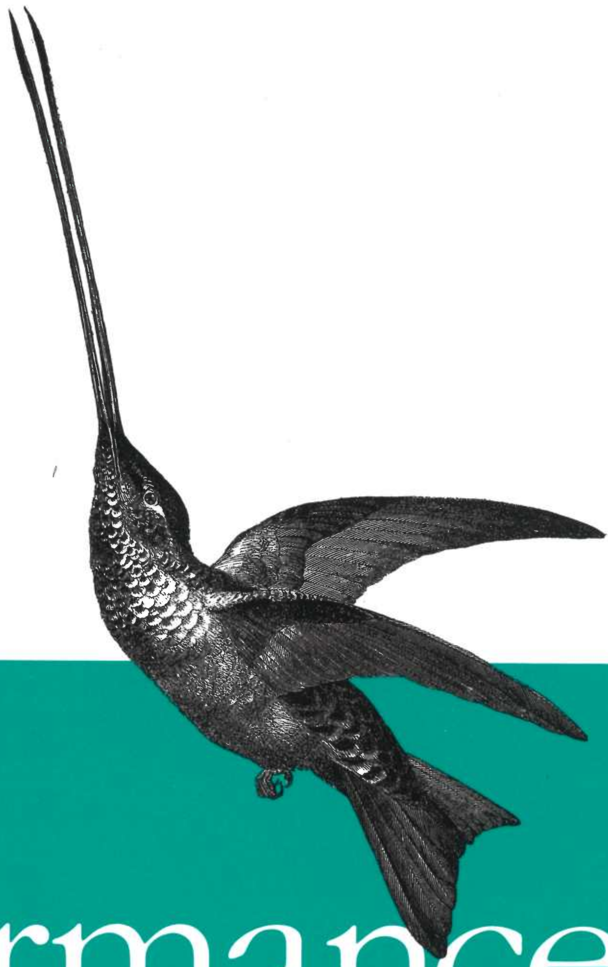
Speeding Up the Web



Web Performance Tuning

O'REILLY™

Patrick Killelea

Microsoft Corp. Exhibit 1044

Web Performance Tuning

Patrick Killelea

O'REILLY™

Beijing · Cambridge · Köln · Paris · Sebastopol · Taipei · Tokyo

Web Performance Tuning
by Patrick Killelea

Published by O'Reilly & Associates, Inc., 101 Morris Street, Sebastopol, CA 95472.

Editor: Linda Mui

Production Editor: Madeleine Newell

Printing History:

October 1998: First Edition.

ISBN: 1-56592-379-0

3 Web Performance Measurement

Parameters of Performance

There are four classic parameters describing the performance of any computer system: latency, throughput, utilization, and efficiency. Tuning a system for performance can be defined as minimizing latency and maximizing the other three parameters. Though the definition is straightforward, the task of tuning itself is not, because the parameters can be traded off against one another and will vary with the time of day, the sort of content served, and many other circumstances. In addition, some performance parameters are more important to an organization's goals than others.

Latency and Throughput

Latency is the time between making a request and beginning to see a result. Some define latency as the time between making a request and the completion of the request, but this definition does not cleanly distinguish the psychologically significant time spent waiting, not knowing whether your request has been accepted or understood. You will also see latency defined as the inverse of throughput, but this is not useful because latency would then give you the same information as throughput. Latency is measured in units of time, such as seconds.

Throughput is the number of items processed per unit time, such as bits transmitted per second, HTTP operations per day, or millions of instructions per second (MIPS). It is conventional to use the term *bandwidth* when referring to throughput in bits per second. Throughput is found simply by adding up the number of items and dividing by the sample interval. This calculation may produce correct but misleading results because it ignores variations in processing speed within the sample interval.

The following three traditional examples help clarify the difference between latency and throughput:

1. An overnight (24-hour) shipment of 1000 different CDs holding 500 megabytes each has terrific throughput but lousy latency. The throughput is $(500 \times 2^{20} \times 8 \times 1000)$ bits/$(24 \times 60 \times 60)$ seconds = about 49 million bits/second, which is better than a T3's 45 million bits/second. The difference is that the overnight shipment bits are delayed for a day and then arrive all at once, but T3 bits begin to arrive immediately, so the T3 has much better latency, even though both methods have approximately the same throughput when considered over the interval of a day. We say that the overnight shipment is *bursty* traffic.
2. Supermarkets would like to achieve maximum throughput per checkout clerk because they can then get by with fewer of them. One way for them to do this is to increase your latency, that is, to make you wait in line, at least up to the limit of your tolerance. In his book *Configuration and Capacity Planning for Solaris Servers* (Prentice Hall), Brian Wong phrased this dilemma well by saying that throughput is a measure of organizational productivity while latency is a measure of individual productivity. The supermarket may not want to waste your individual time, but it is even more interested in maximizing its own organizational productivity.
3. One woman has a throughput of one baby per 9 months, barring twins or triplets, etc. Nine women may be able to bear 9 babies in 9 months, giving the group a throughput of 1 baby per month, even though the latency cannot be decreased (i.e., even 9 women cannot produce 1 baby in 1 month). This mildly offensive but unforgettable example is from *The Mythical Man-Month*, by Frederick P. Brooks (Addison Wesley).

Although high throughput systems often have low latency, there is no causal link. You've just seen how an overnight shipment can have high throughput with high latency. Large disks tend to have better throughput but worse latency: the disk is physically bigger, so the arm has to seek longer to get to any particular place. The latency of packet network connections also tends to increase with throughput. As you approach your maximum throughput, there are simply more packets to put on the wire, so a packet will have to wait longer for an opening, increasing latency. This is especially true for Ethernet, which allows packets to collide and simply retransmits them if there is a collision, hoping that it retransmitted them into an open slot. It seems obvious that increasing throughput capacity will decrease latency for packet switched networks. While this is true for latency imposed by traffic congestion, it is not true for cases where the latency is imposed by routers or sheer physical distance.

Finally, you can also have low throughput with low latency: a 14.4kbps modem may get the first of your bits back to you reasonably quickly, but its relatively low throughput means it will still take a tediously long time to get a large graphic to you.

With respect to the Internet, the point to remember is that latency can be more significant than throughput. For small HTML files, say under 2K, more of a 28.8kbps modem user's time is spent between the request and the beginning of a response (probably over one second) than waiting for the file to complete its arrival (one second or under).

Measuring network latency

Each step on the network from client to server and back contributes to the latency of an HTTP operation. It is difficult to figure out where in the network most of the latency originates, but there are two commonly available Unix tools that can help. Note that we're considering network latency here, not application latency, which is the time the applications running on the server itself take to begin to put a result back out on the network.

If your web server is accessed over the Internet, then much of your latency is probably due to the store and forward nature of routers. Each router must accept an incoming packet into a buffer, look at the header information, and make a decision about where to send the packet next. Even once the decision is made, the router will usually have to wait for an open slot to send the packet. The latency of your packets will therefore depend strongly on the number of router hops between the web server and the user. Routers themselves will have connections to each other that vary in latency and throughput. The odd, yet essential thing about the Internet is that the path between two endpoints can change automatically to accommodate network trouble, so your latency may vary from packet to packet. Packets can even arrive out of order. You can see the current path your packets are taking and the time between router hops by using the *traceroute* utility that comes with most versions of Unix. (See the *traceroute* manpage for more information.) A number of kind souls have made *traceroute* available from their web servers back to the requesting IP address, so you can look at path and performance *to* you from another point on the Internet, rather than *from* you to that point. One page of links to traceroute servers is at *http://www.slac.stanford.edu/comp/net/wan-mon/traceroute-srv.html*. Also see *http://www.internetweather.com/* for continuous measurements of ISP latency as measured from one point on the Internet.

Note that *traceroute* does a reverse DNS lookup on all intermediate IPs so you can see their names, but this delays the display of results. You can skip the DNS lookup with the *-n* option and you can do fewer measurements per router (the default is three) with the *-q* option. Here's an example of *traceroute* usage:

```
% traceroute -q 2 www.umich.edu
traceroute to www.umich.edu (141.211.144.53), 30 hops max, 40 byte packets
 1  router.cableco-op.com (206.24.110.65)  22.779 ms  139.675 ms
 2  mv103.mediacity.com (206.24.105.8)  18.714 ms  145.161 ms
 3  grfge000.mediacity.com (206.24.105.55)  23.789 ms  141.473 ms
 4  bordercore2-hssi0-0.SanFrancisco.mci.net (166.48.15.249)  29.091 ms
39.856 ms
 5  bordercore2.WillowSprings.mci.net (166.48.22.1)  63.16 ms  62.75 ms
 6  merit.WillowSprings.mci.net (166.48.23.254)  82.212 ms  76.774 ms
 7  f-umbin.c-ccb2.umnet.umich.edu (198.108.3.5)  80.474 ms  76.875 ms
 8  www.umich.edu (141.211.144.53)  81.611 ms *
```

If you are not concerned with intermediate times and only want to know the current time it takes to get a packet from the machine you're on to another machine on the Internet (or on an intranet) and back to you, you can use the Unix *ping* utility. *ping* sends Internet Control Message Protocol (ICMP) packets to the named host and returns the latency between you and the named host as milliseconds. A latency of 25 milliseconds is pretty good, while 250 milliseconds is not good. See the *ping* manpage for more information. Here's an example of *ping* usage:

```
% ping www.umich.edu
PING www.umich.edu (141.211.144.53): 56 data bytes
64 bytes from 141.211.144.53: icmp_seq=0 ttl=248 time=112.2 ms
64 bytes from 141.211.144.53: icmp_seq=1 ttl=248 time=83.9 ms
64 bytes from 141.211.144.53: icmp_seq=2 ttl=248 time=82.2 ms
64 bytes from 141.211.144.53: icmp_seq=3 ttl=248 time=80.6 ms
64 bytes from 141.211.144.53: icmp_seq=4 ttl=248 time=87.2 ms
64 bytes from 141.211.144.53: icmp_seq=5 ttl=248 time=81.0 ms

--- www.umich.edu ping statistics ---
6 packets transmitted, 6 packets received, 0% packet loss
round-trip min/avg/max = 80.6/87.8/112.2 ms
```

Measuring network latency and throughput

When *ping* measures the latency between you and some remote machine, it sends ICMP messages, which routers handle differently than the TCP segments used to carry HTTP. Routers are sometimes configured to ignore ICMP packets entirely. Furthermore, by default, *ping* sends only a very small amount of information, 56 data bytes, although some versions of *ping* let you send packets of arbitrary size. For these reasons, *ping* is not necessarily accurate in measuring HTTP latency to the remote machine, but it is a good first approximation. Using *telnet* and the Unix *talk* program will give you a manual feel for the latency of a connection.

