16

Server Performance Tuning

In this chapter:
- *Platform*
- *I/O Subsystem Tuning*
- *Memory Tuning*
- *Kernel and Network Driver Tuning*
- *How to Know When It's Time to Scale Up*
- *Running imapd: inetd Versus Standalone*
- *Charting It Up for the Suits*

Although you may very well want your IMAP server to appear as a black box to your users, it should never appear so to you. In this chapter, we'll focus on a few hints that pertain primarily to IMAP servers.

Platform

On what platform should you run your IMAP servers? This is indeed the sixty-four-thousand-dollar question.

Broadly speaking, IMAP servers have many of the same requirements as other server machines. An IMAP server must be robust enough to handle a large number of connections and processes. It must also be able to stay up "24-by-7" so that users have confidence in the service. It should have enough memory and disk to store and process large amounts of mail without significant variations in performance.

The type of server platform that is able to provide that level of performance depends on your user base. As we mentioned earlier, a desktop machine (e.g., an Intel Pentium II with a 9 GB disk and 64 MB of memory, running Solaris X86, Linux, or FreeBSD) works just fine in some environments—e.g., a small company with 100 to 200 employees or an ISP with 500 to 1,000 customers.

Customers of a large ISP, however, might be far less forgiving of intermittent outages for maintenance than corporate users, who may not even notice an outage at four o'clock in the morning. Unless you use some sort of high-availability scheme like IP load-balancing or server clustering, we recommend that your servers exploit server-quality (or server-class) hardware. A server-quality machine is built

from components that are engineered to higher standards than the typical desktop machine. It is also built with room for expansion and includes fully redundant, hot-swappable components. While commodity hardware may be temptingly inexpensive to purchase initially, a server-quality machine could save you hours of downtime you would have spent replacing a cheap but failing power supply or disk in the cheaper machine. Going with server-class hardware doesn't mean that you have to abandon Intel-based architecture. Companies like Compaq, Dell, and HP make very respectable server machines that, with the right software and hardware, can go toe-to-toe with most Unix-only hardware like Sun SPARC. As we write this, one of the top 200 supercomputers in the world is a cluster of commodity-class PCs running Linux* at Sandia National Labs, all connected by fiber channel. Theoretically, you could even run Linux on a top-of-the-line server from Sun or SGI, but we're not certain doing so would put you in the best position with Sun's or SGI's customer service when it came time to ask them for support.†

So what broad conclusions can we come to?

Size for Twice the Expected Load

Stuff happens. That's the only bona-fide guarantee with respect to capacity planning we make in this book.

Given that, your normal workload should never exceed half the capacity of your server. Exceptional circumstances like SPAM attacks, denial of service attacks, or network failures could easily exacerbate system load.

If you want to run your 12,000-user ISP on a single Pentium I box with a 10-year old copy of Xenix and an IMAP server you wrote in Korn shell scripts, that's okay—as long as you don't mind sitting at the console constantly making judgment calls about which sessions to kill and how next to partition and re-process your mail queue. You may have achieved an IMAP server for $200 in capital expenditures since the disk was the only thing you had to buy and the rest of the system was found in an abandoned warehouse. But once you figure in your salary and the number of projects you weren't able to address because you're continuously baby-sitting your server, the cost may be the same as a high-end server—but with only a small fraction of the performance quality.

---

* There are known performance problems with the default Linux filesystem. Tuning or use of an alternative filesystem on Linux systems should be considered.

† Although a free OS on new Sun or SGI equipment may not be the best idea, it might open up interesting OS possibilities for your ancient machines that are past end-of-life and have dubious original vendor support as is, like that old NeXT cube you might have under a pile of old trade magazines in your office.

One of the benefits of going with open source software is that you can hopefully take the money you would have spent on lower performance commercial closed source software and put it into hardware where it will do more good.

Starting up a new mail server is always a bit of a crap shoot. If you're not upgrading from a legacy system, you never have any firm numbers on which to base your sizing decisions. Conventional wisdom is that it's better to overshoot than undershoot your system capacity. Obviously, it's easy to go overboard in this regard. One ISP we know of went shopping for Usenet news servers, bought a good-sized RAID array for storage, but also bought three Origin 2000 servers to use as the NNTP servers. Two ended up taking care of the workload, while a third ended up being used as a sandbox until they found a real use for it.

Redundancy, Redundancy, Redundancy…

The idea of having an array of truly redundant IMAP servers is still in its early stages. Some folks split their users up between two or more IMAP servers. Some have modified the code in their UW servers to take advantage of the hooks for mailbox and login referrals. Most of the really solid schemes for load-balancing across redundant IMAP servers seem to be implemented in closed source commercial products that have draconian control over the functions of everything from the MTA to the server-side MUA. NFS file locking is the primary deterrent to sharing a networked mailstore between IMAP servers. Doing so involves a very high risk that IMAP-related processes, such as *sendmail*, */bin/mail*, Cyrus *deliver*, *procmail*, and *imapd*, will stomp on one other and corrupt your mailstore.* There are, however, plenty of other places to add redundancy in your mail system.

