5.0 2312 1516 ?       R 16:13:12  0:00 sendmail
root       18650  0.3  3.8 1712 1160 ?       S 15:45:14  0:01 sshd
npc        18716  0.2  2.5 1016  756 pts/4   S 15:45:24  0:00 csh
root       24891  0.2  1.7 2296  492 ?       S 16:13:12  0:00 sendmail
npc        23454  0.2  1.9  856  580 pts/3   S 16:08:02  0:00 stat
root       23449  0.1  1.9  856  580 pts/2   S 16:08:00  0:00 stat
npc        23434  0.1  1.9  788  560 pts/0   S 16:06:48  0:00 script
root           3  0.0  0.0    0    0 ?       S   Feb 04 19:27 fsflush
root           0  0.0  0.0    0    0 ?       T   Feb 04  0:00 sched
root           1  0.0  0.5  652  132 ?       S   Feb 04  0:26 init
root           2  0.0  0.0    0    0 ?       S   Feb 04  0:02 pageout
root         156  0.0  1.8 1464  548 ?       S   Feb 04  0:01 cron
root         159  0.0  2.4 1644  724 ?       S   Feb 04  1:46 sshd
root         174  0.0  1.6  852  480 ?       S   Feb 04  0:00 utmpd
root         203  0.0  2.1 1404  632 ?       S   Feb 04  0:00 sac
root         204  0.0  2.1 1496  624 console S   Feb 04  0:00 ttymon
root         206  0.0  2.3 1496  688 ?       S   Feb 04  0:00 ttymon
root       10540  0.0  3.1 1800  936 ?       S 14:19:23  0:10 sshd
npc        10543  0.0  1.5 1012  448 pts/1   S 14:19:33  0:00 csh
root       10554  0.0  0.0  276    4 pts/1   S 14:20:03  0:00 sh
root       11262  0.0  3.0 1712  904 ?       S 15:01:22  0:02 sshd
npc        11265  0.0  2.5 1028  756 pts/0   S 15:01:25  0:00 csh
root       11456  0.0  2.7 1052  800 pts/1   S 15:05:33  0:00 csh
root       23429  0.0  1.8  764  536 pts/1   S 16:06:46  0:00 script
root       23430  0.0  1.9  788  560 pts/1   S 16:06:46  0:00 script
root       23431  0.0  2.4  996  732 pts/2   S 16:06:46  0:00 csh
npc        23433  0.0  1.8  764  536 pts/0   S 16:06:48  0:00 script
npc        23435  0.0  2.7 1024  804 pts/3   S 16:06:48  0:00 csh
root       23448  0.0  2.2  840  660 pts/2   S 16:08:00  0:00 vmstat
npc        23453  0.0  2.3  848  684 pts/3   S 16:08:02  0:00 iostat
```

almost certainly CPU bound. The caveat is that some systems list an `iowait` state indicating what percentage of processes are waiting for I/O. This number doesn't represent CPU time being consumed, but rather consists of the system's best guess as to the amount of CPU time that would be consumed if no processes were blocked waiting for I/O. If a significant percentage of processes are in the `iowait` state, then the system may show 0% idle while the CPU is barely being used.

In the upper-left corner is the last process identifier (PID) used by the system. From its rate of change, one can deduce how many new processes are spawned per second, giving some idea of how fast sessions are coming and going on the server. This method isn't useful on those few operating systems, such as OpenBSD, that assign new PIDs randomly rather than sequentially.

The memory information displayed isn't as useful as one would first expect. On nearly any system that has been running for a few minutes, or even a few seconds if it's busy, we should expect the amount of free memory listed to stay very near zero. On contemporary operating systems, any RAM that goes unused by processes will be allocated to caching some data. Thus, just because there is very little memory free, it doesn't mean that the system is memory starved. On some operating systems, `top` will show more memory information, such as how much RAM is allocated to filesystem caches; if this number drops near zero, it would likely indicate that the server would use additional RAM effectively.

Even more so than with `ps`, the information displayed via `top` varies from operating system to operating system. A thorough reading of the utility's man page should be performed before its results are interpreted.

7.3.3 `vmstat`

The `vmstat` utility explores the activity of the virtual memory system, which includes real memory used by processes, memory used for caching, and swap space. The first line of data produced summarizes the activity since the system was booted. Generally, this information should be ignored.

While it's much less impressive than the output that one will find on a true high-performance email server, some example output from the CPU-bound server during one of the test cases discussed in this book can be instructive. This output appears in Table 7.2.

Excessive memory activity will cause heavy paging, which translates into relatively large numbers in the `pi` and `po` columns. Of course, what constitutes a large number depends heavily on the particular system. Interpreting these numbers without a baseline will be next to impossible. In the example case, these numbers are so small that we can safely conclude that the system is not memory bound.