The simplest ways to measure web latency and throughput are to clear your browser's cache and time how long it takes to get a particular page from your server, have a friend get a page from your server from another point on the Internet, or log in to a remote machine and run `time lynx -source http://myserver.com/ > /dev/null`. This last method is sometimes referred to as the *stopwatch* method of web performance monitoring.

Another way to get an idea of network throughput is to use FTP to transfer files to and from a remote system. FTP is like HTTP in that it is carried over TCP. There are some hazards to this approach, but if you are careful, your results should reflect your network conditions. First, do not put too much stock in the numbers the FTP program reports to you. While the first significant digit or two will probably be correct, the FTP program internally makes some approximations, so the number reported is only approximately accurate. More importantly, what you do with FTP will determine exactly which part of the system is the bottleneck. To put it another way, what you do with FTP will determine what you're measuring. To insure that you are measuring the throughput of the network and not of the disk of the local or remote system, you want to eliminate any requirements for disk access which could be caused by the FTP transfer. For this reason, you should not FTP a collection of small files in your test; each file creation requires a disk access. Similarly, you need to limit the size of the file you transfer because a huge file will not fit in the filesystem cache of either the transmitting or receiving machine, again resulting in disk access. To make sure the file is in the cache of the transmitting machine when you start the FTP, you should do the FTP at least twice, throwing away the results from the first iteration. Also, do not write the file on the disk of the receiving machine. You can do this with some versions of FTP by directing the result to */dev/null.* Altogether, we have something like this:

```
ftp> get bigfile /dev/null
```

Try using the FTP *hash* command to get an interactive feel for latency and throughput. The *hash* command prints hash marks (#) after the transfer of a block of data. The size of the block represented by the hash mark varies with the FTP implementation, but FTP will tell you the size when you turn on hashing:

```
ftp> hash
Hash mark printing on (1024 bytes/hash mark).
ftp> get ers.27may
200 PORT command successful.
150 Opening BINARY mode data connection for ers.27may (362805 bytes).
#############################################################################
#############################################################################
#############################################################################
#############################################################################
##############################################
226 Transfer complete.
362805 bytes received in 15 secs (24 Kbytes/sec)
ftp> bye
221 Goodbye.
```

You can use the Expect scripting language to run an FTP test automatically at regular intervals. Other scripting languages have a difficult time controlling the terminal of a spawned process; if you start FTP from within a shell script, for example, execution of the script halts until FTP returns, so you cannot continue the FTP session. Expect is designed to deal with this exact problem. Expect is well documented in *Exploring Expect*, by Don Libes (O'Reilly & Associates).

You can of course also retrieve content via HTTP from your server to test network performance, but this does not cleanly distinguish network performance from server performance.

Here are a few more network testing tools:

ttcp
: *ttcp* is an old C program, circa 1985, for testing TCP connection speed. It makes a connection on port 2000 and transfers zeroed buffers or data copied from STDIN. It is available from *ftp://ftp.arl.mil/pub/ttcp/* and distributed with some Unix systems. Try *which ttcp* and *man ttcp* on your system to see if the binary and documentation are already there.

nettest
: A more recent tool, circa 1992, is *Nettest*, available at *ftp://ftp.sgi.com/sgi/src/nettest/*. *Nettest* was used to generate some performance statistics for vBNS, the very-high-performance backbone network service, *http://www.vbns.net/*.

bing
: *bing* attempts to measure bandwidth between two points on the Internet. See *http://web.cnam.fr/reseau/bing.html*.

chargen
: The *chargen* service, defined in RFC 864 and implemented by most versions of Unix, simply sends back nonsense characters to the user at the maximum possible rate. This can be used along with some measuring mechanism to determine what that maximum rate is. The TCP form of the service sends a continuous stream, while the UDP form sends a packet of random size for each packet received. Both run on well-known port 19.

netspec
: NetSpec simplifies network testing by allowing users to control processes across multiple hosts using a set of daemons. It can be found at *http://www.tisl.ukans.edu/Projects/AAI/products/netspec/*.

Utilization

Utilization is simply the fraction of the capacity of a component that you are actually using. You might think that you want all your components at close to 100%

utilization in order to get the most bang for your buck, but this is not necessarily how things work. Remember that for disk drives and Ethernet, latency suffers greatly at high utilization. A rule of thumb is that many components can run at their best performance up to about 70% utilization. The *perfmeter* tool that comes with many versions of Unix is a good graphical way to monitor the utilization of your system.

Efficiency

Efficiency is usually defined as throughput divided by utilization. When comparing two components, if one has a higher throughput at the same level of utilization, it is regarded as more efficient. If both have the same throughput but one has a lower level of utilization that one is regarded as more efficient. While useful as a basis for comparing components, this definition is otherwise irrelevant, because it is only a division of two other parameters of performance.

A more useful measure of efficiency is performance per unit cost. This is usually called *cost efficiency*. Performance tuning is the art of increasing cost efficiency: getting more bang for your buck. In fact, the Internet itself owes its popularity to the fact that it is much more cost-efficient than previously existing alternatives for transferring small amounts of information. Email is vastly more cost-efficient than a letter. Both send about the same amount of information, but email has near-zero latency and near-zero incremental cost; it doesn't cost you any more to send two emails rather than one. Web sites providing product information are lower latency and cheaper than printed brochures. As the throughput of the Internet increases faster than its cost, entire portions of the economy will be replaced with more cost-efficient alternatives, especially in the business-to-business market, which has little sentimentality for old ways. First, relatively static information such as business paperwork, magazines, books, CDs, and videos will be virtualized. Second, the Internet will become a real-time communications medium.

The cost efficiency of the Internet for real-time communications threatens not only the obvious target of telephone carriers, but also the automobile industry. That is, telecommuting threatens physical commuting. Most of the workforce simply moves bits around, either with computers, on the phone, or in face-to-face conversations, which are, in essence, gigabit-per-second, low-latency video connections. It is only these face-to-face conversations that currently require workers to buy cars for the commute to work. Cars are breathtakingly inefficient, and telecommuting represents an opportunity to save money. Look at the number of cars on an urban highway during rush hour. It's a slow river of metal, fantastically expensive in terms of car purchase, gasoline, driver time, highway construction, insurance, and fatalities. Then consider that most of those cars spend most of the day sitting in a parking lot. Just think of the lost interest on that idle capital. And consider the cost of the parking lot itself, and the office. As data transmission costs continue to accelerate their fall, car costs cannot fall at the same pace. Gigabit connections between work and home will inevitably be far cheaper than the daily commute, for both the worker and employer. And at gigabit bandwidth, it will feel like you're really there.

Benchmark Specifications and Benchmark Tests

For clarity, we should distinguish between benchmark specifications and benchmark tests. There are several web benchmarks that may be implemented by more than one test, since there are implementation details that do not affect the results of the test. For example, a well-specified HTTP load is the same regardless of the hardware and software used to generate the load and regardless of the actual bits in the content. On the other hand, some benchmarks are themselves defined by a test program or suite, so that running the test is the only way to run the benchmark. We will be considering both specifications and tests in this section.

The point of a benchmark is to generate performance statistics that can legitimately be used to compare products. To do this, you must try to hold constant all of the conditions around the item under test and then measure performance. If the only thing different between runs of a test is a particular component, then any difference in results must be due to the difference between the components.

Exactly defining the component under test can be a bit tricky. Say you are trying to compare the performance of Solaris and Irix in running Netscape server software. The variable in the tests is not only the operating system, but also, by necessity, the hardware. It would be impossible to say from a benchmark alone which performance characteristics are due to the operating system and which are due to the hardware. You would need to undertake a detailed analysis of the OS and the hardware, which is far more difficult.

It may sound odd, but another valid way to think of a benchmark test is the creation of a deliberate bottleneck at the subject of the test. When the subject is definitely the weakest link in the chain, then the throughput and latency of the whole system will reflect those of the subject. The hard part is assuring that the subject is actually the weakest link, because subtle changes in the test can shift the bottleneck from one part of the system to another, as we saw earlier with the FTP test of network capacity. If you're testing server hardware throughput, for example, you want to have far more network throughput than the server could possibly need, otherwise you may get identical results for all hardware, namely the bandwidth of the network.