The most obvious place to add redundancy is in the media for the mailstore itself. RAID is the most cost-effective way to do this, as we'll discuss later. One technology that's attractive for large mailstores is a Storage Area Network, or SAN. On a SAN-enabled network, compute servers are almost completely divorced from their data server counterparts. Large, gigabit-networked or multigigabit-networked disks are shared by any number of servers with no common operating system. Many of the standards are still being hammered out, as are the ideas of just the right way to market such things. Many of the server-class hardware vendors have schemes where two or more servers can be locally attached to the same disk array and have it appear as local to all the attached servers. All of these schemes, with the exception of RAID, are the kind of thing that can at least double your budget trying to get from 99.9 to 99.99 or 99.999 percent uptime. Decimal places are very expensive.

---

* Qmail's (*http://www.qmail.org/*) *maildir* format is the only free mailstore format we know of that, by its design, allows writing to an NFS-mounted mailstore.

Keep the Spaghetti on Your Plate…

…not in your mail system. Strunk and White's classic guide to writing, *Elements of Style*, opens with the rule "Avoid Unnecessary Words." A similar text for disciplined system design would open with "Avoid Unnecessary Dependencies."

A good example of this would be to run a caching-only name server on each of your mail servers, especially the SMTP servers. That way, in exchange for a periodic tiny reduction in system performance, your server would be insulated from DNS interruptions of small to moderate length. Another example would be to use an authentication system that incorporates some degree of fault-tolerance or redundancy.

You're probably ready for us to get to the meat and potatoes of this chapter. Let's move on to some of the tangible variables you can examine and change on your system to get more out of what you've got.

I/O Subsystem Tuning

Disk storage requirements can grow by as much as 100% per year. The more storage grows, the more important it becomes to provide reliable access to the data. To ensure both economical and reliable access to data, it's important to consider both scalability and high availability. The section discusses factors to consider when you select a disk subsystem for your IMAP server.

Special I/O Considerations for Cyrus and UW Systems

On Cyrus servers, disk configuration is the most critical factor in system performance. The Cyrus server is I/O bound, and if the disk configuration is not tuned, the system will spend too much time in the I/O wait state. The general idea is to have the logging partition, mailstore, mailboxes file, and mail queue partition as spread out as possible over disks and controllers. Read the Performance Notes section of the Cyrus installation guide for performance tuning guidelines.

On UW servers, disk configuration is the most critical factor in system performance. The UW server is I/O bound, and if the disk configuration is not tuned, the system will spend too much time in the I/O wait state. The general idea is to have the logging partition, mailbox files, and mail queue partition as spread out as possible over disks and controllers. A secondary consideration with UW servers is memory; when in doubt, put more RAM on your UW server, but only after tuning your disk configuration first.

Disk Interface

Integrated Drive Electronics (IDE)

Avoid IDE disks. IDE disks are cheap, but not scalable—the limit on the number of IDE disks you can connect to a bus is lower than with SCSI. Additionally, their transfer rates are lower than high-end SCSI transfer rates.

Small Computer Systems Interface (SCSI)

Fast-wide and Ultra SCSI disks are affordable and offer the performance required by an IMAP server. Fast-wide and Ultra SCSI both support up to 15 devices per bus, making them scalable alternatives. Fast-wide SCSI has a transfer rate of 20 MB/s, and Ultra SCSI's transfer rate is twice that. If you run a Cyrus server and opt for a SCSI disk subsystem, the performance Ultra SCSI will buy you is especially worth the extra cost. Although SCSI is widely used and likely will stick around for quite a few more years, keep in mind that the 15-year-old SCSI disk technology has been superseded in efficiency, performance, and scalability by fibre channel technology.

Fibre Channel Arbitrated Loop (FC-AL)

Fibre channel is nothing more than a high-speed serial connection, indifferent to the format of the data. FC-AL is an enhancement to fibre channel specification that employs a simple loop topology. FC-AL disk subsystems typically transfer data using SCSI protocols, but because the data is transferred over a high-speed fibre connection, FC-AL supports transfer rates of up to 100 MB/s. Up to 126 FC-AL devices are supported per host adapter, making FC-AL far more scalable than SCSI. FC-AL disk array implementations also support hot-pluggable components and multiple host connections.

RAID versus standalone disks

Redundant Array of Inexpensive Disks (RAID) was originally intended to combine small, inexpensive drives to achieve the reliability and performance of a single large, expensive disk. However, because disk manufacturers are now shipping large (50 GB) *inexpensive* disks, capacity is no longer the primary benefit—instead, higher performance and reliability are what a RAID system buys you. RAID arrays are designed to achieve higher performance than independent disks by replicating and/or spanning data across multiple disks.