Table 7.2. Sample `vmstat 15` Output from the CPU-Bound Test Server

```
% vmstat 15
 r  b  w  swap   free  re    mf  pi  po   fr  de  sr   in    sy   cs  us  sy   id
 0  0  0   4692   1804  0     0   0   0    0   0   0    3    39   22   0   0  100
 7  1  0  64440   1956  5   925   6  38   38   0   0  256  2150  328  31  68    2
 8  0  0  64008   1800  8   923   0  40   44   0   1  249  2030  285  24  67    9
 8  0  0  64612   2276 10   950   1  46   48   0   0  249  2079  283  27  66    7
 7  1  0  64712   2956  4   954   6  23   94   0  22  262  2101  294  28  67    5
 7  1  0  64320   2852  0  1024   2   0    0   0   0  271  2260  317  27  73    0
 9  0  0  61684   1960  8   995   6  62  170   0  37  281  2186  329  29  71    0
 7  0  0  62968   3836  0  1061   4   0    0   0   0  255  2209  315  29  71    0
15  1  0  58508   1800 12   956   7  71  138   0  27  288  2302  342  30  70    0
 6  0  0  62936   4860  2  1035   1  10   10   0   0  252  2072  299  26  71    2
```

On those systems whose `vmstat` provides this information, another column worth tracking is `de`. It gives a system's expected short-term memory deficiency, for which memory space will have to be actively reclaimed. A nonzero entry will show up occasionally in this column on a healthy but busy system. The more often this result appears, though, the more likely the system could use more memory. Our sample data show no deficiencies, another indication that this system is not memory bound.

The first column, labeled `r`, indicates the number of runnable processes, which provides a snapshot of the system load average. In this example, a number of processes want to run but can't because they have no CPU time slice available to them. The second column, labeled `b`, gives the number of processes that are blocked from proceeding because they are waiting for I/O. If a significant number of processes are listed in this column, the system is likely I/O bound. In our example, we occasionally see a blocked process, but this event is rare, giving us an indication that this system isn't I/O bound. Yet one more variable worth tracking is the third column, labeled `w`. It represents the number of processes that are either runnable or have been idle for a short period of time and have now been swapped out. Frequent nonzero numbers in this column also indicate that the server may be desperately short of RAM. The example looks like it's in good shape on that point.

In the past, one could tell whether a system was memory starved just by looking for swapping activity, as opposed to the more healthy activity of paging. Paging is the process of writing parts of process data to swap space to make room for pages of other data in active memory. An operating system may "page out" part of a process if that page hasn't been accessed in a while, even if the process is running. This efficient behavior allows new processes to start up more quickly because memory reclamations don't need to occur first, and it leaves more room for caching data, leading to better performance. Some amount of paging will occur on all operating systems and is considered normal and healthy.

Swapping usually refers to taking a process and moving its entire memory image to disk. It might happen if the process has remained idle for a very long time (tens of seconds, which is a very long time in computer terms) or if the system desperately needs to make room for new processes. "Desperation swapping" and "thrashing" are terms used to describe a system that is so memory starved that nearly every time a process receives a CPU slice, it must be read in from swap to active memory before it can proceed. This horrible circumstance effectively slows memory access (typically measured in tens of nanoseconds) to disk speeds (measured in ones to tens to hundreds of milliseconds, a difference of two to four orders of magnitude). Once a system starts thrashing, it will not operate efficiently. One should aggressively avoid this situation.

Somewhat unfortunately, as virtual memory algorithms have become more complex and sophisticated over the years, it's become more difficult to tell in a vacuum whether a system is thrashing. In fact, many operating systems don't distinguish between paging and swapping, eliminating the latter behavior altogether. Here is where a baseline becomes crucial. One must understand what sort of paging statistics occur on a heavily loaded but properly operating server before one can determine whether a system is beginning to thrash. However, once the disks with swap on them begin to get loaded, it will be painfully obvious that the system has simply run out of memory. Of course, this behavior will occur beyond the point where a server starts to slow down noticeably.

Solaris 8 introduced a new system for managing the buffer cache. Now the page daemon is no longer needed to free up memory used to cache filesystem information. Consequently, the page daemon does not have to do any work to reclaim memory space for new processes. The upshot is that on Solaris 8, if the `sr` field of `vmstat` output is nonzero, running processes are being paged to disk to make room for new processes. On this operating system, it has now become more straightforward to identify significant memory deficiencies. Significant activity in the `sr` field on other operating systems can indicate that the machine is memory starved, but the demarkation point is not as obvious as it is on Solaris 8.