6 Client Software

Brief History of the Web Browser

The idea of a hypertext browser is not new. Many word processing packages such as FrameMaker and formats such as PDF generate or incorporate hyperlinks. The idea of basing a hypertext browser on common standards such as ASCII text and Unix sockets was an advance first made by the Gopher client and server from the University of Minnesota. Gopher proved to be extremely light and quick, but the links were presented in a menu separate from the text, and Gopher did not have the ability to automatically load images. The first drawback was solved by the invention of HTML, and the second was solved in the first graphical HTML browser, Mosaic, produced in 1993 at the University of Illinois National Center for Supercomputing Applications (NCSA).

Many of the original students who developed Mosaic were among the founders of Netscape the following year. An effort by the University of Illinois to commercialize Mosaic led to the founding of Spyglass, which licensed its code to Microsoft for the creation of Internet Explorer. Netscape and IE have been at the forefront of browser advances in the last few years, but the core function of the browser, to retrieve and display hypertext and images, has remained the same.

How Browsers Work

The basic function of a browser is extremely simple. Any programmer with a good knowledge of Perl or Java can write a minimal but functional text-only browser in one day. The browser makes a TCP socket connection to a web server, usually on port 80, and requests a document using HTTP syntax. The browser receives an HTML document over the connection and then parses and displays it, indicating in some way which parts of the text are links to other documents or images. When the user selects one of the links, perhaps by clicking on it, the process starts all over again, with the browser requesting another document. In spite of the advances in HTML, HTTP, and Java, the basic functionality is exactly the same for all web browsers.

Let's take a look at the functionality of recent browsers in more detail, noting performance issues. To get the ball rolling, the browser first has to parse the URL you've typed into the "Location:" box or recognize which link you've clicked on. This should be extremely quick. The browser then checks its cache to see if it has that page. The page is looked up through a quick hashed database mapping URLs to cache locations. Dynamic content should not be cached, but if the provider of the content did not specify an immediate timeout in the HTTP header or if the browser is not clever enough to recognize CGI output from URLs, then dynamic content will be cached as well.

If the page requested is in the cache and the user has requested via a preference setting that the browser check for updated versions of pages, then a good browser will try to save time by making only an HTTP HEAD request to the server with an `If-modified-since` line to check whether the cached page is out of date. If the reply is that the cached page is still current, the browser simply displays the page from the cache. If the desired web page is not in the cache, or is in the cache but is stale, then the browser needs to request the current version of the page from the server.

In order to connect to a web server, the client machine needs to know the server's 4-byte IP address (e.g., 198.137.240.92). But the browser usually has only the fully-qualified server name (e.g., *www.whitehouse.gov*) from the user's manual request or the HTML of a previous page. The client machine must figure out which IP address is associated with the DNS name of a web server. It does this via the distributed database of domain name to IP mappings, that is, DNS. The client machine makes a request of its local name server, which either knows the answer or queries a higher-level server for the answer. If an IP answer is found, the client can then make a request directly to the server by using that IP address. If no answer is found, the request cannot proceed and the browser will display "No DNS Entry" or some other cryptic message to the user.

The performance problem here is that DNS lookups are almost always implemented with blocking system calls, meaning that nothing else can happen in the browser until the DNS lookup succeeds or fails. If the local DNS server is overloaded, the browser will simply hang until some rather long operating system timeout expires, perhaps one minute. DNS services, like most other Internet services, tend to get exponentially slower under heavy load. The only guaranteed way to avoid the performance penalty associated with DNS is not to use it. You can simply embed IP addresses in HTML or type them in by hand. This is hard on the user, because DNS names are much easier to remember than IP addresses, and because it is confusing to see an IP address appear in the "Location:" box of the browser. Under good conditions, DNS lookup takes only a few tenths of a second. Under bad conditions, it can be intolerably slow.

The client-side implementation of DNS is known as the *resolver*. The resolver is usually just a set of library calls rather than a distinct program. Under Unix, for example, the resolver is part of the *libc* library that most C programmers use for their applications. Fortunately, most DNS resolvers cache recently requested DNS names, so subsequent lookups are much faster than the first.

Once a browser client has the IP address of the desired server, it generates the HTTP request describing its abilities and what it wants, and hands it off to the OS for transmission. In generating the HTTP request, the browser will check for previously received cookies associated with the desired page or DNS domain and send those along with the request so that the web server can easily identify repeat customers. The whole request is small, a hundred bytes or so. The OS attempts to establish a TCP connection to the server and to give the server the browser's request. The browser then simply waits for the answer or a timeout. If no reply is forthcoming, the browser does not know whether it is because the server is overloaded and cannot accept a new connection, because the server crashed, or because the server's network connection is down.

When the response from the server arrives, the OS gives it to the browser, which then checks the header for a valid HTTP response code and a new cookie. If the response is OK, the browser stores any cookie, parses the HTML content or image, and starts to calculate how to display it. Parsing is very CPU-intensive. You can feel how fast your CPU is when you load a big HTML page, say 100K or more, from cache or over a very fast network connection. Remember that parsing text is a step distinct from laying out and displaying it. Netscape, in particular, will delay the display of parsed text until the size of every embedded image is known. If the image sizes are not included in the HTML <IMG> tag, this means that the browser must request every image and receive a response before the user sees anything on the page.

The order in which an HTML page is laid out is up to the particular browser. In Netscape 4.x, web pages are rendered in the following order, once all the image sizes are known:

1. The text of the page is laid out. Links in the text are checked against a history database, and if found, are shown in a different color to indicate that the user has already clicked on them.
2. The boundary boxes for images are displayed with any ALT text for the image and with the image icon.

3. Images are displayed, perhaps with *progressive rendering*, where the image gains in definition as data arrives rather than simply filling in from top to bottom. It is common for Netscape to load and show an image before showing any text.
4. Subsidiary frames are loaded starting over at step 1.

A browser may open multiple connections to the server. You can clearly see this by running *NetStat -c* to poll network activity on a Linux client, and then using Netscape to request a page with multiple embedded images. You'll probably see about five connections open, indicated by the word `ESTABLISHED` in the state column. The number of simultaneous connections is a tunable option in some browsers. Clients with fast Internet access will benefit from simultaneous connections. Clients with slow Internet access may see no improvement from simultaneous connections because they are already using all of their bandwidth. It is more efficient to use HTTP 1.1's persistent connections and to pipeline requests than to use multiple TCP connections. *Pipelining* means starting another request before the previous request has returned a complete response. HTTP 1.1 will be used automatically if your browser and the contacted server support it.

You can see the progress of the various downloads in the Netscape footer messages: every flash of a URL is an HTTP connection for an HTML page, an image, or a Java *.class* or *.jar* file. It is usually too confusing to try to figure out which connection being shown in the footer corresponds to an image on the page. The HotJava™ browser has a much clearer display of the progress of downloading as several parallel lines getting longer as files load.

If the user hits the browser's Stop button, a TCP reset, also called an abortive release, is sent to the server immediately. See *TCP/IP Illustrated, Volume 1*, by Richard Stevens (Addison Wesley), for the TCP details of a reset. If the server is calculating CGI output when the Stop button is hit, the CGI process will not hear about it until it completes and tries to send the output back to the web server for forwarding to the client. Under Unix, the CGI process will get a SIGPIPE signal because the socket to the web server is no longer valid.

Popular Browsers

There are a lot of browsers, but only a few are widely used. See *http://www.boutell.com/openfaq/browsers/* for a comprehensive list. The following five browsers are distinguished either by their market share or their features.

Netscape

As of this writing, Netscape Navigator, usually called "Netscape," still has the largest browser market share, but it has been losing ground to Internet Explorer. Netscape 4 was a major overhaul of Netscape 3. Netscape 1.0 and later versions all have persistent connection ability, but use it via a `Connection: Keep-Alive` header rather than as part of a full implementation of HTTP 1.1. See Chapter 10, *Network Protocols*, for more about HTTP. Netscape exists as native code for Linux, Solaris, Macintosh, Windows, and many other platforms.

Netscape has made the source code for their browser available on the Web at *http://www.mozilla.org/*. This opens the door to performance improvements by the Internet community as a whole. I'm personally hoping that someone will write a filter which eliminates blinking GIF advertisements.

Internet Explorer

Internet Explorer is bundled with every copy of Windows 3.1, 95, and NT. Because Windows has a monopoly on commercial PC operating systems, Internet Explorer is already installed on nearly every PC desktop. This fact, combined with the similarity of the two browsers, removes the incentive to even take the time to install any other browser. Whether Microsoft may continue to bundle IE with Windows is in the courts as of this writing because the bundling looks very much like an abuse of monopoly power.

That said, Internet Explorer does have a few performance features to recommend it. First, it outputs document requests with the HTTP 1.1 header, implying that it has full HTTP 1.1 support. Beyond keepalives, HTTP 1.1 has support for byte range downloading, the continuation of interrupted transfers, and other features that improve performance under certain conditions. IE also seems always to display the text of a page first, before any images, so that you can start to read immediately and decide whether you want to stop the download.

IE exists for Windows, the Mac, and Solaris, although the non-Windows versions have less functionality. There is also a look-alike version for Linux running *fvwm95*, an X Window manager that looks just like Windows 95, available as shareware from *http://jungfrau.ptf.hro.nl/explorer/,* but of course this version contains no code from Microsoft and runs only on Linux.

Appendix C of *Professional Web Site Optimization,* by Scott Ware et al. (Wrox Press) shows that Netscape 3.0 is about twice as fast at loading and displaying web pages than Internet Explorer 3.0. The difference is attributed to IE's need to support the COM threading model.