RAID subsystems can provide high performance if the right configuration is used. If the wrong configuration is used, RAID can actually impair performance. The most popular RAID levels are RAID 0, RAID 1, RAID 1+0, and RAID 5. Each RAID level is explained here and compared in Table 16-1:

RAID 0 (striping)

Striping breaks a stream of data into equally sized chunks and writes the set of chunks (also known as a stripe) sequentially to successive drives in the disk

array. Each stripe spans all drives from the first drive to the last drive in the array. Because each disk in the stripe has its own independent data channel, the transfer rate of RAID 0 approaches the sum of the transfer rates of the individual drives. RAID 0 offers high transaction-based performance because the data is spread across many spindles, which balances the I/O load. RAID 0 does not offer additional fault tolerance over individual disks.

RAID 1 (mirroring)

RAID 1, or mirroring, is designed to offer data redundancy. A mirror consists of two disks. Each write operation is duplicated to both disks—each disk is an identical copy of the other. RAID 1 offers improved performance because reads are taken from only one of the mirrored disks. However, write performance is degraded because both disks are involved in a write operation. RAID 1 is expensive because 100% duplication of data is required.

RAID 1+0 (mirroring and striping)

RAID 1+0 combines mirroring and striping to provide high data redundancy and improved performance. Data is first mirrored for redundancy, then striped for performance. RAID 1+0 can tolerate multiple disk failures with little or no degradation in performance. Like RAID 1, RAID 1+0 is expensive because 100% duplication of data is required.

RAID 5 (striping and distributed parity)

RAID 5 adds fault tolerance to striping by adding error correction information to the data. Both the data and the parity are striped across disks. RAID 5 subsystems typically have good read performance and poor write performance—for each write, the system must first perform four I/O operations and two parity calculations. The main benefit of RAID5 is the cost savings. Depending on the number of disks in the RAID, the overhead is in the range of 20 to 30%, much less than mirroring.

Table 16-1. Comparison of RAID Levels

RAID Level	Strengths	Weaknesses
RAID 0	• Improved I/O performance • Inexpensive (cost = sum of costs of disks)	• No data redundancy—if one disk fails, the entire RAID fails
RAID 1	• Improved read performance in most cases • Data redundancy	• Expensive (cost = 2 × sum of costs of disks) • Decreased write performance
RAID 1+0	• High availability • No performance sacrifice	• Expensive (cost = 2 × sum of costs of disks) • Survives multiple disk failures
RAID 5	• Cheaper than mirroring • Data redundancy	• Poor write performance • Survives only one disk failure

The recommended configuration for Cyrus IMAP servers is either RAID 1+0 or hardware RAID5. For UW IMAP servers, mirroring is recommended because it writes using Berkeley format. The improved performance, especially on large systems, is a necessity.

Filesystem Tuning

If your system has large UFS filesystems, then you will get more efficient storage and improve I/O performance by adjusting UFS parameters. Filesystem parameters are set at the time the filesystem is created with the *newfs* command.

Inode density

Inode density of a filesystem is defined as the number of kilobytes allocated per inode. The inode density determines the fixed number of inodes to create on the filesystem. Another way to think of inode density is as a prediction of the average file size of the files that are stored on the filesystem—the lower the density, the more files. For example, a density of 1 KB/inode is another way of saying that the average size of files on the filesystem is around 1 KB.

It's important from a performance standpoint to make an accurate prediction of the number of inodes. It is even more important for capacity planning. If you underestimate the number and run out of inodes, the result will be the same as if a filesystem reached its capacity—processes, such as your mail delivery agent, will not be able to create new files on the filesystem. Cyrus systems and UW systems have different needs in terms of inode density:

- A Cyrus mailstore stores a large number of small files. The default UFS inode density (2 KB per inode) is sufficient for the needs of the Cyrus system.
- UW systems store a smaller number of "average" sized files, so the default UFS inode density is somewhat thick. Build UW mailstores with a density of at least 8 KB per inode.

This example builds a UFS filesystem with an inode density of 8 KB per inode:

```
# newfs -v -i 8192 /dev/rdsk/md/raid5a
```

Here's an example* of */usr/ucb/df –i* and */usr/ucb/df –lk* output for a typical Cyrus system:†

```
% /usr/ucb/df -i
Filesystem          iused   ifree  %iused  Mounted on
/dev/md/dsk/d0    2509146 3847846    39%   /var/spool/imap
```

* There are a couple of different varieties of *df* on your average Solaris system, and *df* is probably going to have a different syntax on other Unix dialects, so your mileage may vary. Consult the manpage for *df*.