7.3.4 `iostat`

The `iostat` tool is similar to `vmstat`, except that it measures system I/O rather than virtual memory statistics. On many systems, it can measure not only disk-by-disk data transfers, but also I/O information to and from a wide variety of sources,

Table 7.3. Sample `iostat -cx 15` Output from the CPU-Bound Test Server

```
% iostat -cx 15
                              extended device statistics             cpu
device  r/s   w/s  kr/s   kw/s  wait  actv  svc_t  %w  %b  us  sy  wt   id
sd0     0.0   0.6   0.1    3.0   0.0   0.0    9.2   0   1   0   0   0  100
sd3     0.0   0.0   0.0    0.2   0.0   0.0   22.8   0   0
                              extended device statistics             cpu
device  r/s   w/s  kr/s   kw/s  wait  actv  svc_t  %w  %b  us  sy  wt   id
sd0     0.5  12.0   1.7   62.0   0.0   0.1   10.9   0  12  27  68   1    4
sd3     0.0  47.9   0.0  225.8   0.0   0.4    7.4   0  30
                              extended device statistics             cpu
device  r/s   w/s  kr/s   kw/s  wait  actv  svc_t  %w  %b  us  sy  wt   id
sd0     0.7  10.7   1.9   54.3   0.0   0.1   11.0   0  11  29  71   0    0
sd3     0.2  51.2   0.4  251.9   0.0   0.5   10.0   0  34
                              extended device statistics             cpu
device  r/s   w/s  kr/s   kw/s  wait  actv  svc_t  %w  %b  us  sy  wt   id
sd0     0.7  13.9   2.4   71.1   0.0   0.2   11.1   0  14  29  71   0    0
sd3     0.9  47.6   1.3  235.7   0.0   0.4    9.2   0  33
                              extended device statistics             cpu
device  r/s   w/s  kr/s   kw/s  wait  actv  svc_t  %w  %b  us  sy  wt   id
sd0     0.7   9.8   2.3   51.3   0.0   0.1   10.2   0  10  30  70   0    0
sd3     1.0  54.0   1.8  268.9   0.0   0.5    8.3   0  36
```

including tape drives, printers, scanners, ttys, and so on. Like `vmstat`, this command displays CPU information in the last set of columns. On many systems, if one specifies no I/O devices, it can be a good mechanism to track CPU usage in scripts, such as running `iostat -c 60` to get basic output of CPU information every minute on a Linux or Solaris system. As with `vmstat`, the first line of output by the `iostat` program is a summary since boot time and is effectively useless. Table 7.3 gives some data gathered with `iostat` while testing earlier examples in this book.

Typically, `iostat` reports its data as kilobytes per second or transfers per second. In this example, reads and writes per second for each device are listed in the second and third columns, while the amount of data being moved appears in the fourth and fifth columns. Some versions also show how long the average transfer takes, `svc_t` in this example, which can be very useful metric for determining loading. If this number starts going up, it indicates that the device is heavily loaded.

On Solaris and some recent versions of Linux, the `-x` flag gives even more valuable information, as in this example, including the average amount of time each request spends in the wait queue and the percentage of time I/O requests are waiting to be serviced by the disk device. These numbers represent some of the best indicators of disk contention in the absence of a baseline, but they're no substitute for one. A disk can be 100% busy and yet the system can still provide adequate service. In our example, we can clearly see that the two disk devices (`sd0` contains the message store and `sd3` contains the logs and the email queue) are not saturated and, therefore, this system is not I/O bound.

Knowing that a disk always has requests sitting in the wait queue doesn't explain why a change in server behavior has occurred. If kilobytes per second increases while tps remains constant, it would indicate that we're dealing with larger requests, which may alert us to a temporary or permanent change in the type of email flowing through the system.

On some operating systems, `iostat` has problems reporting useful information about disks managed by software RAID or from a hardware RAID system. This is especially true for those numbers indicated on a percentage basis. Absolute throughput numbers such as numbers of reads and writes per second or bytes per second compared against a baseline are likely to be more reliable. Because email servers so often become I/O bound, `iostat` may be the single most important utility in the email administrator's toolkit. Anyone who expects to maintain such a system would be well advised to become very familiar with it.

In the operating system used in the examples here (Solaris 2.6), note that the CPU loading information given by the `iostat` command lists an I/O wait stat (the `wt` column), whereas the `vmstat` command lumps it in with the idle CPU state (the `id` column). Someone who looked at just the `vmstat` output might conclude that the system is not quite CPU bound, whereas this result would become more obvious if the CPU loading information was examined via `top` or `iostat`.