Mosaic

The original web browser developed by Marc Andreessen and others at the University of Illinois at Urbana-Champaign is NCSA Mosaic. The feature that continues to distinguish Mosaic is support for *gzip* decompression. On the server side, you can configure Apache to report to Mosaic that a file is *gzip* compressed by adding the following lines to Apache's *srm.conf*:

```
# AddEncoding allows you to have certain browsers (Mosaic/X 2.1+) uncompress
# information on the fly. Note: Not all browsers support this.
AddEncoding x-compress Z
AddEncoding x-gzip gz
```

This is useful for large text files, where *gzip* can decrease the size by half, but is not worth the trouble for very small files, because the download difference will not be noticeable and you will still have the overhead of starting *gunzip*.

Mosaic supports Unix, Windows, and Macintosh. It has the additional advantage that it is available at no cost along with the source code.

Lynx

Lynx is a text-only web browser available from *http://lynx.browser.org/*. It was originally developed at the University of Kansas. It is capable of displaying images via helper applications. Advantages of Lynx are that it's free along with the source code, you can run it over shell accounts as well as directly via PPP, it has a *-source* option that is convenient for scripting the retrieval of data, and it's very fast. We saw a use of the *-source* option in Chapter 3, *Web Performance Measurement*.

HotJava

HotJava is a web browser from Sun Microsystems written entirely in Java. The performance is lower than native-code browsers, but it has the advantage of pure Java portability. Sun also sells HTML-rendering JavaBeans™ that can be used in Java applications where browser functionality is needed.

Non-Browser Web Clients

Given the simplicity of programming basic web functionality, it is entirely practical to write your own HTTP clients for specific uses. Currently, the most popular non-browser web clients are robots that index the Web for search engines and link checkers that find broken links, but there are an infinite number of possible client programs and devices. In fact, connecting telephones, TVs, copiers, and other devices to the Web is the main business strategy of Spyglass (*http://www.spyglass.com/*), the Mosaic code licensee. See *Web Client Programming with Perl*, by Clinton Wong (O'Reilly & Associates).

The appletviewer program that ships with Sun's Java Development Kit™ loads applets much faster than Netscape or other browsers, probably because it has no cache to check for previously loaded classes. The appletviewer works well with Java *.jar* files.

Browser Speed

The web browser is probably not going to be your bottleneck simply because the average performance of a point-to-point TCP connection on the Internet is only about 50KByte per second,* while most browsers are able to parse and display data faster than that. My own seat-of-the-pants benchmark is that Netscape 4 running on a 75MHz Pentium laptop under Linux 2.0 can parse a large HTML file from memory cache or from a 10Mbps LAN connection at a rate of about 80KB per second. On the other hand, Netscape 4 running on a Mac PowerBook 5300cs seems to achieve only about 4KB per second reading from memory cache or a LAN.

One might think that browsers would store cached documents in a parsed format for quicker subsequent display, but an examination of the cache shows that this is not the case. This raises the interesting possibility of pre-parsing HTML on the server and storing it in parsed format. There is no standard format for parsed HTML, so the performance gain would be at the expense of portability and human readability. In any case, it is unusual for the Internet to return enough data in HTTP replies to overwhelm the parsing capability of the browser. What this means for capacity planning is that performance is not currently a factor in choosing web browsers, although this may change as Internet infrastructure is upgraded.

In the outbound direction, web browsers do not need to make sustained HTTP requests any faster than a human can digest the replies. Web browsers are generally capable of about 10 HTTP connections per second. Even a person ready to click away furiously could not click on 10 distinct links in one second, but it is possible for a multithreaded browser to reach this rate when parsing HTML pages with many embedded images or applets. Even a burst of 10 requests in a second from one browser is no particular problem for most servers. A typical average rate for requests from a browser is less than 1 HTTP operation per second.

* See *http://www.keynote.com/measures/top10.html* and *http://www.orckit.com/* for Internet performance statistics.

Microsoft Corp. Exhibit 1044

Browser Tuning Tips

General Tips

Upgrade

Try to get the latest non-beta version of your browser. Newer versions usually include new features like HTTP 1.1 persistent connections to improve performance. Still, there are some things to be said for the older versions. First of all, beta versions of new browsers often have bugs and performance problems associated with them, while the older non-beta versions are more stable. Netscape 4.0 beta ran Java especially slowly, but this was fixed in the version that was officially released. You may want to wait until a browser is officially released before trying it. Second, browsers have been getting fatter very fast. Netscape 3 for Linux takes about 5M of memory when you first start it up; Netscape 4 takes about 8M. As you use them, they both grow through the loading of features and through memory leaks. If you're memory-constrained, you will get better performance with the older version, especially if it makes the difference between swapping to disk or not.

Do less

You can change your browser's settings so that the browser does only the minimum necessary to get and show you a page. First, turn off automatic loading of images, since they take up most of your bandwidth, and each requires a separate connection unless both your browser and server understand persistent connections. You'll see placeholders for unloaded images that you can click on individually for loading, or you can load them all at once through a menu option. Similarly, you should probably turn Java off if you are bandwidth- or memory-constrained.

To get the browser to start a little more quickly, set it to load only a blank page on startup. Load as few plug-ins as possible, because they add to your startup time. On a Macintosh, load fewer fonts.

To prevent any network access when you have a page in the cache, set the verify option to "never". This risks your viewing out-of-date pages, but it is a significant performance boost if you do have the page in cache.

You can clear your history cache for another slight performance boost, but you will no longer see visited links in a different color. In Netscape 4.0, you clear the history cache like this: Edit → Preferences → Navigator → Clear History button. If you find this is a significant help, you can set your history of links visited to always expire immediately so that you never spend time recognizing and colorizing visited links. This does not affect your browser's cache.

Not accepting or sending cookies will give another slight gain in performance and a large gain in privacy, but this will break many web sites that depend on cookies for their functionality, such as sites that personalize content for you.

Save frequently accessed pages, such as search engine home pages, to a local file on your hard disk with File → Save As, and bookmark where you saved them. The next time you go to one of these saved pages, you won't have any network traffic, and it won't expire from the browser's cache. You may have to modify the HTML in the saved file so that all links are absolute, that is, so they include the server name, because relative links are now relative to the filesystem on your own machine rather than to the document root of the original web server. The easiest way to do this is to add a <BASE> tag to the HTML head, like this:

```
<html>
<head>
<base href="http://www.search.engine.com/">
</head>
...
```

This is also an opportunity to eliminate the lines of HTML that put in the blinking GIF ads so you won't have to see the ads right away. Once you get a page back from the original server—for example by submitting a search request—the HTML you'll get back will be new, and you'll have ads again.

You can cache whole sites in a similar way by choosing a new cache location with Netscape's preferences, then simply browsing the site you want to cache. The new cache will contain the site, and you can keep it there permanently by setting the cache location back to the cache used for most of your browsing. In Navigator 4, you change the cache location with Edit → Preferences → Advanced → Cache → Cache Folder.

Finally, make good use of the Stop button. Stop page downloading if you know you don't want the rest, and stop animated GIFs if you are CPU constrained. Infinitely looping animated GIFs can easily waste 10% of your CPU capacity and will continue to do so until you stop them. This is an issue if you are running the browser and some other CPU-intensive applications at the same time. Some Java applets may also consume CPU time even when you're not using them, or even after you've left that web page. This happens because the applet programmer did not override the applet's `stop()` method, for whatever reason. Applets cannot be stopped with the Stop button, but you can turn off Java to stop them. In Netscape 4, to turn off Java, select Edit → Preferences → Advanced and deselect "Enable Java".

Use shortcuts

A number of shortcuts can improve the user's performance in handling the browser, rather than improving the browser's performance itself.

First of all, you don't always have to type in complete URLs. Most browsers will fill in the rest of a URL with the default "http://www." and ".com" if you just type in the domain name. So, for example, you can type "sun" instead of "http://www.sun.com/". This may actually slow performance or not work at all in some cases. For example, if you are in an organization named Zort, Inc., that has set up its DNS to assume that incomplete URLs should end in your organization's domain name, "zort.com," then entering simply "sun" in your browser will cause a lookup of *http://sun.zort.com*. If there is no local machine named "sun," your DNS may or may not redirect the browser to *http://www.sun.com/*, depending on how DNS was configured.

If the web server machine name is something other than "www," you can still leave off the "http://" part. So if the server is named "web" at zort.com, you can type in "web.zort.com" instead of "http://web.zort.com".

Second, use keyboard shortcuts when they are available, like the Escape key for Stop. You'll find that you can type the keys for Stop, Back, Forward, Open, etc., much faster than you can move the mouse to select them. You have to know what they are, but once you get used to them, you'll never go back. This is also why Unix command-line users tend to disdain GUIs. It's much faster to type a command than to move the mouse. If you are presented with a GUI menu or set of buttons, note that Tab or Alt-Tab will sometimes change which one is currently selected, but it may be quicker here to use the mouse than to tab around in a GUI.

Third, use the Go menu to select a page from your history of recently viewed pages rather than hitting the back button repeatedly. It's much faster. IE has a nice feature that automatically tries to complete the URL you are typing with recently viewed URLs.

Increase caches

Pump up the memory and disk cache in your browser if you can afford it. Clearing your browser's memory and disk cache may help a little if you are hitting new sites, since the browser will have less to check, but it will hurt a lot if you are hitting sites you've seen before, since all the data must be downloaded again. As a rule, you want to set the cache to be as big as your machine can handle.