† Output pertaining to other partitions was removed for brevity.

```
% /usr/ucb/df -lk
Filesystem            kbytes    used   avail capacity  Mounted on
/dev/md/dsk/d0       52222520 25546655 26153640    50%    /var/spool/imap
```

As you can see, the mail spool consumes about 50% of disk capacity and about 40% of the inodes. In practice, this is suitable but probably a little close for comfort. With the previous example in mind, suppose the user community changed its usage patterns suddenly, and the average message size took a dive while the average number of messages per user in the mailstore went up. That could bring the filesystem dangerously close to the inode limit. In keeping with the "engineer so that your sustained load never goes above 50% utilization" rule, it might be better to see inode consumption at around 25% here.

Minimum free space

The minimum free space is the percentage of free space to maintain in the filesystem, between 1% and 99%. Minimum free space is space reserved as working space for the operating system to use when a filesystem reaches its capacity. The free space cannot be written by normal users—once the filesystem fills, only the superuser can write to the filesystem. Although the well-known rule is to set the free space to 10% of the filesystem size, that rule applied back in the old days when filesystems were a few megabytes, not gigabytes, in size. The rule is different nowadays, when filesystems are seldom smaller than a gigabyte. Some operating systems automatically optimize minimum free space for large filesystems. If your operating system does not, set the minimum free space to 1% of the total filesystem size on filesystems larger than 2 GB to prevent wasted disk space. The following example shows how to build a filesystem with 1% free space:

```
# newfs -v -m 1 /dev/rdsk/md/raid5a
```

This example shows how to adjust the free space on an already built, not currently mounted filesystem:

```
# tunefs -m 1 /dev/rdsk/md/raid5a
```

Memory Tuning

How Much RAM Is Enough?

Figuring real memory requirements is a complex task and is somewhat platform dependent. On IMAP systems, the physical memory requirements are directly related to the number of users actively reading their mail at one time. IMAP processes are long-lived and typically inactive for long periods. For example, a user may spend 10 minutes reading and replying to a message during one connection. That being the case, there's no reason why an IMAP process should be held in

physical memory for the duration of the connection—it's perfectly acceptable for the process to be swapped out to the swap space.

A survey of large sites that run UW or IMAP servers shows that an IMAP system optimally requires 1 to 2 MB of physical memory per active IMAP session. For UW servers, the 1 to 2 MB rule also applies, but it's advisable to steer towards 2 MB. A UW IMAP process will grow to accommodate the largest message in the mailbox, so a 40 MB message will result in a 40 MB or larger IMAP process. For Cyrus servers, 1 MB per active server is sufficient, provided you optimize your swap space.

Optimizing Swap Performance

Your swap configuration is every bit as important as the amount of physical memory on your system. Swapping occurs when the operating system moves an inactive process from RAM to disk, freeing up RAM for active processes. Here are some guidelines for configuring your system's swap space for ultimate performance:

- Put swap space on a separate disk partition to rule out problems with disk fragmentation, in case the disk becomes fragmented.
- Put swap partitions on separate physical disks. You get a very large performance benefit from having swap partitions on separate disks, because I/O is spread across more than one channel.
- Put swap partitions on your fastest disks on your system.
- A good guideline for figuring total swap space on your system is to make it at least twice the size of your physical memory. Never configure the system with a swap size less than the size of your physical memory—once the system has exhausted its swap space, memory is no longer available to *any* process on the system.
- If you have more than one disk per controller, don't put swap partitions on more than one of the disks on a given controller.
- Don't swap to NFS-mounted partitions.

Kernel and Network Driver Tuning

System tuning or, more specifically, kernel, driver, and filesystem tuning are "lady or tiger" propositions. The best advice is to embrace conservatism. Before you dive into any wholesale system tuning, take some snapshots of how your system is performing over the course of a week or so, make some modest changes, then watch it for another week or so. All too often, system administrators change 5 or 10 independent variables at a time, then fall prey to the fallacy of false cause, assuming that it was one particular that caused that huge increase in system

performance. Of course, it might just be that it's December 20th and most of your user base just left on Christmas vacation.

Diagnose the Problem

You've got a comfortably low system load. You're barely paging at all. You've lots of disk space, and your disk channels are not I/O bound. Your IMAP server performs wonderfully when you crank up PINE on the server itself and point it to localhost as the default IMAP server. Everything ought to be great, but it isn't. You've got a steadily growing queue of users all complaining that IMAP performance has dropped to its knees. Sounds like you have a networking problem on your hands.

This sounds like a job for *netstat*. *netstat* is one of those gifts that come with all Unix systems. Actually, most systems with good TCP/IP support come with *netstat*. Even MS-Windows has it.

Using netstat

Provided you've got connectivity between your clients and servers,* *netstat* can help you characterize your host's network load and make decisions about how to tune your network drivers. Now, it's important to say here that what we're talking about here are "networking," not "network," problems. If your actual network is bogged down, there's probably not very much you can tune on your mail server to make it perform better than the network can deliver.