7.3.5 `netstat`

The third tool in the "`*stat`" trio is `netstat`. As one would expect, `netstat` provides information about system networking. It can display either a snapshot of very detailed information about nearly every conceivable network parameter (`netstat -s`) or periodic data like that found with `vmstat` or `iostat` (e.g., `netstat -w 5` on BSD systems, `netstat -i 5` on Solaris, or `netstat -c` on Linux).

Obviously, in its periodic mode, some of the parameters provided by `netstat` that we want to carefully observe include the number of packets per second and the number of bytes per second. Both statistics, and especially trends in them, can provide the most direct information on the objective external load on a system, so they should be tracked. How the ratio of input to output statistics might change can also be highly informative.

On some types of shared networks, such as Ethernet, when computers are connected to the network via a hub rather than a switch, two machines could potentially try to send a network packet at the same time. This attempt can result in a collision. Both senders will then wait for a small, random amount of time and try to send their packets again. On a shared network, the number of collisions is a good indicator of general network load. Again, hard and fast numbers are difficult to identify, as they depend on the speed of the network, packet sizes, and the number of other machines on the network, but as a rule of thumb a busy email server should not reside on a network that consistently shows hundreds of collisions per second. On a switched network, no collisions should occur. If they do arise, it might mean that the switch, or the connection between the server and the switch, dropped into a nonswitched mode for some period of time. To avoid this possibility, one can lock network interfaces on switched networks into full-duplex, rather than letting them autonegotiate speed and mode.

The other piece of data of special value from `netstat` in periodic mode involves the error rates. An error usually indicates that a packet has failed its checksum—that is, its contents don't match what the packet header indicates. An output error indicates that this problem occurred somewhere between the formation of the packet by the operating system and its transmission over the wire. This result is never good. Even a handful of entries in this field can indicate a serious problem with the server's NIC and should be investigated. Input errors are less severe, as a packet might legitimately have become corrupted traveling over a network to the server, but input error rates of even 0.1% may indicate a network problem, such as bad cabling, electrical interference, or a bad NIC. An error rate of 1% means something is seriously wrong with the network somewhere, and this problem should be tracked down and eliminated before it worsens and interferes with operations.

7.3.6 `sar`

On System V-derived UNIX versions, you can run the System Activity Reporter (`sar`) program in the background to gather statistics and accounting information, including much of the data reported by the tools that have already been mentioned in this section. It is an excellent baselining tool, and collecting data every 1 to 15 minutes on a system via `sar` and archiving those data is something that every server administrator should seriously consider. This effort will be worthwhile on any system where performance monitoring is important.

Just about every piece of data one could want to examine is available via `sar`. In fact, it's more likely that one will miss key information due to the presence of too much data than that information on the nature of a given problem isn't available. This tool provides a superset of the information available from `vmstat`, `iostat`, `netstat`, and other utilities. Any performance-critical server administrator should become very familiar with `sar` and its affiliated utilities.

7.3.7 Other Utilities

Many other utilities could have been mentioned here, such as `pstat`, `lsof`, `ifconfig`, `systat`, `pstack`, ad nauseum. They have been omitted not because they're not valuable, but because a line must be drawn somewhere. Playing around with these other possibilities is worthwhile with the proviso that before one makes a new utility part of the "canon," it should be demonstrated that programs that are more familiar and already on the system cannot easily generate the same information.

Finally, if for no other reason than to satisfy the reader's curiosity, I'll explain the `stat` shell script that appeared on the example `ps` output. This trivial script receives output from commands such as `vmstat` and `iostat` that do not indicate the date and time the data were gathered, and adds this information. Thus, instead of

```
% vmstat 15
 procs         memory ...
 r b w swap  free  re ...
 0 0 0  4692  1804  0 ...
 7 1 0 64440  1956  5 ...
 8 0 0 64008  1800  8 ...
 8 0 0 64612  2276 10 ...
```

we could run

```
% vmstat 15 | /usr/local/etc/stat
020315 15:14:03  procs       memory ...
020315 15:14:03  r b w swap free  re ...
020315 15:14:03  0 0 0  4692 1804  0 ...
020315 15:14:18  7 1 0 64440 1956  5 ...
020315 15:14:33  8 0 0 64008 1800  8 ...
020315 15:14:48  8 0 0 64612 2276 10 ...
```

Now data from one source can be matched up in time against data from another source.

The `stat` script is trivial:

```
#!/bin/sh

OLDIFS=$IFS
IFS=
while read LINE
do
        echo -n ‘date "+%y%m%d %H:%M:%S"’
        echo " " $LINE
done
IFS=$OLDIFS
```

`IFS` is redefined to be null so that the whitespace isn’t adjusted when each line of input is collected by the `read` command.