Reboot

It is unfortunate but true that Netscape and Internet Explorer both tend to take memory from you and not give it back. This can be due to inadvertent memory leaks or to the accumulation of features loaded into memory as you use the browser. If you notice that your browser runs much faster after you quit and restart it or after you reboot your machine, that's an indication that your browser is hogging memory. There's not much you can do in that case but restart the browser regularly or not load features like Java or the mail reading tool.

Multitask

If a page is taking a long time to load, you can open a new browser window and continue browsing in the new window while the old one churns away. In Netscape, use File → New → Navigator Window, or even better, Alt-N.

Stop it

The Stop button can actually speed things up. Hitting Stop will cause Netscape to stop loading and to display what it's got so far, which may be all you need. If you hit Stop and it turns out nothing was loaded, hit Reload and you may find that the server responds much more quickly the second time around. It is possible that the packet containing your request was lost somewhere on the Internet.

A major flaw with browsers right now is the fact that you cannot stop Java from starting once you've begun to download an applet, and you can't do anything else until Java is fully initialized.

Use Activator

There is an ActiveX control and Netscape plug-in called Activator that will download a current and correct Virtual Machine (VM) from JavaSoft™. Simply refer to the Activator VM in your HTML, and if the new VM is not already installed, the browser will ask you if you'd like to download and automatically install it. Not only does this insure that your clients will consistently have the latest and fully functional VM, but it also solves some performance problems with other implementations of the VM. Netscape has been known to download and instantiate Java classes several times more slowly than IE, but this problem goes away with Activator. See *http://www.javasoft.com/products/activator/*.

Internet Explorer Tips

Don't redraw while scrolling

If your machine doesn't have enough spare CPU cycles to keep the images smooth when you scroll in IE, you can turn off smooth scrolling with ViewInternet Options → Advanced → Disable Smooth Scrolling. You will be spared the ugly image of your machine struggling to keep up, and scrolling will feel snappy.

Browse in a new process

If hitting the back Button on IE causes a long delay and always gets the page via modem regardless of whether or not you told it to get pages from cache, try setting View → Internet Options → Advanced → Browse In A New Process. This will start new copies of IE which do not share system resources. The idea behind the option is to isolate IE from your system in case IE crashes, but it may have the side effect of making the Back button work quickly and correctly.

Netscape Tips

Prestart Java

Rather than get stuck watching the Java VM start up the first time you hit an applet, you can ask Communicator to initialize Java with the browser itself by using the command line *netscape.exe -start_java*. This option is not available for the Unix versions of Netscape.

Use fewer colors

If you don't really care about accurate colors in Netscape on a PC, you can use approximate colors for a significant speed gain. On Communicator 3.0, select General Preferences → Images → Substitute Colors. On Communicator 4.0, select Edit → Preferences → Appearance → Colors and check "Always Use My Colors". This doesn't work on the Macintosh or on Unix.

Make smaller buttons

You can save a little bit of valuable screen real estate by showing the buttons as text only. In Netscape, Edit → Preferences → Appearance → Show Toolbars As → Text Only. Or, you can eliminate them altogether if you know all the keystroke shortcuts.

Figuring Out Why the Browser Is Hanging

Is your modem still on and connected to your computer?

If you have an external modem, diagnosing problems is easier. At the least, a power light should be lit to indicate that the modem is on. If the modem is definitely on, try manually sending something to the modem to prove it is connected to your computer. On Linux you can do this:

```
% echo AT > /dev/modem
```

From a DOS shell on a Windows machine you can do this:

```
> echo AT > COM1
```

If your modem is connected, you will see the send and read lights flash. If the lights do not flash, either the modem is not connected, or you have configured it for the wrong COM port, PCMCIA slot, or other attachment point.

Are you still online, transmitting and receiving?

Your modem should also have a light labelled CD (Carrier Detect) to indicate if there is a carrier signal, that is, whether you are online. If it is not lit, it may be that the remote end hung up on you, or you lost your connection through too much noise on the line.

If you have carrier detect and can manually get your modem to respond but your browser cannot, then the browser is not communicating correctly with your operating system, implying a TCP/IP stack problem or a PPP problem.

Open another window and make another request. If you have an external modem, look at the modem lights. The read and send lights should be flashing. The send light will tell you that your modem is trying to send data out to the Internet. The read light will tell you if your modem is getting anything back from the network. If you cannot see these lights flashing, there is no data flowing through the modem.

Can you do anything at all?

Browsers have been known to hang. On the other hand, your browser may just be thinking some deep thoughts at the moment. Give it a minute, especially if you just requested a page. The system call to resolve DNS names often hangs the browser for a minute if the DNS server is slow. If you give it a minute and it's still stuck, kill the browser and try again.

Can you still resolve names?

Maybe your DNS server is down. Try a known IP address in the browser. In case you don't keep the IP addresses of web servers around, here are a few: 198.137.240.91 (*www.whitehouse.gov*); 192.9.9.100 (*www.sun.com*); and 204.71.200.66 (*www.yahoo.com*). Make a request with a URL like this: *http://198.137.240.91/*. If that works but *http://www.whitehouse.gov/* does not, your problem is DNS resolution.

If you're on a Unix system, try *nslookup* or *dig* on any domain name. You'll know immediately whether you can still resolve names.

Is the remote web server still up and available?

Try to *ping* the server you're interested in. If *ping* replies, then the web server is definitely alive. Telnet to it on port 80. From a Unix command line, to check on the White House web server, you would type this: `telnet www.whitehouse.gov 80`. If telnet connects, the web server is up and accepting connections on port 80. If you can't *ping* or telnet, try a *traceroute* to the server to see how far you can get. The traceroute program comes packaged with most versions of Unix, but there is also a commercial NT version called Net.Medic from VitalSigns software. If *traceroute* stops within your ISP, it could be that your Internet provider is down. Sometimes your whole region may be down because of a NAP or Internet backbone issue.

Did you already get most of the page?
Maybe everything is working fine, but you are stuck waiting for that last GIF. Hit the Stop button and see if the page renders.

Key Recommendations

- Set the browser to check cache only once per session, or never.
- Increase the memory and disk cache as much as feasible.
- Use the latest browser version for HTTP 1.1 and other advantages.
- Set the browser to load a blank page on startup.

In this chapter:
- *Macintosh*
- *Microsoft Windows*
- *Unix*
- *Key Recommendations*

7 Client Operating System

Volumes have been written about optimizing desktop systems, so here I'm going to touch on only a few points that directly relate to your web browsing performance.

Macintosh

68K Emulation

Apple's change of CPU from Motorola's 68000 series to the PowerPC increased Mac performance in one way, but hurt it in another. Performance is helped because the PowerPC is a much faster, more modern CPU. Performance is hurt because most of the software available for the Mac was written for the 68K chip and runs on the PowerPC chip only in emulation. Not only the applications but also parts of the Mac OS themselves were left in 68K binary format. There are a few things you can do to minimize the impact of emulation on your performance:

- First, always try to get a native PowerMac version of your browser in preference to a 68K version.
- Second, try replacing the emulator that ships with the OS with Speed Doubler from Connectix, *http://www.connectix.com/*. Connectix also makes a well regarded RAM Doubler product.
- Finally, upgrade to Mac OS8, which has more native PowerPC code and faster and more robust TCP/IP.

Networking

For the best networking performance, make sure you're using a recent native-binary Open Transport TCP/IP stack. See Open Transport Mac tips at *http://ogrady.com/FAQ/powerbook/9.html#29*.

Macintosh PPP programs can usually be configured to automatically dial the modem and start PPP when an application such as a browser needs network connectivity. This makes start-up a little easier on the user.

Use the Mac TCP Monitor to check whether TCP packets are timing out and retransmitting, that is, if they're being reported as lost when they're really just pokey. Retransmits can happen if you are on a slow connection or if there are errors in the data received. If the TCP Monitor shows retransmits, you may want to set the TCP retransmit timeout higher and see if that helps. If the retransmits were happening because of errors instead of a high latency connection, increasing the timeout will actually hurt performance.

Another network parameter you can modify is the Maximum Transmission Unit (MTU), which is the largest IP packet your machine will send. You want to limit this to the size of the largest MTU allowed along the route to any particular web server. If you hit a router with an MTU smaller than yours, then your IP packets may be fragmented from that router on, which will slow them down. Try changing MTU in PPP from 1500 to 576, which is the largest MTU size guaranteed not to fragment. Again, be careful to check performance before and after because this will definitely slow down performance if fragmentation was not your problem.

Memory and Disk

If you've used Netscape on a Mac after using it on Windows or Unix, you may have noticed that the Mac seems to be missing a memory cache in the Netscape configuration options. There is a memory cache for Netscape, but it isn't handled from the Netscape preferences. Rather, it is handled from the Control Panels as general disk cache, in the sense of caching disk data in memory rather than in the sense of explicitly caching web pages on disk. You want to have a disk cache large enough to improve performance by avoiding a significant number of disk accesses, but not so large that you starve applications of memory and hurt their performance. A rule of thumb is to have about 32K of cache for every 1M of RAM in the computer; so, for example, a 24M computer should have 768K of disk cache. To change the disk cache setting, select Control Panel → Memory → Disk Cache.

Macintosh applications, at least under System 7, cannot dynamically take memory as they need it, as they can under other operating systems; you have to specifically allocate more memory to an application if needed. To find out how much

Microsoft Corp. Exhibit 1044

memory is allocated to Netscape and to change it, first make sure Netscape is not running, then select the Netscape icon and click on File → Get Info.... You'll see two values: the minimum size (what Netscape minimally needs to run), and the preferred size (the most Netscape is allowed to have). You can't do much about the minimum size, but if you give a larger preferred size, Netscape may run a little faster. 16MB is a reasonable value for Netscape 4.0.