If you do find, however, that you're experiencing inconsistent performance on network-based activities between hosts on the same network, it's time to do a little digging with *netstat*. *netstat* can tell you, for example, if your IMAP server is still listening on the IMAP port. Example 16-1 shows how to confirm. We're electing to look for the numeric description of the socket address instead of the name "imap," in case there's a problem in the */etc/services* file, in which such symbolic names are stored.

Example 16-1. Checking Up on Your IMAP with Netstat (Solaris Syntax)

```
% netstat -na | egrep '(TCP|Remote|\.143\ )'
TCP
   Local Address        Remote Address    Swind Send-Q Rwind Recv-Q  State
      *.143                *.*               0      0      0      0 LISTEN
129.120.210.4.32816  129.120.210.4.143    32768      0  8192      0 ESTABLISHED
129.120.210.4.143    129.120.210.4.32816   8192      0 32768      0 ESTABLISHED
129.120.210.4.32926  129.120.210.4.143    32768      0  8192      0 TIME_WAIT
```

* By this, we mean that you can PING each from the other (if that's normally permitted), *telnet* to the IMAP port of the server from the client, or otherwise confirm that everyone who ought to be able to get to your IMAP server(s) can probably do so.

In the example (from the workstation on one of our desks), the *.143 in the TCP section tells you that there's a LISTEN at TCP/143. We're going to presume the listen is *inetd* making that socket available to hand off to *imapd*.

If you check the righthand "State" column, you'll see that the last session in the list is in TIME_WAIT state. When a session is in TIME_WAIT, it means that the session was closed some time back, but the socket numbers associated with that session are being held in a kind of IP purgatory for a while to ensure that some other process won't come in and resume the connection before the *imapd* process has a chance to die.

On a production multiuser IMAP server, of course, your output will hopefully be many, many entries long. On our production IMAP server at this moment, for example, we have a few more connections, as the output from the pipeline command in Example 16-2 illustrates.*

Example 16-2. A Quick Way of Seeing the IMAP Connection Load on Your Server

```
imapServer% netstat -na | egrep '\.143\ ' |\
     awk '{print $7}' | sort | uniq -c | sort -rn

 128 ESTABLISHED
  61 TIME_WAIT
   5 FIN_WAIT_2
   5 FIN_WAIT_1
   1 LISTEN
   1 CLOSING
```

In this example, we see that there are:

- 128 active sessions (ESTABLISHED state)
- 61 sessions that are closed for all intents and purposes, but still in the connection table for bookkeeping purposes (TIME_WAIT state)
- 5 sessions that are shut down on the server, but awaiting closure from the client (FIN_WAIT_2)
- 5 sessions that are shut down on both ends and are transitioning to TIME_WAIT (FIN_WAIT_1)
- 1 entry that represents *inetd* waiting for incoming IMAP connections (LISTEN)
- 1 session that is in the process of closing (CLOSING).

Most of the time, you need only be concerned about the load represented by the number of connections in the ESTABLISHED state. If you have an inordinate

* At least on Solaris—you might have to tweak the *awk* command on other systems.

number of connections in non-established states, it might indicate one of two things. Either your network recently had an outage and many IMAP connections were left hanging, or your system could be better tuned to meet the needs of your users. If, after a couple of hours of stable network performance, you don't see an improvement in the ratio of ESTABLISHED to non-established connections, you might want to start tuning your kernel.

On Solaris, at least, *netstat* may also be used to view a bodacious amount of layer-two and lower network statistics with *netstat –k*, as illustrated in Example 16-3.

Example 16-3. The "Netstat -k" Command (Undocumented) to Display Interface Statistics

```
% netstat -k | sed -n /^hme0:/,/^nfs_client:/qfe0:
ipackets 563551 ierrors 0 opackets 15511 oerrors 0 collisions 153
 0 framing 0 crc 0 sqe 0 code_violations 0 len_errors 0
 0 buff 0 oflo 0 uflo 0 missed 0 tx_late_collisions 0
retry_error 0 first_collisions 0 nocarrier 0 inits 25 nocanput 0
allocbfail 0 runt 0 jabber 0 babble 0 tmd_error 0 tx_late_error 0
rx_late_error 0 slv_parity_error 0 tx_parity_error 0 rx_parity_error 0
slv_error_ack 0 tx_error_ack 0 rx_error_ack 0 tx_tag_error 0
rx_tag_error 0 eop_error 0 no_tmds 0 no_tbufs 0 no_rbufs 0 rx_late_collisions 0
```

Here are a couple of common network-related dilemmas and examples of things you might do to address them:

Symptom:

Long-lived IMAP sessions inexplicably die after a long interval (several hours or days) of idle time.