7.4 syslog

The `syslog` facility isn’t a program used to evaluate system performance, but rather a set of library calls and a daemon, `syslogd`, that records information that the system and its programs think is worth logging. All email server applications mentioned in this book, but especially `sendmail`, use `syslog` to log data about their behavior, and it would be unwise for an email administrator to ignore this fact. The `syslog` package was originally developed as a part of the `sendmail` distribution to help maintain the information it would log. It was later adopted by the rest of Berkeley UNIX around the BSD 4.1 timeframe, and from there spread to other UNIX versions. Now it has become so ubiquitous that the fact that its origin is tied to `sendmail` has been largely forgotten.

7.4.1 syslog and sendmail

During an SMTP message reception, `sendmail` logs the sender information at the end of the message transaction, whether the message is actually sent or not. If the message is accepted, then the log entry occurs after the end of the DATA phase. If the message is rejected, then the log entry is made immediately after the rejection. During a successful SMTP message reception, `sendmail` logs the recipient information at the end of the session, although the precise timing can vary depending on `sendmail`’s delivery mode. At the conclusion of each failed delivery attempt, that attempt is also logged. At least two log entries for each successful delivery and one log entry for each unsuccessful attempt, plus various other entries for start-up, encountering errors, `STARTTLS` information, and so on, will be made. The two types of log entries mentioned initially make up the bulk of the log messages generated, however, and are the two that occur as a result of a successful delivery. All of these log messages can result in a lot of data, and this information can provide significant insight into what is happening on the server.

Here is an example `sendmail` log entry for a successful message delivery:

```
Mar 12 14:39:49 discovery sendmail[44639]: g2CMdnkq044639:
 from=<npc@acm.org>, size=1047, class=0, nrcpts=1,
 msgid=<200203122239.g2CMdcQL044635@mail.acm.org>,
 proto=ESMTP, daemon=MTA, relay=mail.acm.org [199.222.69.4]
Mar 12 14:39:49 discovery sendmail[44641]: g2CMdnkq044639:
 to=<npc@gangofone.com>, delay=00:00:00, xdelay=00:00:00,
 mailer=local, pri=30052, dsn=2.0.0, stat=Sent
```

The first question asked might be, “How do we count the total number of messages that flow through the system?” This question isn’t as simple to answer as it might seem. The answer depends on whether one wants to count the number of SMTP connections, the number of unique messages as sent by a sender, or the number of messages (often sent to multiple recipients) that end up in someone’s mailbox somewhere. Each of these metrics is a valid choice, but in my judgment the bulk of the work is done for each successful message recipient, so I generally choose to count the number of `syslog` entries with both the `to=` pattern and `stat=Sent` in them. I call that measure the number of messages that the system has successfully processed, mindful that it is merely one statistic that is much more nebulous than it appears at first glance.

If we consider the format of these log entries to contain a set of fields delimited by whitespace, the first three fields contain information about the date and time when the log entry was made. This information can be parsed to track the busiest time of day for the server. In the `from` entry, the eighth field contains the size of the message in bytes, which we can use to find out the average message size handled by the system. On the same entry, the tenth field lists the number of recipients per message, another interesting statistic to track. In the `to` entry, the information in the `delay` and `xdelay` fields are of particular interest. The `delay` field measures the total amount of elapsed time between the receipt of the message and this particular delivery attempt. The `xdelay` field, which stands for transaction delay, measures the amount of time consumed on this particular delivery attempt, which should reveal something about the current connectivity to a particular site.

A great deal more information available in the logs can be extracted for various purposes, but at this point the next step will be left to the imagination of the reader. Section 2.1.1 of the *Sendmail Installation and Operation Guide* provides additional information on the `sendmail` log entries.

A similar set of information can be extracted from the logs left by any of the POP or IMAP daemons discussed in this book. Combined with other statistical information gathered with the tools described here, one can plot number of processes versus load average, connection rates versus disk activity, and so on to obtain a thorough understanding of any email server's performance. These checks can be easily automated, and at least the most basic ones should be part of an email administrator's baselining effort.

7.4.2 `syslog` and Performance

If a server handles a large volume of email, the resources consumed by `syslog` in writing out the many log entries can be significant. On very large servers, mounting `/var/log` or its equivalent on its own disk might be appropriate. Beyond this point, an additional `syslog` issue directly affects performance that should be mentioned. On Linux systems, by default the `syslog` daemon will `fsync()` its log files after each entry is written to them. On a busy email server, this operation can cause a measurable slowdown. In most organizations, email server logs aren't so critical. This behavior can be switched off by preceding the appropriate entry in the `/etc/syslog.conf` file with "`-`":

```
mail.* -/var/adm/mail
```

If logging continues to pose a performance problem for a host, it may be appropriate to log the information to a dedicated remote logging host. If this step is taken, replacing the `mail.` entries in `/etc/syslog.conf` with one like the following may be appropriate:

```
mail.* @loghost.example.com
```

The `loghost.example.com` machine may end up aggregating log information for a large number of hosts. Because the host name is included in each log entry, it should be straightforward to split the entries out again on the log host if desired. On the log host, a RAID system with a high-performance filesystem may be mounted on `/var/log` to handle this load.