Extensions

Extensions consume system resources, so removing all unnecessary extensions will improve overall performance as well as making boot time shorter. You can turn off all extensions by holding down the Shift key when the Macintosh is booting, but this will also turn off extensions that help performance, like the Symantec JIT.

When you use Netscape to browse a Java-enabled page, Java will start more quickly if you move the Symantec JIT, labelled "Java Accelerator for the Power PC," out of the *Netscape* folder and into the *Extensions* folder. This ensures that it will be loaded on startup. Make sure you move it and don't just copy it.

Microsoft Windows

The most effective way to improve the performance of PC hardware running Windows is to erase Windows and install a version of Unix for Intel, such as Linux, Solaris x86, FreeBSD, BSDI, or SCO Unix. Unfortunately, this is not an option for most people because of the complexity of installing an operating system and the need to run applications that run only on Windows. If you have to live with Windows, there are some things you can do to improve browser performance.

System Clutter

Back up your system files, then remove utilities you don't use from your *win.ini* and *system.ini* files. As with eliminating unused Mac extensions, the utilities that are left will run faster.

Specific Video Drivers

A device driver written specifically for your video card should give you better performance and stability than the default VGA driver. Here are some indications that you need a better video driver:

- Changing Control Panel → Display → Settings causes crashes if you are using particular screen sizes or number of colors but not otherwise.

- Setting Control Panel → System → Performance → Graphics → Maximum Hardware Acceleration causes the machine to crash.
- The browser bombs reliably at certain web sites. This could also be due to a Windows or JavaScript security problem.

You can usually download the correct driver from the web site of your video card manufacturer, or from the web site of the manufacturer of the chip on your video card.

Newest drivers

Newer versions of device drivers usually give better performance than older versions. Also check to see that you have the latest Windows service packs and TCP/IP stacks. BIOS should also be upgraded every other year or so for BIOS improvements, but your PC will probably be obsolete in two years anyway.

Memory and Disk

Windows disk caching schemes can help or hurt you. On one hand, the disk cache will make most writes to disk appear to be much faster, because the data is not being written out to disk synchronously with the write command, but queued up to be written some time later. This is known as "write behind" caching. This was the function of the `smartdrv.exe` line in your *AUTOEXEC.BAT* file under Windows 3.1. Disk cache can also be used to read ahead, anticipating that you will probably want more data from the disk after the current read completes. This is known as *read ahead* caching and can be turned on in Windows 95 like this: Control Panel → System → Performance → Advanced → File System → Read Ahead Optimization → Max.

On the other hand, some older hard disk drivers delay lower-priority interrupts while synchronizing disk with disk cache. This would be okay, except that low-priority COM port interrupts must be handled before the UART buffer overflows. If the disk has the COM port locked out and data is lost, the data must be retransmitted, slowing perceived network performance. The best solution is to get an updated driver for your disk, but you could also turn off the disk cache for a quick fix of network performance at the expense of disk performance. For Windows 3.1, the */X* switch to *smartdrv.exe* will turn off write behind caching. For Windows for Workgroups, the write behind cache is turned off by adding a line to the `[386enh]` section of *system.ini* that says `ForceLazyOff=C` where `C` is the letter of the disk drive. Turning off write behind cache should be only a temporary measure to help network performance until you can get updated disk drivers.

Being short on disk space will hurt Netscape performance, as it will have to search around for enough open space to write its cache and work files. Defragmenting your disk regularly with the *defrag* or *scandisk* utilities will help disk performance. You can defragment under Windows 95 like this: My Computer → Hard Drive → Properties → Tools → Defragment Now.

You can also help disk performance with 32-bit disk access, which you can turn on in Windows 3.1 like this: Control Panel → 386 Enhanced → VM → Change → 32-Bit Disk Access Enabled. Also, when you're about to shut off the machine, you can safely delete files that end in *.swp* or *.tmp* or that begin with a tilde (~), for additional disk space.

System Monitor

Windows 95 has a useful System Monitor tool that you can use to help pinpoint bottlenecks. To use it: Start → Run → enter `sysmon` → Add all of the options, but especially these:

- Memory Manager: Swapfile in Use
- Memory Manager: Free Memory
- Dial-Up Network Adapter: Buffer Overruns

You don't want to see any network buffer overruns. If there are overruns, you are not retrieving data from the UART as fast as it is filling up. This could be due to COM interrupts being starved for attention, an old UART (not likely on anything better than a 386), or the buffer being too small. You can change the buffer size under Windows 95 like this: Start → Settings → Control Panel → System → Device Manager → Modem → Connection → Port Settings → use slider to increase capacity of Receive Buffer. You do want to see a bit of headroom in Free Memory and not too much swapfile in use.

You can detect whether there is noise on your line by looking for Cyclic Redundancy Check (CRC) errors with *sysmon*: Start → Run → enter `sysmon` → Edit → Add → Dial-Up Networking Adapter → CRC errors. You shouldn't see any errors on a reasonably good line.

If your CPU is shown to be overloaded, the solutions are to upgrade to a faster CPU or reduce the number or kind of applications you are running.

Network Utilities

There are a few Unix network utilities available for Windows. Windows 95 has versions of *ping* and *traceroute*. To run *ping*: Start → Run → type command → at C:\ type `ping www.someserver.com`. *Traceroute* is called *tracert* under Windows; run it just like *ping*. There is also a third-party version of *traceroute* for Windows called QuickRoute, from Starfish's Internet Utilities 97 (*http://www.starfishsoftware.com/*).

MTU

If you are on a LAN, leave your MTU at the default of 1500 bytes. If you're dialing up and dealing with the Internet, try setting the MTU to 576 bytes. You may see some increased performance due to reduced fragmentation, as described previously in the Macintosh section. See *http://www.sysopt.com/maxmtu.html* for additional information on MTUs under Windows.

Active Desktop Problems

Active Desktop on Windows 95 consumes a lot of system resources and doesn't provide much essential benefit. You'll probably see better performance if you disable it: right-click on the desktop → Active Desktop → Customize → uncheck "View my active desktop as a web page".

Unix

Any Unix workstation can run a web client, but it is probably overkill to dedicate workstation hardware to web browsing when Macintosh or PC hardware will do just fine. Linux is the most popular version of Unix* for commodity PC hardware, partly because it is free along with all of the source code. Given identical PC hardware, you can get much better performance from Linux than from Windows, but until recently there has been little commercial software available for Linux because of its origins in the hobby world. Linux does have sufficient software to be a good web client because there is Netscape for Linux and a good Java Virtual Machine as well as all the usual Unix tools that make it relatively easy to figure out exactly what's going on in the system and the network: *top*, *vmstat*, *strace*, *traceroute*, *ping*, etc. Linux is a boon for anyone interested in operating systems. Here are a few of the things you can do to figure out exactly what is going on in your Linux web client:

- Start *top* and use `M` to sort all processes by memory usage. Leave it running. You'll probably see that Netscape is your single largest process and that it keeps growing as you use more of its features.
- Leave *netstat -c* running and you can see Netscape open connections as you retrieve web pages. Connections are open when *netstat* says they are in the ESTABLISHED state. The connections should all eventually be closed, except perhaps for connections such as the one to your POP mail server that polls for new mail.
- You can run Netscape with the *strace* command to see all system calls as they are made and understand a bit more of what Netscape is doing. Netscape's

* Though Linux is not Unix in a legal sense.

performance will probably be unbearably slow while you're tracing it, but it is instructive. You can also get the Netscape source code from *http://www.mozilla.org/*, if you want to try to modify the code to improve the browser's performance yourself.

- You can use *ping* to see what the latency to any given web server is. Most web servers are configured to respond to *ping*. If you *ping* yourself and the latency is significantly over 1 ms, this indicates that you have a poor implementation of IP. Pinging a web server on the same LAN should have a latency under 10 ms. If not, you may have an overloaded LAN or a very poor implementation of IP. If you're pinging a web server across the Internet, anything under 50 ms is good, but latency up to 200 ms is quite normal. When latency gets close to 1 second, you have a poor connection to that server. The *traceroute* command can tell you at exactly which router things start to slow down. This is useful for detecting the quality of an ISP.

If a server is not responding at all, try using *nslookup* or *dig* on the web server name. It may just be that you can't resolve the name, not that the web server itself is actually down. DNS service, like most Internet services, does not degrade linearly but instead hits a wall at a certain load. You can use *nslookup* with DNS servers outside your own organization, find the correct IP for a site, and then browse by using the IP address in place of the web server's name in the URL. You can even point your machine to the other DNS server permanently if your ISP's DNS server is not satisfactory, but this is rather rude unless you have some explicit agreement with the owner of the other DNS server. Consider running a DNS server on your own machine. A private DNS server will cache your frequent queries and provide much better response time for those queries. See *DNS and BIND*, by Paul Albitz and Cricket Liu (O'Reilly & Associates), for instructions on setting up your own DNS server.

Use the latest Linux kernel for the latest TCP/IP fixes and speed improvements, but don't feel the need to upgrade more than once or twice a year. Compile in only the drivers you need, and no others. For example, you don't need floating point emulation on most modern chips, because a floating point unit is now standard. Compile in the TCP retransmit interval to be one second, if it isn't already. Change the */etc/rc.d/** files to run as few daemons as necessary.

Key Recommendations

- Use your OS's tools to tell you if you're running out of RAM or mistransmitting or misreceiving packets, and if the server is up and responding.
- Increase receive buffer size if possible.
- Use the latest and best implementation of drivers and TCP/IP.