Resolution:

It could be that there's a stateful firewall* between your clients and servers. One way to keep TCP sessions from dying, as they might with long-idle IMAP sessions, is to enable TCP keepalives on your mail server. TCP has no built-in mechanism for "pinging" the other side of connection every so often to make sure it's alive, but some operating systems, including Solaris and Linux, have it built into the kernel. On a Solaris host, you could enable TCP keepalives to be sent once every 50 minutes by putting the command:

```
solaris# /usr/sbin/ndd -set /dev/tcp tcp_keepalive_interval 3000000
```

in an appropriate system startup script (values are given in microseconds in Solaris).

---

* Some firewalls, especially those that make use of Network Address Translation (NAT) IP Masquerading techniques, or so-called private IP address space, use state tables to keep track of who's talking to whom. Timeouts are integral to the operation of such firewalls. Without timeouts, the state table would grow until the firewall became so resource-depleted it failed.

On Linux, the value is given in seconds:

```
linux# sysctl -w net.ipv4.tcp_keepalive_time=3000
```

or:

```
linux# echo 3000 > /proc/sys/net/ipv4/tcp_keepalive_time
```

Symptom:

Connections to your IMAP server from slow (usually dial-up) devices result in an inordinate amount of traffic.

Resolution:

It could be that your TCP driver is a bit too impatient. TCP is a guaranteed delivery protocol that sits on a non-guaranteed delivery protocol (IP). One of the mechanisms for that extra bit of reliability is the TCP ACK, a kind of receipt that is sent to acknowledge each TCP segment received. Because dial-up connections are very slow compared to direct Internet connections, your TCP driver might have its timeouts set at too short an interval to be realistic. On Solaris, you may set these to a longer, say three-second, interval by issuing these commands in an appropriate system startup script:

```
solaris# /usr/sbin/ndd -set /dev/tcp tcp_keepalive_interval_min 3000
solaris# /usr/sbin/ndd -set /dev/tcp tcp_keepalive_interval_initial 3000
```

On Linux:

```
linux# sysctl -w net.ipv4.tcp_keepalive_time=3
```

Symptom:

Large numbers of connections to your IMAP server from average-to-fast devices result in inexplicable network slowdowns.

Resolution:

You may want to crank *down* your maximum TCP retransmit interval. Doing so will increase the number of connections your machine can handle at one time. Here, we're taking it down to one minute on Solaris:

```
solaris# /usr/sbin/ndd -set /dev/tcp tcp_rexmit_interval_max 60000
```

On Linux, decrease the default value of *tcp_fin_timeout* from 180 seconds to 30 seconds, and decrease the default value of *tcp_keepalive_time* from 10,800 seconds to 1,800 seconds:

```
linux# echo 30 > /proc/sys/net/ipv4/tcp_fin_timeout
linux# echo 1800 > /proc/sys/net/ipv4/tcp_keepalive_time
```

tcp_fin_timeout is the time in seconds to wait before forcibly closing a stale connection. *tcp_keepalive_time* is the time in seconds before a keepalive will be sent on a connection.

How to Know When It's Time to Scale Up

If you're reading this section, chances are that you've come across a bottleneck on your system and you need to find the cause. A good way to diagnose the problem is to check the usage levels of each of the following resources:

- CPU usage
- Physical memory usage
- I/O usage
- Networking

This section shows how to use standard Unix tools to characterize your system's performance.

CPU Usage

To check your system's CPU usage, use the *vmstat* command. An example *vmstat* command and its output are shown in Example 16-4. The argument 3 tells *vmstat* to report on system usage every 3 seconds.

Example 16-4. vmstat

```
# vmstat 3
 procs     memory            page            disk          faults      cpu
 r b w   swap  free  re  mf pi po fr de sr f0 s0 s1 s6   in   sy   cs us sy id
 0 0 0 328912 40008   0   0  0  0  0  0  0  0  0  0  0  142 40479  47 77 23  0
 0 0 0 328912 40008   0   0  0  0  0  0  0  0  0  0  0  147 40425  44 83 17  0
 0 0 0 329160 40224   0 256  0  5  5  0  0  0  3  0  0  160 38942  84 79 21  0
 0 0 0 331000 40792   0   2  0  0  0  0  0  0  1  0  0  143 40485  49 83 17  0
 0 0 0 331000 40792   0   0  0  0  0  0  0  0  0  0  0  142 40463  47 87 13  0
```

The last three columns, under the heading "cpu," report the average percentage CPU usage over all processors. The "us" column reports the percentage of processor time used by user processes, the "sy" column reports the percentage CPU used by system processes, and the "id" column reports the percentage idle time. If you have the *top* program installed on your system, you can use it to get much the same information as *vmstat* provides.

What is a heavy CPU load?