One downside to remote logging is that `syslog` sends its messages to the log host using UDP. Thus, if a log message becomes lost en route, it will not be retransmitted. This behavior makes this method less useful if saving each log message is critical. One way to work around it is to replace the default `syslog` daemon with `syslog-ng` [SYS] or one of several packages with similar feature sets that support logging over TCP.

7.5 Removing Bottlenecks

Let's assume that the system bottleneck has been revealed using the techniques described earlier in this chapter. The next logical question to ask is, "What should be done about it?" Some of the methods for effecting an improvement in system performance will be obvious, and many have been discussed already. In this section we will explore some of the ways in which bottlenecks may be alleviated and some of the pitfalls that may be encountered.

It may seem that identifying the bottleneck and planning the fix should be the most difficult part of improving system performance, and they usually are. Additional frustration may arise, however. We never really completely eliminate bottlenecks—we just improve the throughput of one aspect of a system. A truism of information technology seems to be that the load placed on servers increases over time. As the load on an email server grows, it is inevitable that we will eventually encounter another bottleneck that must be removed. Perhaps this next bottleneck lurks just around the corner, raising its ugly head after our capacity increases just a few percentage points from the level at which the last obstacle was removed. If we have a set of disks on a SCSI-2 interface that is saturated at peak times while delivering 8 Mbps of data to our applications, we won't have long to wait after upgrading the disks before the SCSI controller becomes a bottleneck. Sometimes, as in this example, the next hurdle that will need to be overcome is easy to see. At other times, it's almost invisible. With experience comes better instincts about where the next problem lurks, and with some support from the folks who control budgets, perhaps some of these roadblocks can be eliminated before they slow down the system again. No matter how much experience a person has, no one can anticipate everything. This uncertainty is just one of the things that makes the job so challenging.

7.5.1 CPU-Bound Systems

With a CPU-bound system, the first step is to see if anything currently running on the server can be stopped or moved to another server. If that's possible, it would be a fortunate fix. Of course, one cannot eliminate unnecessary tasks indefinitely.

Sometimes a shortage of CPU power really masks another problem—for example, the system may be working very hard to move processes in and out of swap space. The danger in interpreting utilities that seem to report CPU utilization but also report I/O utilization, such as some versions of `top` or `vmstat`, has already been discussed.

By some measures, having a CPU-bound system is a good thing. It usually indicates that the rest of the system is well tuned and operating efficiently. Besides, CPU is often the easiest component to upgrade. Even in the worst-case scenario, we can expect the new chip released in the next quarter to offer a larger percentage improvement over the current product line than for any other computer component of the next-generation system.

Finally, the email applications discussed in this book have all run many processes simultaneously to handle multiple requests. Thus they operate in parallel very nicely to work on multiprocessor computers. If an email server with two processors becomes CPU bound, it's almost certain that the same vendor has an upgrade plan to a four-CPU box, and upgrading to a system with more CPUs is almost always easier to plan and execute than upgrading I/O controllers, software, or storage systems. Not only is there typically less to configure, but also it's straightforward to estimate the actual improvement in CPU capability between the old and new systems. This factor is generally much easier to predict than the effects of upgrading a storage system or increased RAM.

7.5.2 Memory-Bound Systems

As we've already learned, some amount of paging on a system is normal. Excessive paging, or "thrashing," causes problems, however. This condition is not always easy to detect when it is mild, but it is patently obvious when severe. If the system is truly memory bound, the only solution is to add more RAM. Fortunately, memory is relatively cheap when it comes to system costs. For most applications, having extra memory will help reduce I/O by providing more filesystem cache space. Email is helped less than many other applications by surplus RAM, but extra memory does help, sometimes a great deal. Rarely will a recently constructed email system require more memory storage than can fit in that machine. That is, CPU, networking, and I/O all tend to make an entire computer chassis obsolete before it needs to be completely filled with RAM.

Another pitfall may become evident: It's very easy to be fooled into thinking a system problem is a memory problem when that's not the case. In point of fact, *every* time a server runs out of a finite resource, if the load keeps coming, the system will *always* run out of memory. Consider the following scenario:

- A computer is relaying email from the Internet to an internal email server. The internal server can handle anything the gateway can throw at it.
- The gateway's queue resides on a single disk, and just today, the load has reached the point where metadata operation contention exhausts the I/O capability of the disk.