12

Server Operating System

In this chapter:

The operating system sits between the hardware and the web server software, translating the web server's request for services into hardware actions and delivering data from the hardware back up to the web server. The server hardware has a maximum performance that is fixed by its physical specifications. You can approach this maximum performance by appropriate configuration of the operating system. The question for web servers is how to configure the operating system to take requests from the network, find the correct file to return or run the correct program, and push the result back out to the network, all as fast as possible.

Unix and the Origin of the Web

Networking has been central to the development of Unix. Unix was originally developed around 1970 as a research project by AT&T Bell Laboratories, building on multiuser and multitasking ideas from the Multics government research project of the 1960s. Since AT&T was barred from selling software because it was the U.S. telephone monopoly, it allowed universities to use the source code for education and research. The University of California at Berkeley, in particular, continued development on the TCP/IP implementation in the Unix kernel. The Berkeley group introduced enough changes that Unix was for about ten years split into two main camps: Berkeley Unix and AT&T Unix. Around 1988, these were merged into

System V Release 4 (SVR4) Unix, but derivatives of the two original camps continue to exist, such as BSDI and SCO.

The HTTP protocol and the first web server were both developed on Unix platforms and are natural outgrowths of previously existing Unix work. HTTP inherited many characteristics of FTP, but extended FTP with automated requests. The first popular web server software, the University of Illinois' *httpd*, was a classical Unix daemon. At that point, all Unix machines had TCP/IP networking ability and FTP by default, so the technological jump from the existing file transfer protocol to the web protocol was much smaller than would be expected from its subsequent impact on the world. As almost all Unix machines were networked, it was a trivial matter to retrieve *httpd* from the University of Illinois and start it running on your local machine. There was no charge for the software, since it came from a tax-sponsored research project. Web usage exploded because the Internet was primed and ready for a new and simpler front end.

Web servers and clients were quickly ported to other platforms, such as Windows, the Macintosh, and even mainframes and the AS/400, but the majority of web servers remained on Unix platforms. Unix proved superior in stability and performance because of its longer development history and open nature. Simply put, more people have been able to contribute to and debug Unix for longer than any other operating system. Development of proprietary platforms has been limited by the number of paid employees of any one company, while Unix benefited from the work of the entire academic and Internet community. An interesting hybrid approach between proprietary and public code is Netscape's desire to capture the creativity of the Internet community by releasing the source code to its browser for inspection and improvement. The Netscape browser source code is available at *http://www.mozilla.org/*.

Unix Flavors

It might be said that the Unix market enjoys competition, but it could also be said that the Unix market suffers from competition. The problem has been that individual vendors have created their own versions of Unix that are incompatible with all other versions. This means that a binary executable for one version is usually incapable of running on any other Unix, even if the hardware is identical.

The problem is gradually being solved by several forces. First, the SVR4 standard has become dominant. An old joke goes that Sun succeeded in uniting the Unix industry, but unfortunately it united in a coalition against Sun. Nonetheless, SVR4 is now the de facto standard. Programs written to the SVR4 standard have source-level portability to other SVR4 systems, meaning that you should be able to recompile the code for each system without changes. Second, the increasing popularity of Sun Microsystems' Solaris implementation of SVR4 means that Solaris is becoming the de facto standard for binary-level portability, while the future of some other versions of Unix is in doubt. Finally, as more applications are written in Java, small differences between operating systems will become irrelevant. Systems will be forced to compete on performance and cost, not compatibility. Here are the major versions of Unix used for web services:

Solaris

Solaris is Sun Microsystems' (*http://www.sun.com/*) version of Unix. When the Berkeley-derived SunOS was made compatible with SVR4, its name was changed to Solaris. There are versions of Solaris for Sun's SPARC hardware as well as for the Intel x86 architecture. Solaris on the SPARC hardware has the largest Unix market share, so you can be reasonably sure that any commercial multiplatform Unix software will be ported to Solaris first, debugged, and probably optimized for Solaris. This is important because software vendors do not have the resources to port and optimize for every version of Unix. Solaris has the largest market share for web servers, partly due to its performance and reliability, but also because Solaris has the largest share of the computer science education market. Students who learn to program and administer on Solaris systems tend to keep using it once they go out into the business world. One of Solaris's strengths is its efficiency in memory allocation, which happens so frequently that it has a large impact on overall system performance. Memory allocation is accelerated by hardware called with special opcodes available only to the OS, not to applications.

Solaris 2.6, the latest version as of this writing, comes with default parameters more appropriate to web services than previous versions, so a web server run out of the box on Solaris 2.6 will not need much tuning. Solaris 2.6 also includes many fundamental improvements that increase web server performance, such as a more efficient TCP/IP stack with more networking code in the kernel rather than user space, and better use of multiple CPUs. Sun claims a 350% increase in web server performance on Solaris 2.6 over Solaris 2.5 on multiprocessor systems.

Digital Unix

Digital Unix is known for extremely good I/O performance because of efficient disk driver and TCP/IP implementations, which make it very suitable for web serving. Digital Unix is already 64-bit, so it is ahead of most other Unixes there. It is scalable up to very large SMP systems, like the showpiece AltaVista search engine.

Linux

Linux is a free Unix kernel originally written for the Intel architecture, but since ported to the Alpha, PowerPC, and even Sun's SPARC. The project was started by Linus Torvalds of Finland. Remember that Linux is just a kernel and not the utilities that allow user interaction with the kernel. Linux comes in distributions from various organizations, usually including GNU tools such as the *gcc* compiler and *bash* shell, as well as a version of the X Window System called XFree86.

The scalability of Linux is not as mature as that of Solaris. Until recently, you were limited to 256 simultaneous processes and there was no multiprocessor support. Still, Linux is as robust and high-performance as many commercial versions of Unix. There is an information page on Linux at *http://www.li.org* (and a competing page at *http://www.linux.org*). Excellent free support is available via the many Linux Usenet newsgroups, and paid support is supplied by companies such as Cygnus. Now that Oracle has been ported to Linux, Linux is being accepted by the business community as a serious platform. There are several high-volume web sites that run the free Apache server on Linux.

Irix

Irix is the version of Unix developed by Silicon Graphics for the company's high-performance graphics workstations. It runs only on Silicon Graphics hardware. Silicon Graphics hardware has been optimized for graphics manipulation, but many of the graphics features, such as very quick memory and disk access, are also useful for high-performance web serving. Unfortunately, Irix is not as well supported as Solaris by software vendors like Netscape, so there is often a delay before the Irix version is available. See *http://www.sgi.com/* for extensive information on tuning Irix for web services.

BSD

The Berkeley Standard Distribution of Unix (BSD) is similar to Linux in that it tends to be favored at smaller web sites and runs on Intel x86 hardware. Unlike Linux, BSD development includes not just the kernel but all of the utilities and documentation as well. BSDI (*http://www.bsdi.com/*) and FreeBSD (*http://www.freebsd.com/*) are derivatives of BSD, and have continued the development of Berkeley Unix.

Mach OS

The list of Unix versions used for web services would not be complete without Mach OS. Mach was developed at Carnegie-Mellon and provided the foundation for the NextStep operating system, on which the original implementation of HTTP was built by Tim Berners-Lee at CERN in Switzerland.

Let's take a look at how Unix works, keeping in mind the performance perspective.

Processes and the Kernel

Unix work is divided up into processes, which you can think of as tasks to be done. Each process has a unique process ID, which is simply an integer, and an are an indication of whether you have enough memory. You want to keep both of them low. Here's some sample output from *vmstat*:

```
% vmstat 1

 procs     memory              page            disk          faults      cpu
 r b w   swap   free  re   mf pi po fr de sr s2 s3 s4 s1   in   sy   cs us sy id
 0 0 0    352    888   0  534 18  3  7  0  0  0  1  0  1  271    1  418  3  6 91
 6 0 0 192960 111528   0 2139 0   0  0  0  0  0  0  0  0  488 2780 1106  4 23 73
 1 0 0 196128 112136   0 4412 0   0  0  0  0  0  0  0  0 1086 3193 2511  7 33 60
 0 0 0 194016 111416   0 3419 0   0  0  0  0  0  0  0  0  689 1977 1722  2 22 76
 0 0 0 193664 111296   0 4319 0   0  0  0  0  0  0  0  0  890 2597 2231  2 28 69
 1 0 0 195608 111992   0 4100 0   0  0  0  0  0  0  0  0  887 2341 2210  4 23 72
 0 0 0 196480 112256   0 1234 0   0  0  0  0  0  0  0  0  390  924  785  2  7 92
 0 0 0 194720 111720   0 1616 0   0  0  0  0  0  0  0  0  360 1095  738  2  8 91
 0 0 0 189440 110016   0 2900 0   0  0  0  0  0  0  0  0  701 1767 1732  2 20 78
 1 0 0 195040 111768   0 5486 0   0  0  0  0  0  2  0  1 1046 3969 2729 10 32 58
 0 0 0 194280 110808   0 4147 0   0  0  0  0  0  0  0  0  787 2368 1984  2 27 70
 2 0 0 191816 110152   0 5556 0   0  0  0  0  0 31  0  0 1122 3268 2840  4 38 58
 0 0 0 194368 111528   0 3784 0   0  0  0  0  0  0  0  0  750 2157 1914  4 18 78
 2 0 0 194776 112064   0 4684 0   0  0  0  0  0  0  0  0  893 2607 2316  4 28 67
 0 0 0 195424 111888   0 2366 0   0  0  0  0  0  1  0  0  526 1433 1240  4 16 81
 0 0 0 195424 111888   0 3650 0   0  0  0  0  0  0  0  0  758 2381 1896  2 24 73
 0 0 0 194368 111592   0 4166 0   0  0  0  0  0  0  0  0  830 2405 2108  3 24 73
 0 0 0 194368 111528   0 3430 0   0  0  0  0  0  0  0  0  676 2116 1724  2 22 76
 0 0 0 194360 112000   0 3988 0   0  0  0  0  0  0  0  0  772 2266 1896  2 26 71
```

sar

sar provides much the same information as *vmstat*, but can save data in a compact binary format suitable for long-term statistics gathering. *sar* itself provides display options for the data collected. It is much more complete and flexible than *vmstat*. Here's sample output from *sar*:

```
% sar -c 5 5

SunOS pokey.patrick.net 5.5.1 Generic_103640-12 sun4u    02/17/98

15:03:46 scall/s sread/s swrit/s  fork/s  exec/s rchar/s wchar/s
15:03:51    2418       5      84   79.64    0.60    1126     537
15:03:56    1703       3      58   57.80    0.00     478     505
15:04:01    1676      10      57   51.70    0.00     640     523
15:04:06    2553      14      93   84.83    0.00     739     559
15:04:11    1807       4      60   56.80    0.60     965     556
```