For an IMAP server, a load average greater than 1 is considered a heavy load. In that case, you need to determine whether to upgrade to a faster CPU, add another processor to your system, or tune other parts of the system.

If it seems that your CPU load is too high, it almost never means that the CPU is too slow. It's very likely that the CPU is not doing any useful work at all, but

instead is thrashing because of inadequate disk bandwidth or inadequate memory. When in doubt, add another disk and move something to it from a busy disk. If that doesn't improve the situation, add more memory. Memory is cheap, and if you don't need it now, you probably will later on.

There are various ways to figure the CPU load of your machine, and many more opinions on what an acceptable upper threshold is. The *vmstat* output has the information you need to make that determination.

One number to look at is the idle time reported by *vmstat*. In the previous example, the CPU was never observed to be idle. If this machine were ideally loaded, those numbers would always be greater than zero. Fifteen percent or so is nice.

Another important number is the number of jobs in the run state. A snapshot of that number is in the "r" column under "procs" in the *vmstat* output. A more meaningful insight into these numbers is available from *uptime* or *w –u*, where you'll get 1, 5, and 15 minute averages of the run queue size. The output of both the *uptime* and the *w –u* commands is frequently the same:

```
% w -u
 10:54pm  up 4 day(s),  3:40,  1 user,  load average: 1.14, 1.16, 1.16
```

This is the load number used by *sendmail* to indicate the threshold at which to only queue mail, or shut down its listen entirely. We've seen machines function at loads of 20 or 30, but they were crawling along at a snail's pace. System loads of 3 or 4 with peaks at 6 or 7 are usually acceptable. Bear in mind, though, that with some multiprocessing operating systems, the system load is given as the total load across all processors. A load of 16 on a 4-processor system is the same effective load as a system load of 4 on a single processor machine. We've been bitten by this when we unknowingly set *sendmail*'s threshold way too low on a multiprocessor server.

How does one go about lightening the load?

First, check if there are processes contending for CPU. If there are, it will show up in the "procs" column under the "r" heading. The numbers under the "r" column denote the number of processes in the run queue. If the number is 0, as in Example 16-4, then there is no process contention. If the number is greater than 0, then there is contention, and adding an additional processor will help relieve that contention.

If there is no processor contention, then upgrading to a faster CPU will improve performance.

Physical Memory Usage

If it looks like your CPU utilization is okay, then the next point to check is physical memory usage. The *vmstat* provides some clues about memory usage. In Example 16-4, the columns under the "memory" and "page" headings pertain to memory usage.

"swap" reports the kilobytes of swap space in use, and "free" reports total free physical memory. On most Unix systems, this information is ultimately not very useful in determining actual memory usage, because the *vmstat* reports memory that the kernel has reserved for the file cache as used memory. However, that memory is actually available to user applications.

You can better determine your system's memory usage by looking at its paging activity.

I/O Usage

The *iostat* command can be used to monitor how the I/O work is being distributed across your devices. In Example 16-5, the *iostat* command is being run on a system with four disks, one of which (md3) is not currently in use. The *–D* option tells *iostat* to report usage in reads per second, writes per second, and percent utilization. The *–M* option displays data throughput in MB/second (KB/second is the default), and the parameter 3 tells *iostat* to report statistics every three seconds.

Example 16-5. iostat

```
# iostat -DM 3
          md0           md1           md2           md3
 rps wps util  rps wps util  rps wps util  rps wps util
   8   9 14.7    4   9 11.6    4   9 10.7    0   0  0.0
   0   1  0.8    0   1  0.7    0   1  0.7    0   0  0.0
   5   9 17.2    3   9 14.1    3   9 11.8    0   0  0.0
   4  41 40.2    2  41 37.3    2  41 34.0    0   0  0.0
   4  11 16.2    2  11 12.7    2  11 11.6    0   0  0.0
   0   0  0.0    0   0  0.0    0   0  0.0    0   0  0.0
   2   6  7.3    1   6  5.8    1   6  5.9    0   0  0.0
^C
#
```

The two most common symptoms to look for are:

Uneven disk usage

If the percent-utilization varies widely from one disk to another at any given time, then I/O performance can be improved by moving the data to a RAID 0 filesystem striped across several disks.

Busy disks
: If a disk is constantly utilized more than 50% of the time, you can improve performance by redistributing its data across several disks.

Networking

If your server is unable to process requests as fast as your network is sending them, then packets will be dropped and will have to be retransmitted. Retransmissions eat up yet more network bandwidth and cause further congestion.

How do you tell if your network is the bottleneck?

The *netstat* command provides information on network performance. Two factors to pay attention to are the collision rate and the number of input and output errors. In Example 16-6, the "Collis" column reports the collisions, and the "Ierrs" and "Oerrs" columns report input and output errors, respectively. There should be zero collisions on a switched full-duplex line.