- The server is now processing as much email as it possibly can, but the rest of the Internet won't be sympathetic and back off. Instead, email keeps getting sent, and at a faster rate than data can be moved into and out of the queue. Putting some numbers to this scenario, let us suppose that email comes into the server at a rate of 5 Mbps, but the queue is processed at 4 Mbps.
- Consequently, more `sendmail` processes are spawned on the server than exit in a given time period, causing the number of processes to start to increase.
- While they share a single text image in memory, each process has its own data image that starts eating away at available RAM.
- This shortfall in memory causes the system to reduce the size of the buffer cache, placing more I/O demands on the queue disk that cannot be satisfied. The total number of processes increases further as each process takes longer to complete its work and exit.
- Eventually, real memory pages become exhausted by all of these surplus processes, and the system starts to thrash.

When email administrators come to this machine and start running diagnostics, they will see that the server is out of memory and thrashing. Stopping there, they will erroneously conclude that the system needs more memory. They can obtain more and install it, but next time this situation occurs the server will merely flail around for a longer period before it begins to thrash. Adding memory will not correct the real problem.

In fact, this example leads to a general maxim about Internet servers. *Surplus RAM acts as a buffer against temporary resource shortages. More RAM does not eliminate the problem, but it does buy the server more time in which the shortage might become resolved, or at least be abated.* In our example, the resource shortage was disk I/O, but the same sort of scenario plays out for an email server that communicates directly with other servers around the Internet when the organization's Internet link is severed. Email backs up on the server filling the queue. As the queue grows deeper, the amount of time any one process spends in the queue increases, leading to resource contention that may become apparent to POP or IMAP users. Having more RAM on the server means that a longer outage may be tolerated before intervention becomes necessary.

Similar sorts of outages occur frequently and can be mitigated by good planning and architecture, but cannot be completely eliminated no matter how much effort is expended. DNS server outages, routers being given bad information or rebooting, "backhoe fade," or even unusual transient spikes in load can all cause these sorts of problems. A good server will be resilient against these sorts of situations, but it can never be made impervious to them. For this reason, it's more difficult to provide reliable Internet services than it is to provide many other utility services such as reliable dial-tone phone service. When a phone switch runs out of circuits, it can say "no" to the next entity wanting to use its resources; the process of saying "no" does not significantly drain the switch's resources. This result is much harder to achieve in the Internet case. Even saying "no" takes more resources, and the load will keep coming despite the refusal.

If a server is running normally at full capacity without any problems, but occasionally runs out of resources without having to process either more email messages (transactions) or larger messages (overall volume), then running out of memory is not the cause of the problem, but rather a symptom. The server is running out of "something else," where that something could be network bandwidth, I/O, CPU, or any number of other possibilities. If a server running out of memory is correlated with a higher demand being placed on that machine, that condition may indeed be a memory shortage.

7.5.3 I/O Controller-Bound Systems

On most disk systems, the data may be accessed by only a single I/O controller. If this controller becomes saturated, few remedies exist. Splitting up the load onto multiple storage systems using different controllers is the first thing to try, but that can't happen beyond the limit of one disk/SSD/RAID system per controller. If a controller with one device becomes saturated, the only option is to upgrade to a faster controller. However, this upgrade is a solution only if the storage device on the other end of the bus can support the faster speed. If a SCSI-2 controller is saturated talking to a single SCSI-2 disk, upgrading the controller isn't enough, because the disk will still speak SCSI-2 and the faster controller won't make a difference. In this case, the disk must be upgraded as well.

The use of system NVRAM can help mask controller saturation, but disks and controllers aren't that expensive, so upgrading shouldn't impose any special burden. It's generally a good idea to buy a high-end SSD or RAID system that comes with the highest-speed interface supported by the device, even if one has to buy a new controller to match. Even if the bandwidth isn't needed now, it would be a tragedy to purchase an ultra-fast storage system that must be completely replaced at a later date solely because its controller runs out of bandwidth before the storage system does. Only the very highest-end storage devices (SSDs and the most powerful RAID systems) can saturate the fastest controllers on the market by themselves in typical email environments. If this event comes to pass, the only solution is to divide the load over multiple disk systems on separate controllers, either with multiple mount points or by using software RAID to create a single storage image out of multiple devices by striping them together.

Because email data access patterns tend to be small and random, one can usually place several, or even many, disks on a single controller with confidence. In an environment where only a single large file will be read at a time, two or three disks per controller might be the maximum supportable. For email servers, it's usually safe to put several disk drives on the same SCSI bus—perhaps as many as six on a SCSI-2 chain, or even a dozen on high-speed controllers. It's still a good idea, though, to dedicate controllers to email tasks. Controllers can become saturated on email servers, especially if solid state disks or high-performance storage systems are employed.