Unix Versus NT as the Web Server OS

All of this is not to say there is no competition for Unix as a web server platform. It is simple to set up a web server on any Windows or Macintosh machine and get reasonable performance, as long as the load is light. But for heavy loads, the only competitor to Unix for a web server platform is Windows NT. The creators of NT applied many of Unix's features, such as the concepts of a kernel, user processes, and preemptive multitasking, to the design of NT. Let's take a look at what each OS has to offer.

NT Pros and Cons

NT has the traditional Microsoft advantages of close integration with other Microsoft products, a consistent look and feel, and GUI rather than command line administration for those who don't like to type commands and don't want or need a fine level of control. NT has the ability to run some legacy Windows applications. NT can run on cheap commodity PC hardware, but so can many versions of Unix.

NT does not have especially good performance or scalability for web serving. See the article "The Best OS for Web Serving: Unix or NT?" by Barry Nance (*Byte Magazine*, March 1998) for one experiment confirming this. More important, NT is very unstable compared to Unix, frequently crashing or requiring reboots, which is a serious drawback to using it for important sites. NT comes with no remote administration and no multiuser mode.

Running a high-performance web site on PC hardware is also difficult. Scalability is limited by the PC's legacy I/O architecture, which was never intended for large loads. By some estimates, you cannot handle more than about 250 concurrent transaction processing users on even the best PC hardware. PC hardware is also generally less reliable than true workstation hardware, because it is built for the mass market, where cost rather than reliability is the driving factor.

Microsoft also has some credibility problems. Their "Wolfpack" scalability demonstration used the easily fudged debit-credit benchmark, rather than the more reliable TPC-C and TPC-D transaction processing benchmarks. And Microsoft has been accused of misrepresenting the capability of NT workstation relative to its NT server edition. The cheaper NT workstation was limited to 10 open connections, allegedly because it couldn't handle more than that; users were supposed to pay much more for a supposedly higher-performance version. A binary *diff* showed that the executables were identical, except for a few bytes that were presumably a switch allowing more connections.

Unix Pros and Cons

Unix operating systems are very robust, usually capable of running for many months and sometimes several years without rebooting. Equally important is that Unix has much better scalability than NT, allowing a small site to grow by adding hardware such as CPUs and I/O controllers to existing machines rather than requiring multiple new machines to handle increasing load. Finally, Unix has better performance, partly because it is usually run on better hardware rather than commodity PCs.

A large majority of high-performance web sites run on some version of Unix, so the issues are well worked out.

A professional Unix system will typically have a higher entry-level cost than NT, even if the price/performance ratio at that level is better. If cost is a big consideration, there are many versions of Unix that will run on the same commodity PC hardware as NT and provide the same or better performance, such as Solaris x86, Linux, and BSDI. Pricing for Solaris is similar to that for NT.

Unix also requires more highly trained system administrators, although GUI tools for the Unix-averse are starting to appear, such as Sun's Netra product line.

Key Recommendations

- Don't write back the file access time. Either mount a filesystem as read-only or change the filesystem source and recompile.
- Use short paths.
- Don't use symbolic links.
- Don't run a window system on the server; use terminal mode.

In this chapter:
- *CGI Internals and Performance Problems*
- *General CGI Tips*
- *CGI Language-Specific Optimization Tips*
- *Daemonize It*
- *CGI Database Access Performance*
- *Key Recommendations*

15 CGI Programs

While plain HTML documents stored on your web server can contain whatever text you like, that text is static. Everyone who requests that document through a web browser will get exactly the same document. Quite often, however, you'd like to customize the response for a particular user. For example, a retail chain might want to query a database for the user and return the address of the branch store nearest the user. One way to do this is to have the web server run a program that will query the database and format the result in HTML. The first widely available method for incorporating dynamic content like this into web pages was the Common Gateway Interface (CGI) standard. CGI was introduced as part of the web server developed at the National Center for Supercomputing Applications (NCSA).

CGI provides a standard interface between web servers and programs that can generate HTML or other web content. CGI got its start as a literal gateway between web servers and older Unix programs that send their output to the terminal, but it quickly became clear that the real value of CGI was that it could provide a web interface to almost any software. Programs started by the web server using the CGI interface are referred to as *CGI programs*, or just *CGIs*, though CGI is technically the interface and not the programs that use the interface.

The definitive description of CGI 1.1, the current version, is on the Web at *http://hoohoo.ncsa.uiuc.edu/cgi/*. From here on, I will assume that the reader understands how to write at least a simple CGI program. If you'd like an excellent tutorial on CGI, read *CGI Programming on the World Wide Web*, by Shishir Gundavaram (O'Reilly & Associates). If you already know CGI programming and would like to keep up with the latest developments, read the Usenet newsgroup *comp.infosystems.www.authoring.cgi*.

Server APIs, such as the Apache API, NSAPI, and ISAPI, are a huge performance win over CGI at the expense of portability. Once you've written a program for a server's API, there is a cost to porting it to another server (unlike CGIs or Java servlets), but on the other hand, the APIs have no parameter parsing and no separate CGI process. Programs written using the API run as part of the web server process. Note that some databases are also web servers, which eliminates even the overhead of the separate *httpd* process.

CGI Internals and Performance Problems

Though the CGI mechanism for generating dynamic web content is very versatile, the basic structure of CGI limits its performance. The main performance penalty is that a new instance of the program is executed for each user's request. This process exits immediately after sending its output back to the web server. If the CGI program opens a database connection, the database connection must be reopened for the next instance of the CGI. This load on the operating system severely limits the number of CGI requests that can be serviced per second. CGI execution time is likely to be the bottleneck under any but the lightest loads. CGIs typically take far more CPU and other resources than serving HTML pages. Another inefficiency is that CGIs that are hit more than once throughout the day, say for stock quotes or weather, return mostly unchanged HTML and graphics with only a little bit of new content. This is very wasteful of network bandwidth.

Let's take a closer look at the sequence of events in starting a CGI program and where the performance problems are. When a CGI request comes in, the web server must parse the input URL and the request headers, recognize that the user desires to execute a CGI program, and begin the CGI with the `fork()` and `exec()` system calls. Parsing and `fork()` and `exec()` account for much of the cost of CGI. The server sets up the environment variables and the standard I/O for the child process, then it begins to write the URL-encoded data to the CGI's standard in. The CGI reads the data, stopping when it has read the number of bytes specified in the CONTENT-LENGTH environment variable. The CGI may also read URL-encoded command-line arguments, which are given by placing them after the script name in the URL like this: *http://www.nowhere.com/script.cgi?cmd_line_arg*.

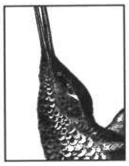
It is then up to the CGI to decode the data and decide what to return to the browser. This is the meat of the CGI and varies widely in complexity. When the CGI is done, it outputs its results to the web server, which adds HTTP headers and forwards everything to the browser.* The CGI then exits. There is a rather dated

* Alternately, many web servers allow the CGI itself to provide all the headers and communicate directly with the client's browser.

World Wide Web/Internet

O'REILLY™

Web Performance Tuning



As long as there's been a Web, people have been trying to make it faster. The maturation of the Web has meant more users, more data, more features, and consequently longer waits on the Web. Improved performance has become a critical factor in determining the usability of the Web in general and of individual sites in particular.

Web Performance Tuning is about getting the best possible performance from the Web. This book isn't just about tuning web server software; it's also about getting optimal performance from a browser, tuning the hardware (on both the server and browser ends), and maximizing the capacity of the network itself.

Web Performance Tuning hits the ground running, giving concrete advice for quick results—the "blunt instruments" for improving crippled performance right away. The book then shifts gears to give a conceptual background of the principles of computing performance. The latter half of the book examines each element of a web transaction—from client to network to server—to find the weak links in the chain and suggest ways to strengthen them.

Tips include:

- Using simultaneous downloads to locate bottlenecks
- Adjusting TCP for better web performance
- Reducing the impact of DNS
- Upgrading device drivers
- Using alternatives to CGI
- Locating the web server strategically
- Minimizing browser cache lookups
- Avoiding symbolic links for web content

This book is for anyone who has waited too long for a web page to display, or watched servers slow to a crawl. It's about making the Web more usable for everyone.

ISBN 1-56592-379-0
US $32.95
CAN $48.95
90000
9 781565 923799
6 36920 92379 4