Example 16-6. Using the netstat Command for Simple MAC-Layer Statistics

```
% netstat -i
Name  Mtu  Net/Dest      Address        Ipkts  Ierrs Opkts  Oerrs Collis Queue
lo0   8232 loopback      localhost      1570   0     1570   0     0      0
le0   1500 129.120.210.0 nec.unt.edu    13489  0     6689   1     56     0
```

The specific symptoms to look for are:

Collis/Opkts > 10%
: If collision rate is more than 10% of "Opkts" (outgoing packets), that indicates that your network is congested.

Oerrs/Opkts > 0.025%
: A ratio of Oerrs to Opkts that is greater than 0.025% indicates a network hardware problem.

Ierrs/Ipkts > 0.025%
: A ratio of Ierrs to Ipkts in excess of 0.025% indicates that there is an insufficient number of receive buffers.

A wide variety of freely available protocol analysis packages do 90 to 95% of what the high-dollar packages do. *tcpdump* is a popular program (bundled with some flavors of Unix), as is *snoop*, which comes bundled with all Solaris installations. Using *snoop*, as in Example 16-7, you can watch all the IMAP traffic that goes in and out of your machine.

Example 16-7. The snoop Command Used to Capture Two IMAP Protocol Packets

```
# snoop -c2 -vv proto imap
Using device /dev/le (promiscuous mode)

------------------------------
imap.themullets.net -> nec.unt.edu  ETHER Type=0800 (IP), size = 193 bytes
imap.themullets.net -> nec.unt.edu  IP  D=129.120.210.4 S=181.100.100.101 LEN=179,
 ID=12184
imap.themullets.net -> nec.unt.edu  TCP D=32907 S=143     Ack=2835991927
Seq=2841518292 Len=139 Win=8760

------------------------------
 nec.unt.edu -> imap.themullets.net ETHER Type=0800 (IP), size = 85 bytes
 nec.unt.edu -> imap.themullets.net IP  D=181.100.100.101 S=129.120.210.4 LEN=71,
ID=19460
 nec.unt.edu -> imap.themullets.net TCP D=143 S=32907     Ack=2841518431
Seq=2835991927 Len=31 Win=8760
```

In this example, *snoop* was only used to capture two packets with very little detail, so little can be determined here except that one host was indeed able to connect to the IMAP server of another. *snoop* does have the capacity, however, to display a great deal of detail about the packets you snag off the network. Even more useful is its ability to capture packets to a file for later re-examination with a variety of *snoop* options.

Running imapd: inetd Versus Standalone

The question has been brought up from time to time as to whether it's better to run *imapd* standalone, particularly on heavily loaded systems, rather than have *inetd* fork the daemon process. The UW server and Cyrus server prior to the 2.0 release run under *inetd*. Version 2.0 of the Cyrus server runs as a standalone daemon.

If a daemon has to do a lot of work at startup time (e.g., processing a large amount of configuration information), then running it standalone will result in faster connect times for your users. *sendmail* is a good example of a daemon that's better suited to run standalone—its configuration file is large. *imapd* reads a very small configuration file at startup.

The life of the connection and associated *inetd* overhead are other factors to consider. If connections to a daemon are long-lived, then there are fewer startups and thus, less *inetd* overhead. That makes the daemon a good candidate to run under *inetd*. If a connection, on the other hand, is short-lived, it's better not to run the daemon under *inetd* because of the extra overhead *inetd* will concur. *imapd* is an example of a daemon with long-lived connections, compared with daemons that make many short-lived connections, such as *sendmail* and *popd*.

Even on a heavily loaded system (e.g., 1,000 active processes), each *imapd* process will start once and remain active throughout a user's IMAP session. Since the number of connections in any given time period is small, compared to daemons like *sendmail* and *popd*, no significant benefit would be gained by running *imapd* standalone.

Charting It Up for the Suits

A picture's worth a thousand words. Doubly so when doing performance analysis or workload characterization. One of the most popular packages for graphing and gathering performance statistics is the Multi Router Traffic Grapher (MRTG) by Tobias Oetiker and Dave Rand.* Another popular route is the combination of the Gnuplot† and the NetPBM packages.‡

The MTRG package is the better integrated and slicker of the two. MRTG is pretty much a "soup to nuts" system statistical graphing package. It will gather the stats, archive them, and generate the HTML for your system status web page and even draw the graphs for you.

If you've already got a mechanism to gather the statistics (or have one in mind), the Gnuplot/NetPBM package might be a good choice. Gnuplot is very flexible in drawing the graphs, and NetPBM is equally flexible at batch reformatting of the graphics.

* *http://ee-staff.ethz.ch/~oetiker/webtools/mrtg/mrtg.html*

† *http://www.cs.dartmouth.edu/gnuplot_info.html*

‡ *http://wuarchive.wustl.edu/graphics/graphics/packages/NetPBM/*