7.5.4 Disk-Bound Systems

Conceptually, upgrading disk systems is fairly easy. Get faster disks, get faster controllers, and get more disks. The problem is predicting how much of an improvement one might expect from a given upgrade.

If the system is truly spindle bound, and the load is parallelizable such that adding more disks is practical, this route is almost always the best way to go. When a straightforward upgrade path exists, there's no more likely or predictable way to improve a system's I/O than by increasing the number of disks. The problem is that a straightforward path for this sort of upgrade isn't always obvious. As an example, assume we have one state-of-the-art disk on its own controller storing `sendmail`'s message queue, and the system has recently started to slow down. There are two ways to effectively add a second disk to a `sendmail` system. First, we could add the disk as its own filesystem and use multiple queues to divide the load between the disks. This upgrade will work, but will become more difficult to maintain and

Christenson Christenson-Textures-07 August 22, 2002 18:4

potentially unreliable if it is repeated too many times. Second, we could perform a more hardware-centric solution, upgrading to either create a hardware RAID system, install a software RAID system to stripe the two disks together, or add NVRAM to accelerate the disk's performance. With any of these solutions, upgrading the filesystem might also become necessary. None of these steps is a trivial task, and there's no way to be nearly as certain about the ultimate effect on performance with the addition of so many variables.

Obviously, we can't add disks without considering the potential effect on the I/O controller, and sometimes limits restrict the number of controllers that can be made available in a system. While we rarely push the limits of controller throughput with a small number of disks because email operations are so small and random, it's possible to add enough disks on a system such that we run out of chassis space in which to install controller cards.

Any time a system has I/O problems, it would be a mistake to quickly dismiss the potential benefits of running a high-performance filesystem. This solution is usually cheap and effective, and where available can offer the best bang for the buck in terms of speed improvement. If I am asked to specify the hardware for an email server, in situations where I have complete latitude in terms of the hardware vendors, I know I can get fast disks, controllers, RAID systems, and processors for any operating system. The deciding factor for the platform then usually amounts to which high-performance filesystems are supported. This consideration is that important.

If a RAID system is already in use, performance might potentially be improved by rethinking its setup. If the storage system is running out of steam using RAID 5, but has plenty of disk space, perhaps going to RAID 0+1 will give the box some more life. If it is having problems with write bandwidth, lowering the number of disks per RAID group, and thus having a larger percentage of the disk space devoted to parity may help. Losing unused space is certainly preferable to buying a new storage system. Changing the configuration of the storage system is especially worth consideration if it wasn't set up by someone who really understood performance tuning. The vendor could very easily have given some advice that wasn't optimal for email applications.

If a RAID system has been set up suboptimally, it may also be possible to improve its performance via upgrading. Vendors often provide upgrade solutions to their RAID systems that can improve their throughput, both in terms of hardware components and the software that manages the system. Also, to save money, the system might have originally included insufficient NVRAM or read cache; performance might improve dramatically if more, or any, is installed.

7.5.5 Network-Bound Systems

Two networks are considered: the network(s) under one's control, typically one or more LANs, and the network connection(s) to the Internet, which are usually much more difficult and expensive to upgrade. If the problem lies with the latter, one can do little except to upgrade the server or add an off-site Spillover MX host to ride out the times when network contention causes email to back up. Unfortunately, this tactic doesn't really solve the problem, but merely mitigates it. Further, it's fairly costly in terms of server hardware, maintenance, and potential rack space at a better-connected site. When the problem lies with an internal network, the solution is usually much more tractable.

Reducing the number of other servers contending for time on a cramped network and going to a switched topology are the first things to try if the email server resides on a shared network. If the network is already switched, upgrading speeds and NICs will be necessary, and one will want to make sure the switch itself isn't overloaded.

7.6 Summary

- If an email server sometimes runs out of memory, it may not be memory bound. Swapping can be a symptom that the system has run out of "something else," which has caused processes to back up on the server. This problem eventually leads to memory exhaustion.
- I/O-bound email servers are a common occurrence. Generally, fixing this problem requires adding disks, upgrading storage systems, or upgrading filesystems.
- Often, if a system is network bound, it will be impractical to upgrade the network in the short term. Instead, the email server may need to be upgraded to support deep queues and many concurrent processes.
- When a server becomes saturated, very often the best option is to just let it "work its way out of a jam," rather than trying to find some way to reduce its load. If a server can't handle the load being thrown at it, the demand for its services won't decrease in the short term. The email will keep coming.
- Many tools are useful for determining why a server might run slowly.
- Email applications tend to log a lot of information. This logged information is valuable for assisting in performance tuning, but the logging process itself consumes resources.